WML Reference

Version 1.1

http://www.forum.nokia.com
Product number: SDK-01-000-004
September 1999

WML Reference

Version 1.1

Product number: SDK-01-000-004

Copyright© Nokia Corporation 1999. All rights reserved.

We welcome and consider all comments and suggestions. Please send them to:

Nokia Group Finland
P.O. Box 226,
FIN-00045 NOKIA GROUP

Tel. +358 9 180 71
Fax. +358 9 656 388

Internet mail address:
wap.sw.developer@nokia.com

http://www.forum.nokia.com

This document is part of the Nokia Wireless Application Protocol Toolkit. The contents of this guide are based on the Wireless Application Protocol Wireless Markup Language Specification Version 1.1 (WAP WML Version16-June-1999).

Reproduction, distribution or transmission of part or all of this documentation in any form without the prior written permission of Nokia is prohibited.

The content of this documentation may be changed without prior notice.

"Nokia," the arrows symbol and Nokia's product names are trademarks of Nokia.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

Portions of the Nokia WAP Toolkit contain technology used under license from the World Wide Web Consortium and are copyrighted by the World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University).

Contents

Introduction

This guide introduces the Wireless Markup Language (WML). WML is a markup language based on the Extensible Markup Language (XML) and was developed for specifying content and user interface for narrowband devices such as cellular phones and pagers.

WML is designed to work with small, wireless devices that have four characteristics:

Small display screens with low resolution. For example, most mobile phones can only display a few lines of text, and each line can contain only 8–12 characters.

The input devices have limited capacity, or are designed for a special purpose. A mobile phone typically has a numeric keypad and a few additional function-specific keys. More sophisticated devices may have software-programmable buttons, but not a mouse or other pointing device.

The computational resources are often limited by a low power CPU, a small memory and power constraints.

The network offers low bandwidth and high latency. Devices with 300 b/s to 10 kbit/s network connections and 5-10 second round-trip latency are not uncommon.

The characteristics of WML can be grouped into four major areas:

WML offers text and image support, and has a variety of formatting and layout commands.

WML cards are grouped into decks. A WML deck is similar to an HTML page in that it is identified by an URL and is the unit of content transmission.

WML offers support for managing navigation between cards and decks, and includes commands for event handling. These can be used for navigating or executing scripts. WML also supports anchored links, similar to those used in HTML version 4.

Parameters can be set for all the WML decks using a state model. Variables can be used in place of strings and are substituted at runtime. Setting parameters this way allows network resources to be used more efficiently.

All of the WML information is transmitted in encoded format over the wireless networks.

Typographical conventions

The following conventions are used in this guide.

Notation	Explanation
`Courier`	Text that appears onscreen, program code, file and directory names, function names. Optional attributes in WML element syntax.
`Courier Bold`	WML tags and mandatory attributes, Uniform Resource Locators.
`Courier Italic`	Parameter values (for example, `title=`*`NMTOKEN`*), variables in commands and other types of specialized language.
?	The specified element can be included zero or one time inside the described parent element. This convention is used in the *Contained elements* section of the WML element descriptions. For example, in the description of the `WML` element, the contained element `head ?` means that you can use the `head` element only once inside the `WML` element, or that you can leave the `head` element out.
+	The specified element must be included at least once inside the described parent element. This convention is used in the *Contained elements* section of the WML element descriptions. For example, in the description of the `wml` element, the contained element `card +` means that you must include at least one `card` element inside the `wml` element.
*	The specified element can be included any number of times inside the described parent element. This convention is used in the *Contained elements* section of the WML element descriptions. For example, in the description of the `template` element, the contained element `do *` means that you can include the `do` element any number of times inside a `template` element, or that you can leave the `do` element out.

Notation	Explanation
|	Separates alternative items. For example, `<img align="left|right|center">` means that the image can be aligned left, right or center.

The element descriptions have the following standard content:

Description: This section provides a short description of the element and its use in WML applications.

Contained elements: This section lists the elements that can be included in the current element.

Syntax: This section explains the attributes of the element.

Example: This section provides a simple example illustrating how the element and its attributes are used. The examples have a running numbering.

Related documents

The following documents contain additional information on the Nokia WAP Toolkit and the Wireless Application Protocol. The web address provided after each document specifies the Internet location where the document can be obtained.

Documents included in the Nokia WAP Toolkit

Nokia WAP Toolkit Getting Started

This guide provides basic information on the Nokia WAP Toolkit and the Wireless Markup Language, and provides instructions on installing and using the product.

Nokia WAP Toolkit Developer's Guide

This guide provides information on the Nokia WAP Toolkit and the Wireless Markup Language for developers who want to create their own wireless services on the WAP platform.

WMLScript Reference

This guide provides reference information on the WMLScript language. It introduces the WMLScript and its standard libraries.

Other references

Wireless Markup Language Specification.
WAP Forum, 16-June-1999.
http://www.wapforum.org/

WMLScript Specification.
WAP Forum, 16-June--1999.
http://www.wapforum.org/

Wireless Application Environment Specification.
WAP Forum, 16-June-1999.
http://www.wapforum.org/

Wireless Application Protocol Architecture Specification.
WAP Forum, 16-June-1999.
http://www.wapforum.org/

Wireless Session Protocol Specification.
WAP Forum, 16-June-1999.
http://www.wapforum.org/

The Unicode Standard: Version 2.0.
`http://www.unicode.org`

Extensible Markup Language (XML).
W3C Proposed Recommendation, 10-February-1998, REC-xml-19980210.
`http://www.w3.org/TR/REC-xml`

RFC2045: Multipurpose Internet Mail Extensions (MIME)
Part One: Format of Internet Message Bodies.
`http://www.alternic.net/info/rfcs/2000/rfc2045.txt.html`

RFC2068: Hypertext Transfer Protocol - HTTP/1.1.
`http://www.w3.org/Protocols/HTTP/1.1/`
`draft-ietf-http-v11-spec-rev-03.txt`

RFC2396: Uniform Resource Identifies (URI): Generic Syntax

Core WML data types

This chapter describes the core data types included in WML, as specified in the WAP WML Specification Version 1.1. These data types are used in the WML element descriptions throughout this guide.

Character data

All character data in WML is defined in terms of XML data types. The following table summarizes the character data types:

Data type	Explanation
CDATA	Text which may contain numeric or named characters. **CDATA** is used only in attribute values. Examples: "$(**value**)" name="**value**"
PCDATA	Parsed CDATA. Text which may contain numeric or named characters. This text can also contain tags. **PCDATA** is used only in elements. Example: Text written <**big**> IN CAPS </**big**>.
NMTOKEN	A name token, containing any mix of numbers, letters, and some punctuation characters ".", "-", "_", and ":". Note that you can start a name token with a punctuation character. A name token cannot be used in variable reference or element names. Examples: "text" _card1 a.name.token .a-perfectly-valid.name.token

Data type	Explanation
id	An unique identifier for the element. Example: <card id="card1"/>

Length

The %length type may be specified either as an integer representing the number of screen pixels, 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. And for heights, the value "50%" means half of the available vertical space.

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.

Name	Type	Usage
%length	CDATA	[0-9] for pixels or [0-9] + % for percentage length.

Example [1.]

For example, the values of the hspace and vspace attributes are %length.

```
<img src="bitmaps/moon.wbmp" hspace="1" vspace="1"/>
<img src="bitmaps/moon.wbmp" hspace="2%" vspace="2%"/>
```

Vdata

The %vdata type represents a string that may contain variable references. This type is only used in attribute values.

Name	Type	Usage
%vdata	CDATA	Attribute value possibly containing variable references.

Example [2.]

```
<card id="card1" title="$(showme)">
```

Flow, inline and layout

The %flow type represents "card-level" and the %inline type "text-level" information. In general, %flow is used anywhere general markup can be included. The %inline type indicates areas where only pure text or variable references are handled.

Name	Type	Usage
%layout	br	Text layout, such as line breaks.
%inline	%text %layout	Indicates areas where only pure text or variable references are handled.
%flow	%inline img anchor a table	Covers card-level elements, such as text and images.

Example [3.]

The data types %flow and %inline are used for general text that may have formatting attributes such as italics, underline, and bold.

```
<em>
   An emphasized line.
   <big>
      A big and emphasized line.
   </big>
</em>
A line with no text formatting.
```

Text

The %text type can include the following entities:

Name	Type	Usage
%text	#PCDATA %emph	Indicates text that contains formatting.

Example [4.]

```
A line with plain text.
<em>
   <strong>
      A line with strong emphasis.
   </strong>
</em>
```

Href

The %href type refers to either a relative or an absolute Uniform Resource Identifier (URI).

Name	Type	Usage
%href	%vdata	URI, URL, or URN designating a hypertext node. May contain variable references.

Example [5.]

```
<go href="http://wapforum.org/"/>
<go href="file:///d:\dir\file.wml"/>
<go href="app.wml"/>
```

The values of intrinsic events are URLs:

```
<card onenterforward="#card2"/>
```

The src attribute of the img element is an URL:

```
<img src="earth.wbmp"/>
```

Boolean

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

Name	Type	Usage
%boolean	true | false	Logical value of true or false.

Example [6.]

```
<card newcontext="true"/>

<do optional="true" type="accept"/>
```

Number

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

Name	Type	Usage
%number	NMTOKEN	A number, from [0–9].

Example [7.]

```
<select tabindex="2"/>
<input name="setvar" size="4" maxlength="20" tabindex="3"/>
```

Emphasis

The %emph type covers text formatting tags described in the following table.

Name	Type	Usage
%emph	em | strong | b | i | u | big | small	Text formatting, for example italics or underlining.

Example [8.]

```
<em>
   An emphasized line.
   <big>
      A big and emphasized line.
   </big>
</em>
```

WML character set

WML is an XML language inheriting the XML document character set. In WML, a document character set is the set of all logical characters that a document type may contain, for example the letter 'T' and a fixed integer identifying that letter. A WML 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. Currently, this character set is identical to Unicode 2.0. For more details on the character sets, refer to the *XML Specification* and the *ISO10646 Specification*.

WAP supports the following Unicode subset document character sets:

UTF-8

UCS Transformation Format 8 is used as a transfer encoding to transmit the international character set. UTF-8 is a file safe encoding which avoids using byte values which have special significance during the parsing of pathname character strings. UTF-8 is an 8-bit encoding of the characters in the UCS. Some of UTF-8's benefits:

— It is compatible with 7-bit ASCII, so it does not affect programs that give special meanings to various ASCII characters

— It is immune to synchronization errors; its encoding rules allow for easy identification.

— It has enough space to support a large number of character sets.

ISO-8859-1

The ISO-8859-1 character set is an extension of the ASCII character set and can be used to represent all western European languages. Also known as ISO Latin-1, ISO-8859-1 is very similar to the ANSI character set used in Windows, though the two are not identical. The HTTP protocol presumes the use of ISO Latin-1 unless another character set is specified. This means that to represent non-ASCII characters on a WML page, you need to use the corresponding ISO Latin-1 code.

UCS-2

UCS-2 is the 2-byte (16-bit) encoding of the Universal Multiple-Octet Coded Character Set (UCS) defined in ISO 10646. The character code values of UCS-2 are identical to those of the Unicode character encoding standard published by the Unicode Consortium.

Reference Processing Model

WML documents may be encoded with any character encoding as defined by the HTML 4.0 Specification.

Character encoding of a WML document may be converted to another encoding (or transcoded) to better meet the user agent's characteristics. However, transcoding can lead to loss of information and must be avoided when the user agent supports the document's original encoding. Unnecessary transcoding must be avoided when information loss will result. If required, transcoding should be done before the document is delivered to the user agent.

User agents must determine the character encoding of a WML document according to the following precedence (listed highest to lowest):

Based on the "charset" parameter of the "Content-Type" transport header (e.g. WSP or HTTP).

Based on meta-information placed within the document (e.g., the charset parameter in an `http-equiv` meta element).

Based on the encoding on the XML declaration.

Based on some other heuristics or user settings. For example, if content type is text based (i.e., `text/vnd.wap.wml`), the charset may be assumed to be US-ASCII; otherwise, the default content encoding can be assumed (i.e., UTF-8).

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

User agents must correctly map to Unicode all characters in any character encoding that they recognize, or they must behave as if they did.

Any processing of entities is done in the document character set.

A given implementation may choose any internal representation that is convenient.

Character entities and special characters

A given character encoding may not be able to express all characters of the document character set. For such encoding, or when the device characteristics do not allow users to input some document characters directly, you may use character entities. Character entities are a character encoding-independent mechanism for entering any character from the document character set.

WML supports both named and numeric character entities. Note 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 the notation `&#123;` 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 `&#123;`

Hexadecimal numeric character entities, such as `&#x12;`

The following table illustrates the seven named character entities that are particularly important in the processing of WML.

Entity	Notation	Explanation
`quot`	`&#34;`	quotation mark
`amp`	`&#38; #38;`	ampersand
`apos`	`&#39;`	apostrophe
`lt`	`&#60;`	less than

Entity	Notation	Explanation
gt	>	greater than
nbsp		non-breaking space
shy		soft hyphen (discretionary hyphen)

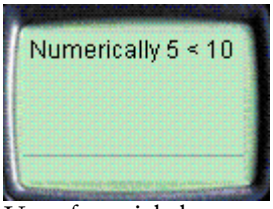
! **Note**: The semicolon (;) is part of the escape sequence for a special character.

Example [9.]

To include a special character, simply use the escaped notation described in the above table. For example, the following code includes a *less than* character (<) in the escaped form <.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <card id="Card_1">
      <p>
      Numerically 5 &#60; 10
      </p>
   </card>
</wml>
```

The deck generates the following user interface in the user agent:



Use of special characters.

WML syntax

WML inherits most of its syntactic constructs from XML. For detailed information on the syntactical issues of XML, refer to the *XML Specification*.

Entities

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

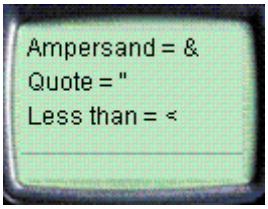
the named entity `&`. All entities begin with an ampersand and end with a semicolon.

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

Example [10.]

```
<card id="card1">
   <p>
   Ampersand = &amp; <br/>
   Quote = &quot; <br/>
   Less than = &lt; <br/>
   </p>
</card>
```

The deck generates the following user interface in the user agent:



Example of entities.

Tags

A tag is a language element descriptor. A tag describes an element and contains an element type name and a unique identifier. A tag could also include attributes describing other properties.

WML consists of content surrounded by formatting tags, each enclosed in a pair of angle brackets, < and >.

`<tag>` This starts an element. The start tag can contain attributes.

`</tag>` This ends an element.

`<tag/>` This is an empty element, for example `<br/>`, indicating a line break.

Elements

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

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

- or-

```
<tag/>
```

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

Attributes

WML attributes specify additional information for an element. Attributes are always specified in the start tag of an element. For example,

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

Note that attribute names must be lowercase.

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

Some attributes are mandatory, and these are emphasized in **bold** in the element descriptions and attribute tables later in this guide. For example, the `go` element requires the `href` attribute:

```
<go href="http://www.acme.com"/>
```

Comments

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

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

Comments are intended to be used by the WML author and are not displayed to the user by the user agent. Note that WML comments cannot be nested.

Variables

Parameters can be set for WML cards and decks 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, that is, 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.

For information on variable conversions, see "Substituting variables" on page 54.

Example [11.]

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

The WML code could take the following form:

```
This is a $$ character.
The value is $(amount)$$.
```

In the user agent, this would be displayed as:

```
This is a $ character.
The value is 5000$.
```

Case sensitivity

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

Example [12.]

The following attribute values are all different:

```
id="Card1"
id="card1"
id="CARD1"
```

Cdata section

`Cdata` sections are used to escape blocks of text and are legal in any `pcdata` section, for example, 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.

For more information on `cdata` sections, refer to the *XML Specification*.

First steps in WML

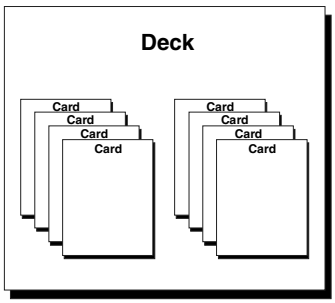
This chapter gives you an overview of the WML language and its most important elements, illustrated by code examples.

Card and Deck

All WML information is organized into a collection of cards and decks. Cards specify one or more units of user interaction, for example 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 information requested, makes choices and moves on to another card.

Cards are grouped together into decks. A deck is the smallest unit of WML that a server can send to a user agent.

The following figure illustrates the card and deck metaphor:



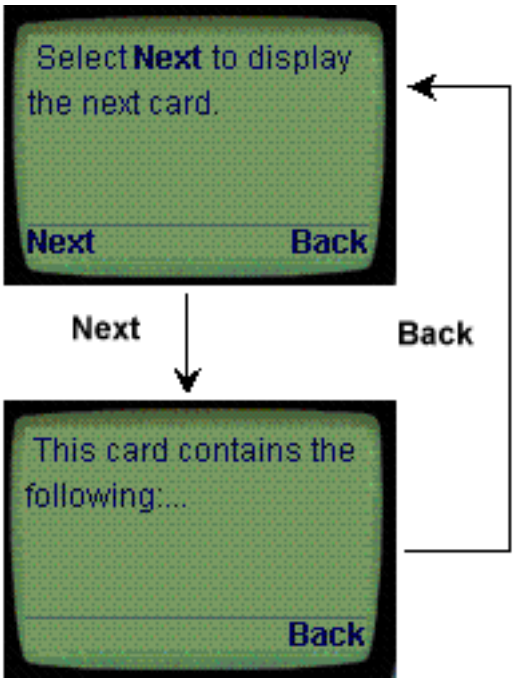
WML deck and cards.

The first WML example [13.]

Our first WML example introduces a simple WML deck containing two cards. When the user presses the ACCEPT soft key labeled “Next,” the user agent navigates to the second card of the deck and displays its content.

```
<?xml version="1.0"?>                                          <!-- 1 -->
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">                     <!-- 2 -->
<wml>                                                          <!-- 3 -->
   <card id="First_Card">                                      <!-- 4 -->
      <do type="accept" label="Next">                          <!-- 5 -->
         <go href="#Second_Card"/>                             <!-- 6 -->
      </do>                                                    <!-- 7 -->
      <p>                                                      <!-- 8 -->
      Select <b>Next</b> to display the next card.             <!-- 9 -->
      </p>                                                    <!-- 10 -->
   </card>                                                    <!-- 11 -->

   <card id="Second_Card">                                    <!-- 12 -->
      <p>                                                     <!-- 13 -->
         This card contains the following:...                 <!-- 14 -->
      </p>                                                    <!-- 15 -->

   </card>                                                    <!-- 16 -->
</wml>                                                        <!-- 17 -->
```

The deck generates the following user interface in the user agent:



A deck containing two cards.

The following is a line-by-line explanation of this example:

1 The first two lines define the document prologue that identifies the XML subset. This prologue must be included at the beginning of every WML deck, that is, before each `<wml>` tag..

3 The third line defines the header of the WML deck. All WML decks must begin with a `<wml>` tag and end with a `</wml>` tag. For more information, see "wml element" on page 24.

4 The fourth line of the deck specifies the header of the first card. Like decks, cards require begin and end tags, for example, `<card>` and `</card>`.

   Most WML elements allow you to specify attributes. Attributes are entered in the form `attribute=value`, where `attribute` is the attribute name and *`value`* is an alphabetic or numeric value that you specify.

   For more information, see "card element" on page 26.

5 The fifth line defines an action, which specifies what the user agent should do when the user presses a specified function key. The `type` attribute identifies the key (ACCEPT) and the `label` attribute a label name (Next) for the specified key.

6 The sixth line specifies the action related to the specified key. The `href` attribute identifies the target URI destination, for example, the card named `Second_Card`.

Template

A WML deck may contain a template that defines deck-level characteristics that apply to all cards of a deck. In an individual card, you can override these characteristics by specifying the same characteristics under the same name.

For more information on templates and using them, see "template element" on page 29.

For more information on overriding template tasks in individual cards, see "Card and deck task override" on page 44.

WML and URLs

The World Wide Web is a network of information and devices, where three areas of specification ensure widespread interoperability:

Uniform Resource Locators (URLs) provide a standard for naming any network resource.

Standard protocols (for example, HTTP) for transporting information.

Standard content types (for example, HTML and WML).

WML uses the same reference architecture as HTML and the World Wide Web. Content is named using URLs and is retrieved over standard protocols that have

HTTP semantics, such as Wireless Session Protocol (WSP). URLs and the character set used to specify URLs are defined in *RFC2396*.

In WML, URLs are used when specifying navigation (hyperlinking, for example) or external resources (an image or a script, for example).

Fragment anchors

WML has adopted the HTML way 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, the URL names an entire deck, and the deck URL also identifies the first card in a deck.

Example [14.]

The following go element includes an URL referring to another card in the same deck. In this case the URL includes the fragment identifier (#):

```
<go href="#Next_Card"/>
```

For another fragment anchor example, see the simple WML example in “Card and Deck” on page 17.

Relative URLs

WML has adopted the use of relative URLs, as specified in *RFC2396*. The base URL of a WML deck is the URL that identifies the deck.

Example [15.]

The following simple example demonstrates the use of a relative URL. When the user activates the `go` task, the user agent navigates to the `options` directory in the same domain where the current deck is located.

```
<wml>
   <card>
      <do type="options" label="Options">
         <go> href="/options/foo.wml"/>
         label="menu"
      </do>
      <!-- rest of the card -->
   </card>
</wml>
```

Browser context

The WML state is stored in the “browser context”. The browser context is used to manage all parameters and user agent states, including variables, the navigation history and other implementation-dependent information related to the current state of the user agent.

History

WML includes a simple navigational history model that allows you to efficiently manage navigating backwards. The user agent history is implemented as a stack of URLs that represent the navigational path the user traveled to arrive at the current card. You may perform three operations on the history stack, described in the table below.

Operation	Explanation
Reset	The history stack may be reset to a state where it contains only the current card. For more information, see the discussion on the `newcontext` attribute on page 26.
Push	A new URL is pushed onto the history stack when you navigate to a new card.
Pop	The current card's URL (top of stack) is popped when you navigate backwards.

When you enter a card, the card URL is added to the history stack. This allows you to navigate back to the previous card in the history by executing a `prev` task. The execution of `prev` pops the current card URL from the history stack.

WML elements

This chapter provides reference information on WML elements and attributes. The usage of each element is demonstrated with an example.

Decks and cards

WML data is 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 the information requested, makes choices and navigates to another card or returns to a previously visited card.

The following sections describe the WML “building blocks”: document headers and deck and card components.

Common attributes

All WML elements have two core attributes, `id` and `class`, that can be used for such tasks as server-side transformations. The `id` attribute provides an element an unique name within a single deck. The `class` attribute affiliates an element with one or more classes.

Multiple elements can be given the same class name. All elements of a single deck with a common class name are considered to be part of the same class. Class names are case sensitive. An element can be part of multiple classes if it has multiple unique class names listed in its `class` attribute. Multiple class names within a single attribute must be separated by white space. Redundant class names as well as insignificant white space between class names may be removed The WML agent should ignore this attribute.

All elements containing text may contain the xml:lang attribute. The xml:lang attribute specifies the natural language of an element or its attributes. The attribute identifies to the user agent the language used for text, such as an element's content and attribute values, that may be presented to the user.

Document header

A valid WML deck is a valid XML document and therefore must contain an XML declaration and a document type declaration. A typical document header contains:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
```

You can also save the document type declaration in a file, thus allowing for a faster bytecode translation.

Note the following document identifiers:

The SGML public identifier is `"-//WAPFORUM//DTD WML 1.1//EN"`.

The WML media type identifier is:

— In textual form: `"text/vnd.wap.wml"`

— In tokenized form: `"application/vnd.wap.wmlc"`

! **Note**: These types are not yet registered with the IANA and are consequently ***experimental*** media types.

wml element

Description

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

Contained elements

```
head ?
template ?
card +
```

Syntax

The `xml:lang` attribute of the `wml` element is explained in the following table.

Attribute	Explanation
`xml:lang=`*`nmtoken`*	This attribute specifies the natural or formal language in which the document is written. For information, see "Common attributes" on page 23.

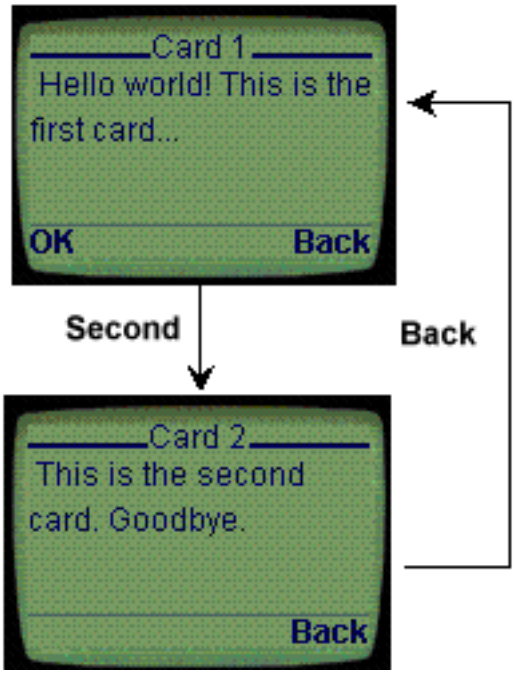
Example [16.]

The following is a deck containing two cards, each represented by a `card`. After loading the deck, the user agent displays the first card. If the user activates the `do` element, the user agent displays the second card.

The `xml:lang` attribute specifies that the document is written in US English.

```
<wml xml:lang="en-us">
   <card id="card1" title="Card 1">
   <do type="accept">
         <go href="#card2"/>
      </do>
      <p>
         Hello world!
         This is the first card...
      </p>
   </card>

   <card id="card2" title="Card 2">
      <p>
         This is the second card.
         Goodbye.
      </p>
   </card>
</wml>
```

The deck generates the following user interface in the user agent:



A WML deck containing two cards.

card element

Description

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

The `card` element is a container for text and input elements that is flexible enough 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, while giving considerable freedom for implementing layout and user input schemes in the user agent. For example, a `card` can be presented as a single page on a large-screen device or 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. Note that the order of elements in the card is significant. You may use a card's id as a fragment anchor. See "Fragment anchors" on page 20 for more information.

Contained elements

```
onevent *
timer ?
do *
p *
```

! **Note**: If a `card` element contains `onevent` elements, the `onevent` elements must be first. If a `card` contains a `timer` element, the `timer` must follow any `onevent` elements and precede any `do` or `p` elements.

Syntax

The attributes of the `card` element are explained in the following table.

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies advisory information on the card.
`newcontext=`*`boolean`*	If you set this attribute to true, the current browser context is re-initialized upon entry to this card and performs the following operations: Clears the navigational history state. Resets the implementation-specific state to a well-known value.

Attribute	Explanation
	`newcontext` is only performed as part of the `go` task. For an example of the `newcontext` attribute, see Example [46.] on page 85. The default value is false.
`ordered=boolean`	This attribute gives an indication to the user agent about how the `card` content is organized. This indication can be used to organize the content presentation or to otherwise influence the layout of the card. `ordered="true"` — The card is naturally organized as a linear sequence of field elements, for example, 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. For example, sending an email message requires a To: address, a subject and a message, and they are logically specified in this order. `ordered="false"` — 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 updates individual input fields. For 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 must choose the right boundaries between fields. User agents use the following heuristic for finding the screen flip location: `fieldset` — Defines a logical boundary between fields. Fields (e.g. `input`) may be individually displayed. When this is done, the line of markup (`flow`) immediately preceding the field is treated as a

Attribute	Explanation
	field prompt and displayed with the input element. The `table` must be treated differently than `input` and `select`. The user agent must insert a line break before each `table` element, except when it is the first non-whitespace markup in a card. The user agent must insert a line break after each `table` element, excpt when it is the final element in a card.
`onenterforward=`*`href`*	The `onenterforward` event occurs when the user navigates into a card using a `go` task. For more information, see “onenterforward event” on page 39.
`onenterbackward=`*`href`*	The `onenterbackward` event occurs when the user navigates into a card using a `prev` task. For more information, see “onenterbackward event” on page 40.
`ontimer=`*`href`*	The `ontimer` event occurs when a `timer` expires. For more information, see “ontimer event” on page 38.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [17.]

The following is an example of a simple `card` embedded in a WML deck. The card contains text, which is displayed by the user agent.

When the user navigates to this card, the browser context is re-initialized, meaning that all variables are cleared and the history stack is emptied.

```
<wml>
   <template>
      <do type="accept" name="exit" label="EXIT">
         <prev/>
      </do>
   </template>
   <card id="card_1" title="Welcome" newcontext="true">
      <p>
      Hello World!
      </p>
   </card>
</wml>
```

The deck generates the following user interface in the user agent:

A single card.

template element

Description

The `template` element declares a template for cards in the deck. Event bindings specified in the `template` (for example, `do` or `onevent`) apply to all cards in the deck. Specifying an event binding in the `template` is equivalent to specifying it in every card. A card element may override the behavior specified in the `template`. In particular, note the following override rules:

`do` elements specified in the `template` may be overridden in individual cards if both elements have the same `name` attribute value.

Intrinsic event bindings specified in the `template` may be overridden by event bindings in a card.

Contained elements

```
do *
onevent *
```

Syntax

The attributes of the `template` element are explained in the following table.

Attribute	Explanation
`onenterforward=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URL when the user enters this card. For more information, see “onenterforward event” on page 39.
`onenterbackward=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URL when the user navigates backwards to this card, for example, using a `prev` task. For more information, see “onenterbackward event” on page 40.

Attribute	Explanation
`ontimer=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URL after the timer has expired. For more information, see “ontimer event” on page 38.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [18.]

The following `template` element includes a `do` element indicating navigation to the previous card in the deck.

```
<template>
   <do type="prev" label="Previous">
      <prev/>
   </do>
</template>
```

head element

Description

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

You must include one of the following elements at least once inside a `head` element:

`access`

`meta`

Contained elements

`(access|meta) +`

Example [19.]

See the `access` and `meta` element examples:

`access`: Example [20.] on page 32.

`meta`: Example [21.] on page 34.

access element

Description

The `access` element specifies access control information for the entire deck. Note that it is a WML syntax error for a deck to contain more than one `access` element. If a deck does not include an `access` element, access control is disabled. When access control s disabled, cards in any deck can access this deck.

The default access control settings let you access any URIs in the same domain. The user agent uses a prefix match to compare the URIs of decks trying to access your deck with the attribute values you define.

The following table lists the elements that let you navigate between decks and the associated access settings the target deck must specify.

Element	Access requirements
`prev`	None.
`go href=`*`href`*	The deck located at the specified URI must specify `domain` and/or `path` attributes that match the URI of the requesting deck.

Contained elements

None.

Syntax

The attributes of the `access` element are explained in the following table.

Attribute	Explanation
`domain=`*`cdata`* `path=`*`cdata`*	A deck's `domain` and `path` attributes specify the other decks that can 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.

Attribute	Explanation
	`Domain` suffix matching is done using the entire element of each sub-domain and must match each element exactly. For example, www.acmecorp.com matches `acmecorp.com`, but does not match `corp.com`. `Path` prefix matching is done using entire path elements and must match each element exactly. For example, `/X/Y` matches path="`/X" attribute`, but does not match path="`/XZ" attribute`. 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. `Domain` and `path` follow URL capitalization rules.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [20.]

For example, given the following access control attributes:

```
domain="acmecorp.com"
path="/pub"
```

The following referring URLs would be allowed to access the deck:

```
acmecorp.com/pub/stocks.cgi
www.acmecorp.com/pub/demos/packages.cgi
```

The following referring URLs would not be allowed to access the deck:

```
www.test.net/pub
www.acmecorp.com/internal/foo.wml
```

The following `head` element includes an access control element indicating that only decks in the `WML` directory of the domain `mycompany.com` can access this deck.

```
<head>
   <!-- NOTE: The DOMAIN and PATH must be customized for
        your network location of the WML decks -->

   <access domain="mycompany.com" path="/WML" >
</head>
```

meta element

Description

The `meta` element contains generic meta information relating to the WML deck. Meta information is specified with property names and values.

Note that it is a WML syntax error for a `meta` element to contain more than one attribute specifying a property name, that is, more than one attribute from the following set: `name` and `http-equiv`.

Contained elements

None.

Syntax

The attributes of the `meta` element are explained in the following table.

Attribute	Explanation
`content=`*`cdata`*	This attribute specifies the property value. It is required.
`name=`*`cdata`*	This attribute specifies the property name. The user agent ignores named meta-data. Network servers do not emit WML content containing meta-data named with this attribute.
`http-equiv=`*`cdata`*	This attribute may be used in place of `name`; it indicates that the property should be interpreted as an HTTP header. Meta-data named with this attribute is converted to a WSP or HTTP response header if the content is tokenized before it arrives at the user agent.

Attribute	Explanation
`forua=`*`boolean`*	This attribute specifies that the author intended the property to reach the user agent. If the value is `false`, an interrmediate agement must remove the `meta` element before the document is sent to the client. If the value is `true`, the meta data of the element must be delivered to the user-agent. The method of delivery may vary. For example, `http-equiv` meta data may be delivered using HTTP or WSP headers.
`scheme=`*`cdata`*	This attribute specifies a form or structure that may be used to interpret the property value. Note that scheme values vary depending on the type of meta-data.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [21.]

The following `head` element includes an access control element, described above in the previous example, and a `meta` element that provides information to the user agent on the character set used in the WML deck.

```
<head>
   <!-- NOTE: The domain and path must be customized for
        your network location of the WML decks -->

   <access domain="mycompany.com" path="/WML" >
   <meta content="charset" user agent="character-set=UTF-8"/>
</head>
```

Events

WML includes several elements that allow you to handle navigation and events by specifying the processing of user agent events. For example, you may associate events with tasks so that when an event occurs, the associated task is executed. You may specify a variety of tasks, such as navigation to an URL. Event bindings can be implemented by several elements, including `do` and `onevent`.

do element

Description

The do element gives the user a general mechanism for performing actions on the current card, that is, a card-level user interface element. The do element is mapped to an unique user interface *widget* that the user can activate. For example, the widget mapping may be to a graphically rendered button, a soft key 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 an indication to the user agent of how the WML author intended the element to be used, and is taken by the user agent to provide a suitable mapping onto a physical user interface construction. Note that WML authors must not rely on the semantics or behavior of an individual type value, or on the mapping of type to a particular physical construction.

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

Card-level: The do element may appear inside a card and may be located anywhere in the text flow. If the user agent intends to render the do element inline (that is, in the text flow), it should use the element's anchor point as the rendering point. Note that WML authors must not assume that the inline rendering of the do element is correct; nor must they assume that the inline rendering of the element is positioned correctly.

Deck-level: The do element may appear inside a template, indicating a deck-level do element. A deck-level do element applies to all cards in the deck, that is, it is equivalent to specifying the do within each card. For the purposes of inline rendering, the user agent behaves as if deck-level do elements were located at the end of the card's text flow.

In particular, you should pay attention to the following:

A card-level do element overrides a deck-level do element if they have the same name. 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.

The inactive do elements and the active do elements with a noop task element are not presented to the user.

The user can access all the do elements with a task other than noop. The user can activate these user interface items when viewing the card containing the active do elements. When the user activates a do element, the associated task is executed.

Contained elements

go|prev|noop|refresh

Syntax

The attributes of the `do` element are explained in the following table.

Attribute	Explanation
**`type=`*`cdata`*	The `do` element type. This attribute provides an indication to the user agent about how the WML author intended the element to be used, and how it should be mapped to a physical user interface construction. All types are reserved, except for those marked as experimental. This attribute is required. User agents accept any `type`, but may treat any unrecognized type as the equivalent of `unknown`. In the following, the `*` character represents any string. For example, `Test *` indicates any string starting with the word `Test`. `accept` — Positive acknowledgement (acceptance). `prev` — Navigates backwards through history. `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, e.g., type="". `X-*, x-*` — Experimental types. This set is not reserved.

Attribute	Explanation	
	`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.
`label=`*`vdata`*	This attribute specifies a textual string for labeling the user interface widget. If an element cannot be dynamically labeled, this attribute is ignored. To work well, labels should have no more than six characters.	
`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`) and at the deck-level (in a `template`) and both elements have the same `name`, the deck-level `do` element is ignored. It is a WML syntax error to specify two or more `do` elements with the same `name` in a single card or in the `template`. 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 you set this attribute to `true`, the user agent may ignore this element. The default is `false`.	
`xml:lang`	See "Common attributes" on page 23.	
`id`	See "Common attributes" on page 23.	
`class`	See "Common attributes" on page 23.	

Example [22.]

This example demonstrates a `DO` element that includes a `go` task. When the user activates the element by selecting **Next**, the user agent goes to `card2` in the current deck and displays it to the user.

```
<card id="card1">
   <do type="accept" label="Next">
      <go href="#card2"/>
   </do>
   <p>
      Select <b> Next </b> to go to the next card.
   </p>
</card>

<card id="card2">
   <p>
      This is card 2.
   </p>
</card>
```

For a more comprehensive example of using the `do` element, see “Override example [29.]” on page 45.

ontimer event

Description

The `ontimer` event can be specified inside the following elements:

`card`

`template`

The event occurs when a timer expires.

Syntax

The attributes of the `ontimer` event are explained in the following table.

Attribute	Explanation
`ontimer=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URI after the timer has expired.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [23.]

The following is a simple example of the `ontimer` element:

```
<card id="cardname" ontimer="http://wapserver/hello.wml" title="title">
   <timer value="50"/>
   <p>
      Hello World!
   </p>
</card>
```

onenterforward event

Description

The `onenterforward` event occurs when the user enters a card using a `go` task or any method with identical semantics.

The `onenterforward` intrinsic event may be specified inside the following elements:

`card`

`template`

Event bindings specified in the `template` apply to all cards in the deck and may be overridden as specified in “Card and deck task override” on page 44.

Syntax

The attributes of the `onenterforward` event are explained in the following table.

Attribute	Explanation
`onenterforward=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URI when the user enters this card.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [24.]

You can specify that certain tasks are to be executed when an intrinsic event occurs in two ways.

First, you can specify an URI to be navigated to when the event occurs. This event binding is specified in a well-defined element-specific attribute which is the equivalent of a `go` task. For example:

```
<card onenterforward="http://wapserver/hello.wml"> Hello World! You came
back </card>
```

Note that the message "Hello World! You came back" is not displayed when you navigate forward to this card. Similarly, when you navigate to this card backwards, the card is displayed.

Second, you can use an expanded version of the method above, which gives you more control over user agent behavior. An `onevent` element is implemented 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 href=" http://wapserver/hello.wml "/>
   </onevent>
   <p>
      Hello World! You came back.
   </p>
</card>
```

However, the user agent treats the attribute syntax as an abbreviated form of the `onevent` element where the attribute name is mapped to the `onevent` type.

onenterbackward event

The `onenterbackward` event occurs when the user navigates into a card using a `prev` task or any method with identical semantics. In other words, the `onenterbackward` event occurs when the user navigates into a card by using an URL retrieved from the history stack.

The `onenterbackward` intrinsic event may be specified inside the following elements:

`card`

`template`

Event bindings specified in the `template` apply to all cards in the deck and may be overridden as specified in "Card and deck task override" on page 44.

Syntax

The attributes of the `onenterbackward` event are explained in the following table.

Attribute	Explanation
`onenterbackward=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URI when the user navigates backwards to this card, for example, using a `prev` task.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [25.]

In the following example, the `onenterbackward` event causes the user agent to navigate to `card2` when the user enters this card using a `prev` task or navigating backwards in the history stack. This means that `card2` is displayed to the user instead of `card1`. Note that if the user navigates forward to this card by using the `go` task, for example, `card1` is displayed.

```
<card id="card1">
   <onevent type="onenterbackward">
      <go href="#card2"/>
   </onevent>
   <p>
      Hello World!
   </p>
</card>

<card id="card2">
   <p>
      You came back!
   </p>
</card>
```

Similar to the previous example, `card1` could also be presented as follows:

```
<card id="card1" onenterforward="#card2"> Hello World! </card>
```

onpick event

Description

The `onpick` event occurs when the user selects or deselects the item in which the event is specified.

The `onpick` intrinsic event may be specified inside the `option` element.

Syntax

The attributes of the `onpick` event are explained in the following table.

Attribute	Explanation
`onpick=`*`href`*	Specifies an intrinsic event that instructs the user agent to go to the specified URI when the user selects the option (or deselects it if the `select` element allows multiple choices) in which the event is specified.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [26.]

In the following example, when the user selects an item from the option list, the user agent navigates to the appropriate deck in the same domain. For example, if the user selects item `1`, information on cats, the user agent navigates to the deck `cat.wml` containing information on cats.

```
<card id="Card_1">
   <p>
      Select your favorite animal:
      <select name="animal">
         <option value="1" onpick="cat.wml"> Cat     </option>
         <option value="2" onpick="dog.wml"> Dog     </option>
         <option value="3" onpick="horse.wml"> Horse </option>
      </select>
   </p>
</card>
```

onevent element

Description

The `onevent` element binds a task to a particular intrinsic event for the immediately enclosing element; that is, specifying an `onevent` element inside an element associates an intrinsic event binding with that element.

The user agent ignores any `onevent` element specifying a `type` that does not correspond to a legal intrinsic event for the immediately enclosing element.

Contained elements

`go|prev|noop|refresh`

Syntax

The attributes of the `onevent` element are explained in the following table.

Attribute	Explanation
type=*CDATA*	The type attribute indicates the name of the intrinsic event. This attribute is required.
id	See "Common attributes" on page 23.
class	See "Common attributes" on page 23.

Example [27.]

The following card indicates that when a user navigates backwards to it, the onenterbackward event is activated and the card's message is not displayed to the user. Instead, the user agent navigates to the deck foo.wml located in the domain www.acme.com.

Note that if the user navigates forward to this card, its content is displayed.

```
<card>
   <onenvent type="onenterbackward">
      <go href="http://www.acme.com/foo.wml"/>
   </onevent>
   <p>
      Welcome to a new age!
   </p>
</card>
```

postfield element

Description

The postfield element specifies a field name and value for transmission to an origin server during a URL request. The actual encoding of the name and value will depend on the method used to communicate with the origin server.

Contained elements

None.

Syntax

The attributes of the postfield element are explained in the following table.

Attribute	Explanation
name=*vdata*	The name attribute specifies the field name. This attribute is required.

Attribute	Explanation
`value=`*`vdata`*	The `value` attribute specifies the field value. This attribute is required.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [28.]

The following example posts three name values to the web server, letting you send data back from the client. HTTP POST method is used to send the data.

```
<go method="post" href="http://hostname/servlet/bank">
   <postfield name="money" value="100"/>
   <postfield name="account" value="12345"/>
   <postfield name="operation" value="deposit"/>
</go>
```

Card and deck intrinsic events

You may specify the `onenterforward` and `onenterbackward` intrinsic events at both the card and deck level using the override semantics defined in “Card and deck task override” on page 44. 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 `onenterforward` element and vice versa.

Card and deck task override

You can use a variety of elements 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` and specify event-processing behavior for that particular card.

Deck-level: The event-handling element may appear inside a `template` and specify event-processing behavior for all cards in the deck. A deck-level event-handling element is equivalent to specifying the element in each card of the deck.

In particular, you should notice the following override rules:

A card-level event-handling element overrides a deck-level event-handling element if they both specify the same event.

A card-level `onevent` element overrides a deck-level `onevent` element if they both have the same `type`.

A card-level `do` element overrides a deck-level `do` element if they have the same name.

If a card-level element overrides 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-level and deck-level elements will be ignored and no side effects will occur on delivery of the event. In this case, the user agent will not show the element to the user, for example, render a UI control. In effect, the `noop` removes the element from the card.

Override example [29.]

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` and will displays the `do` to the user.

The second card overrides the deck-level `do` with a `noop`. The user agent does not display the `do` element when displaying the second card.

The third card overrides the deck-level `do`, 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 id="first">
      <!-- deck-level do not overridden. The card
           exposes the deck-level do as part of the current card. -->

      <!-- rest of the card -->
      ...
   </card>

   <card id="second">
      <!-- deck-level do is overridden with noop.
           It is not exposed to the user. -->
      <do type="options" name="do1">
         <noop/>
      </do>

      <!-- rest of the card -->
      ...
   </card>

   <card id="third">
      <!-- deck-level do is overridden.
           It is replaced by a card-level do -->
      <do type="options" name="do1" label="options">
```

```
        <go href="/options"/>
      </do>

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

Tasks

WML allows you to specify tasks that can be performed when a certain event occurs, such as navigating to a specified card or deck.

WML includes four task elements that are described in more detail in the following sections:

- `go`
- `prev`
- `noop`
- `refresh`

Tasks are bound to events in the following events:

- `do`
- `onevent`

An `anchor` element may contain a `go`, `prev`, or `refresh` task.

go task

Description

The `go` element declares a `go` task, indicating navigation to an URL. If the URL names a WML card or deck, it is displayed. A `go` executes a push operation on the history stack.

Contained elements

`setvar *`

`postfield *`

Syntax

The attributes of the `go` element are explained in the following table.

Attribute	Explanation
`href=`*`href`*	This attribute specifies the destination URI, for example, the URI of the card to display. This attribute is required.
`sendreferer=`*`boolean`*	If you set this attribute to true, the user agent includes the URL of the deck containing this task in the URI request. This allows a server to perform a form of access control on URIs, based on the decks that link to them. Specifying `sendreferer=true` causes the user agent to set the HTTP "Referer" header to the smallest relative URI of the requesting deck. Note that if you want to restrict access to your services, the decks that request URIs of your services must set this option to true. The default value is false.
`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.. The default value is get.
`accept-charset=`*`CDATA`*	This attribute specifies the list of character encodings for data that the web server must accept when processing input. The value of this attribute taken from a list of character encoding names (`charset`), separated by commas or spaces 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, that is, the server must accept all of the acceptable character encodings. The default value for this attribute is the reserved string `unknown`. The user agent interprets this value as the character encoding that was used to transmit the WML deck containing this attribute. For a list of character sets supported by the WAP Toolkit, see "WML character set" on page 9.
`id`	See "Common attributes" on page 23.

Attribute	Explanation
`class`	See "Common attributes" on page 23.

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

1. The field name/value pairs are identified and all variables are substituted.
2. The user agent should transcode the field names and values to the correct character set, as specified explicitly by the `accept-charset` or implicitly by the document encoding.
3. The field names and values are escaped using URL-escaping and assembled into an `application/x-www-form-urlencoded` content type. URL-escaping is defined in RFC2396 and the assembly of the content is specified in RFC2070.
4. The request is preformed according to the `method` attribute's value:

   `get` – if the `method` attribute has a value of `get` and the `href` attribute value is an HTTP URI, the submission data is added to the query component of the URI. An HTTP GET operation is performed on the resulting URL.

   `post` – if the `method` attribute has a value of `post` and the `href` attribute value is an HTTP URI, the submission data is sent to the origin server with an HTTP POST operation. The submission is identified with a content type of application/x-www-form-urlencoded, with a `charset` parameter indicating the character encoding.

Example [30.]

In the following example, the `go` element would cause an HTTP GET request to the URL `"/foo?x=1"`:

```
<go href="/foo">
   <postfield name="x" value="1"/>
</go>
```

The next example causes an HTTP POST to the URL "/bar" with a message entity containing `"w=12&y=test"`:

```
<go href="/bar" method="post">
   <postfield name="w" value="12"/>
   <postfield name="y" value="test"/>
</go>
```

The following example creates a user interface widget with the label **Help** and goes to the card named `help` when activated.

```
<card id="card1">
   <do type="help" label="Help">
      <go href="#help"/>
   </do>
</card>

<card id="help">
   <p>
   Help topics:
   ...
   </p>
</card>
```

prev task

Description

The `prev` element declares a `prev` task, indicating navigation to the previous URI in the history stack. A `prev` performs a pop operation on the history stack and removes the current URI from the history stack. If there is no previous URI in the history stack, the `prev` element has no effect.

Contained elements

`setvar *`

Syntax

The attributes of the `prev` element are explained in the following table.

Attribute	Explanation
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [31.]

The following example creates a user interface widget with the label **Back** and goes to the previous card when activated.

```
<do type="accept" label="Back">
   <prev/>
</do>
   <p>
      Hello, World!
   </p>
```

refresh task

Description

The `refresh` element declares a `refresh` task, indicating an update of the specified card variables. Side effects of the state changes that are visible to the user (for example, a change in the screen display) occur during the processing of the `refresh` task. A refresh and its side effects must occur even if the elements have no `setvar` elements given that context may change by other means (e.g., `timer`).

Contained elements

`setvar *`

Syntax

The attributes of the `refresh` element are explained in the following table.

Attribute	Explanation
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [32.]

The following WML will set a value `foo` for the variable `product` and refresh the display to update variable values.

```
<refresh>
   <setvar name="product" value="foo"/>
</refresh>
```

noop task

Description

The `noop` element specifies that nothing should be done, that is, “no operation”. This element is useful for overriding deck-level `do` elements.

The attributes of the `noop` element are explained in the following table.

Attribute	Explanation
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Variables

Parameters can be set for all WML content, giving you a great deal of flexibility in creating cards and decks that dynamically change display content and navigation based on user input. WML variables can be used instead of strings and are substituted at runtime with their current value.

A variable is set if it has a value that does not equal 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.

For a comprehensive variable example, see Example [46.] on page 85.

setvar element

Description

The `setvar` element specifies the variable to set in the current browser context as a side effect of executing a task. The element is ignored if the `name` attribute does not evaluate to a legal variable name at runtime.

Contained elements

None.

Syntax

The variable attributes are explained in the following table.

Attribute	Explanation
`name=`*`vdata`*	Specifies the variable name. This attribute is required.
`value=`*`vdata`*	Specifies the value to be assigned to the variable. This attribute is required.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [33.]

The following example sets a variable named `product` with the default value `WapSdk` and provides a soft key labeled **Clear**. When the user selects **Clear**, the product variable value is cleared and the current deck display is refreshed with the empty value.

```
<card id="Product" title="Choose Product">
   <p>
      Product name:
         <input title="Product name" name="product" value="WapSdk"/>
            <do type=”accept” label="Clear">
               <refresh>
                  <setvar name="product" value=""/>
               </refresh>
            </do>
   </p>
</card>
```

Naming variables

WML variable names consist of an US-ASCII letter or underscore followed by a zero or more letters, digits or underscores. Any other characters are illegal and result in an error. Note also that variable names are case sensitive. Parentheses are required anywhere the end of a variable cannot be inferred from the surrounding context, for example, if the variable ends with an illegal character such as a white space. The following examples demonstrate legal variables:

```
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
```

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 when specified as `&#x24;`, results in a variable substitution.

In order to include a $ character in a WML deck, it must be explicitly escaped by using the following syntax:

```
$$
```

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

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

would be displayed as:

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

To include the $ character in URL-escaped strings, specify it with the URL-escaped form `%24`.

! **Note**: Variable names are case sensitive. This means that `variable1`, `Variable1` and `varIABle1` are all different variables.

Validating variables

Within the WML document, any string following a single dollar sign ('$') must be treated as a variable reference and validated. Each reference must use proper variable name syntax. Each reference must be placed either within a card's text (`#PCDATA`) or within `%vdata` or `%href` attribute values. The deck is in error if any variable reference uses invalid syntax or is placed in an invalid location.

Example [34.]

The following examples show invalid variable use:

```
<!-- bad variable syntx -->
Balance left is $10.00

<!--bad placement (in the type attribute) -->
<do type="x-$(type)" label="$type">
```

Restricting variable context

User agents may provide users means to reference and navigate to resources independent of the current content. For example, user agents may provide bookmarks or a URL input dialog. Whenever a user agent navigates to a resource that was not the result of an interaction with the content in the current context, the user agent must establish another context for that navigation. The user agent may terminate the current context before establishing another one for the new navigation attempt.

Setting variables

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

The `setvar` element allows you to set the variable state as a side effect of navigation. `Setvar` may be specified in the following task elements:

`go`

`prev`

`refresh`

Variables can also be set in the following situations:

Input elements set the variable identified by the `name` attribute to any information entered by the user.

— The `input` element assigns the entered text to the variable.

— The `select` element assigns the value present in the `value` attribute of the chosen `option` element.

Note that the user input is written to variables when the user commits the input to the `input` or `select` element.

The `setvar` element specifies a variable name and value, for example:

```
<setvar name="location" value="$(X)"/>
```

The variable name specified in the `name` attribute (for example, `location`) is set as a side effect of navigation. For more information on the processing of the `setvar` element, see "History" on page 21.

Note the following when setting variables:

You can change variable values set with WML using WMLScript and vice versa. This indicates that WML and WMLScript use the same variables.

You can set and edit variables in the WAP Toolkit **Variables** view.

You can use the `card` element's `newcontext` attribute to clear all variable values of the current browser context.

If a `go` element's `href` attribute has variable references, the variable references are only resolved against the variables in the Browser Context. These variable references are not resolved against any `setvar` elements within the `go` element.

If a `go` element contains `setvar` elements and the name or value of the `setvar` elements contain variable references, the variable references are resolved against the variables in the Browser Context. These variable references are not resolved against any `setvar` elements within the `go` element.

Substituting variables

You can substitute variable values into formatted text (*`PCDATA`*), option values (*`vdata`*) and *`href`* attributes in WML elements. However, note that only textual information can be substituted, that is, no substitution of elements or attributes is possible. The substitution of variable values happens at runtime in the user agent. As the substitution is merely a string substitution operation, it does not affect the current value of the variable. If an undefined variable is referenced, it results in the substitution of an empty string.

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 summarizes the current conversions and their legal abbreviations.

Variable reference	Explanation
`$setvar` or `$(setvar)`	Substitutes the value of `setvar`. The user agent escapes the variable using URL escaping conventions in the appropriate context.
`$(setvar:e)` or `$(setvar:escape)`	Substitutes the value of `setvar`, escaping non-alphanumeric characters according to URL encoding conventions.

Variable reference	Explanation
`$(setvar:unesc)`	Substitutes the value of `setvar`, unescaping non-alphanumeric characters according to URL encoding conventions.
`$(setvar:N)` or `$(setvar:noesc)`	Substitutes the value of `setvar`, without escaping non-alphanumeric characters.

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

URL escaping is detailed in *RFC2396*. All lexically sensitive characters defined in WML must be escaped, including all `reserved` and `unsafe` URL characters, as specified by *RFC2396*.

If no conversion is specified, the variable is substituted using the conversion format appropriate for the context. The `onenterbackward`, `onenterforward`, `href` and `src` attributes default to escape conversion; elsewhere, no conversion is done. Specifying the `noesc` conversion disables context-sensitive escaping of a variable.

The following example illustrates posting name value pairs to a web server so that data can be sent from the client:

```
<go method="post" href="http://hostname/servlet/dealer">
   <postfield name="make" value="ford"/>
   <postfield name="car" value="escort"/>
</go>
```

For more detailed information on the variable substitution syntax, refer to the *WML Specification*.

Parsing the variable substitution syntax

The variable substitution syntax (for example, `$X`) is parsed after all XML parsing is complete. This implies that all variable syntax is parsed after the XML constructs, such as elements and entities, have been parsed. In the context of variable parsing, all XML syntax takes precedence over the variable syntax, for example, 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;
```

User input

The following sections discuss how to handle user input in WML.

input element

Description

The `input` element specifies a text entry object. You can specify the format of the user input with the optional `format` attribute. If a valid input mask is bound to an input object, the user agent must ensure that any value collected by the entry object conforms to the bound input mask. If the input collected does not conform to the input mask, the user agent must not commit that input and must notify the user that the input was rejected an d allow the user to resubmit new input. The user agent must not initialize the input object with any value that does not conform to the bound input mask. In the event that initializing data does not conform to the input mask, the user agent must behave as if there was no initialization data.

Contained elements

None.

Syntax

The attributes of the `input` element are explained in the following table.

Attribute	Explanation
`name=`*`nmtoken`*	The `name` attribute specifies the name of the variable to set with the result of the user's text input. The `name` variable's value is used to pre-load the text entry object. The `name` attribute is required.
`value=`*`vdata`*	The `value` attribute indicates the default value of the variable named in the `name` attribute. When the element is displayed and the variable named in the `name` attribute is not set, the `name` variable is assigned the value specified in the `value` attribute. If the `name` variable already contains a value, the `value` attribute is ignored. If the `value` attribute specifies a value that does not conform to the input mask specified by the `format` attribute, the user agent ignores the `value` attribute. In the case where no valid value can be established, the `name` variable is left unset.
`type=(text|password)`	This attribute specifies the type of the text-input area. The default type is `text`. The following values are allowed:

Attribute	Explanation	
	`text`	A text entry control. The input is echoed in a manner appropriate to the user agent and the input mask. If the submitted value conforms to an existing input mask, the user agent must store that input unaltered and in its entirety in the variable named in the `name` attribute. For example, the user agent must not trim the input by removing leading or trailing white space from the input. If the variable named by the `name` attribute is unset, the user agent should echo an empty string in an appropriate manner.
	`password`	A text entry control. Input of each character is echoed in an obscured form, in a manner appropriate to the user agent. For example, visual 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 insecure and critical applications should not be dependent on it. Similar to a `text` type, if the submitted value conforms to an existing input mask, the user agent must store input unaltered and in its entirety in the variable named in the name attribute. User agents should not obscure non-formatting characters of the input mask. If the variable named by the name attribute is unset, the user agent should echo an empty string in an appropriate manner.

Attribute	Explanation
`format=`*`cdata`*	This 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. An input mask is only valid when it contains legal format codes only. User agents must ignore invalid masks. The format control characters specify the data format that the user is expected to enter. The default format is "`*M`". The format codes are explained in the following. `A` Allows any uppercase alphabetic or punctuation character, that is, uppercase non-numeric character. `a` Allows any lowercase alphabetic or punctuation character, that is, lowercase non-numeric character. `N` Allows any numeric character. `X` Allows any uppercase character. `x` Allows any lowercase character. `M` Allows any character. The user agent may choose to assume that the character is uppercase for the purposes of simple data entry, but must allow entry of any character. `m` Allows any character. The user agent may choose to assume that the character is lowercase for the purposes of simple data entry, but must allow entry of any character. `*f` Allows any number of characters; f is one of the above format codes and specifies what kind of characters can be entered. Note that this format can only be specified once and must appear at the end of the format string.

Attribute	Explanation	
	nf	Allows *n* characters where *n* is a number from 1 to 9; *f* is one of the above format codes (except *f) and specifies what kind of characters can be entered. Note that this format can only be specified once and must appear at the end of the format string.
	\c	Displays the next character, c, in the entry field. Allows escaping of the format codes as well as introducing non-formatting characters so they can be displayed in the entry area. Escaped characters are considered part of the input's value, and must be preserved by the user agent. For example, the stored value of the input "12345-123" having a mask "NNNNN\ -3N" is "12345-123" and not "12345123". Similarly, if the value of the variable named by the name attribute is "12345123 and the mask is "NNNNN\ –3N" the user agent must unset the variable since it does not conform to the mask.
emptyok=*boolean*	If you set this attribute to true, the input element accepts empty input although a format string that is not empty 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, that is, emptyok=false.	
size=*number*	This attribute specifies the width, in characters, of the text input area.	
maxlength=*number*	This attribute specifies the maximum number of characters that the user can enter in the text entry area. The default value for this attribute is an unlimited number of characters.	

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a title for the `input` element. The title may be used in the presentation of this object.
`tabindex=`*`number`*	The `tabindex` element 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.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [35.]

The first example specifies an `input` element that accepts any characters and displays the input to the user in a form the user can read. The maximum number of characters that can be entered is 32, and the resulting input is assigned to the variable named `X`.

```
<input name="X" type="text" maxlength="32"/>
```

The next 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 name="name" type="text" value="Robert"/>
```

The following example is a card that prompts the user for a first name, last name and age. Note that in the Age field the user can enter a two-digit number.

```
<card>
<p>
   First name: <input type="text" name="first"/><br/>
   Last name: <input type="text" name="last"/><br/>
   Age: <input type="text" name="age" format="NN"/>
</p>
</card>
```

select element

Description

Select list is an input element that allows the user to choose from a list of options. WML supports both single-choice and multiple-choice lists.

The `select` element lets users pick from a list of options. Each option is specified by an `option` element having one line of formatted text. You can organize `option` elements into hierarchical groups using the `optgroup` element.

You must include one of the following elements at least once inside a `select` element:

`optgroup`

`option`

On entry into a card containing a `select` element, the initial options are selected as follows:

If the `iname` attribute exists, the indices in the variable named by the `iname` 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 `iname` attribute does not exist and the `name` attribute exists, the value of the variable specified by `name` is used to select the options. If the variable specified by `name` 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 `name` is updated to the value of the option.

Both `name` and `iname`, or `value` and `ivalue` may be specified. `ivalue` takes precedence over `value`, and `iname` takes precedence over `name`.

Contained elements

```
optgroup +
option +
```

Syntax

The attributes of the `select` element are explained in the following table.

Attribute	Explanation
`multiple=`*`boolean`*	If you set this attribute to true, the select list accepts multiple selections. If it is not set, the select list accepts only a single selected option. The default value is false.

Attribute	Explanation
name=*nmtoken*	The name attribute indicates the name of the variable that gets the value of the chosen item. The variable is set to the string value of the chosen option element, which is specified with the value attribute. The name variable's value is used to pre-select options in the select list.
value=*vdata*	The value attribute indicates the default value of the variable specified by the name attribute. If the variable specified by the name attribute does not have a value when the card is displayed, the user agent assigns it the value specified in the value attribute. If the name variable already contains a value, the value attribute is ignored. Note that any application of the default value is done before the list is pre-selected with the value of the name variable. If this element allows the selection of multiple options, the result of the user's choice is a list of all the selected values, separated by semicolons. The name variable is set with this result. In addition, the value attribute is interpreted as a list of pre-selected options separated by semicolons
iname=*nmtoken*	The iname attribute indicates the name of a variable containing an index number. The user agent uses the index number to set the default option. The number 1 specifies the first item, the number 2 the second item, and so on. An index of zero indicates that no option is selected. Index numbering begins at one and increases monotonically.
ivalue=*vdata*	The ivalue attribute indicates the index of the option element selected by default. If the variable specified by the iname attribute is not set when the card is displayed, it is assigned the entry selected by default. If the variable already contains a value, the ivalue attribute is ignored. If the iname attribute is not specified, the ivalue value is applied every time the card is displayed.

Attribute	Explanation
	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 semicolons (for example, 1;2). The `iname` variable is set with this result. In addition, the `ivalue` attribute is interpreted as a list of pre-selected options separated by semicolons.
`title=`*`vdata`*	This attribute specifies a title for the `select` element, which may be used in the presentation of this object.
`tabindex=`*`number`*	The `tabindex` element 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.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [36.]

The following example specifies a multiple-choice list. Note the following:

The "dog" and "cat" options would be pre-selected if the variable "I" had not been previously set.

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".

```
<wml>
   <card>
      <p>
         Please choose <i>all</i> of your favorite animals:
         <select name="X" iname="I" ivalue="1;2" multiple="true">
         <option value="D">Dog</option>
         <option value="C">Cat</option>
         <option value="H">Horse</option>
         </select>
      </p>
   </card>
</wml>
```

The deck generates the following user interface in the user agent (as shown on a 6150 model phone):

A card with a multiple-choice list.

option element

Description

The `option` element specifies a single choice option in a `select` element.

Contained elements

`onevent *`

Syntax

The attributes of the `option` element are explained in the following table.

Attribute	Explanation
`value=`*`vdata`*	This 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 `name` variable.

Attribute	Explanation
	The `value` attribute may contain variable references, which are evaluated before the `name` variable is set.
`title=`*`vdata`*	This attribute specifies a title for the `option` element, which may be used in the presentation of this object.
`onpick=`*`href`*	The `onpick` event occurs when the user selects or deselects this option. A multiple-selection option list generates an `onpick` event whenever the user selects or deselects this option. A single-selection option list generates an `onpick` event when the user selects this option, that is, no event is generated for the de-selection of any previously selected option.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [37.]

The following example specifies a simple single-choice list. If the user were to choose the “dog” option, the variable “X” would be set to a value of “D”.

See also:

Example [36.] on page 63.

Example [38.] on page 67.

```
<wml>
   <card>
      <p>
         Please choose your favorite animal:
         <select name="X">
            <option value="D">Dog</option>
            <option value="C">Cat</option>
         </select>
      </p>
      </card>
</wml>
```

The deck generates the following user interface in the user agent (as shown on a 6150 model phone):

A card with a single-choice list.

The next example specifies a single-choice list with a default value.

Note the following:

The "dog" option would be pre-selected if the "I" variable had not been previously set.

If the user were to choose the "cat" option, the variable "I" would be set to a value of "2".

```
<wml>
   <card>
      <p>
         Please choose your favorite animal:
         <select iname="I" ivalue="1">
            <option value="D">Dog</option>
            <option value="C">Cat</option>
         </select>
      </p>
   </card>
</wml>
```

optgroup element

Description

The `optgroup` element allows you to group related `option` elements into a hierarchy. The user agent uses this hierarchy to facilitate layout and presentation.

You must include one of the following elements at least once inside an `optgroup` element:

`optgroup` (nested element)

`option`

Contained elements

```
(optgroup|option) +
```

Syntax

The attributes of the `optgroup` element are explained in the following table.

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a title for the `optgroup` element, which may be used in the presentation of this object.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [38.]

The following example demonstrates the use of option groups. The card contains two geographical groups, “Scandinavia” and “Europe”.

```
<wml>
   <card id="card1" title="Country">
      <p>
         Select a country:
         <select name="country" multiple="true" tabindex="2">
            <optgroup title="Scandinavia">
               <option value="den">Denmark</option>
               <option value="fin">Finland</option>
               <option value="nor">Norway </option>
               <option value="swe">Sweden </option>
            </optgroup>
            <optgroup title="Europe">
               <option value ="fra">France </option>
```

```
            <option value ="ger">Germany</option>
            <option value ="ita">Italy  </option>
            <option value ="spa">Spain  </option>
          </optgroup>
        </select>
      </p>
      </card>
</wml>
```

The deck generates the following user interface in the user agent (as shown on a 6110 model phone):

A deck with option groups.

fieldset element

Description

The `fieldset` element allows you to group related fields and text. The grouping provides information to the user agent for optimizing layout and navigation. `fieldset` elements may be nested, providing the user with a means of specifying behavior across a wide variety of devices. For information on how the `fieldset` element may influence layout and navigation, see “card element” on page 26.

Contained elements

`input` *
`select` *
`fieldset` * (nested element)
`a` *
`img` *
`tab` *
`br` *
`(em|strong|b|i|u|big|small)` *

Syntax

The attributes of the `fieldset` element are explained in the following table.

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a title for the `fieldset` element, which may be used in the presentation of this object.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [39.]

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 id="info" title="Personal Info">
      <do type="accept" label="Submit">
         <go href="/submit?f=$(fname)&amp;l=$(lname)&amp;
                  s=$(sex)&amp;a=$(age)"/>
      </do>

      <p>
         <fieldset title="Name">
         First name: <input type ="text" name="fname"
maxlength="32"/><br/>
         Last name: <input type="text" name="lname" maxlength="32"/><br/>
         </fieldset>

         <fieldset title="Info">
            <select name="sex">
               <option value="F">Female</option>
               <option value="M">Male</option>
            </select>
            <br/>
            Age: <input type="text" name="age" format="*N"/>
         </fieldset>
      </p>
   </card>
</wml>
```

Anchors, images and timers

The following sections provide information on the anchor, image and timer elements of WML.

anchor element

Description

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

You can use anchors anywhere formatted text is legal, except for `option` elements.

An anchored link must have an associated task that specifies the behavior when the anchor is selected. You must anchor one of the following task elements to a link:

```
go
prev
refresh
```

Note that it is a WML syntax error to specify more than one task in an `anchor` element.

Contained elements

```
(go|prev|refresh|br|img) *
```

Syntax

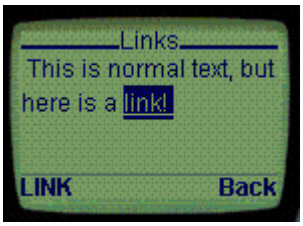
The attributes of the `anchor` element are explained in the following table.

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a brief text string identifying the link. For it to work well with a broad range of user agents, keep your labels under six characters.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [40.]

```
<wml>
<card id="links" title="Links">
<p>
      This is normal text, but here is a
      <anchor title="LINK">link!
        <go href="dir/file.wml">
          <setvar name="var_name" value="var_value"/>
        </go>
      </anchor>
    </p>
  </card>
</wml>
```

The deck generates the following user interface in the user agent:



A card containing a link.

a element

Description

The a element is a short form of the anchor element and is bound to a go task without variables. For example, the following markup:

```
<anchor>follow me
   <go href="destination.wml"/>
</anchor>
```

is the same as:

```
<a href="destination.wml"> follow me</a>
```

It is a WML syntax error to nest a elements.

Contained elements

```
(br|img) *
```

Syntax

The attributes of the a element are explained in the following table.

Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a brief text string identifying the link. For it to work well with a broad range of user agents, keep your labels under 6 characters.
`href`	This attribute specifies the destination URI, for example, the URI of the card to display.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

img element

Description

You can place images in the text flow by using the `img` element. Images use the same layout as normal text.

Contained elements

None.

Syntax

The attributes of the `img` element are explained in the following table.

Attribute	Explanation
`alt=`*`vdata`	This attribute specifies an alternative textual representation for the image used when the image cannot be displayed using any other method, that is, the image contents cannot be found, or the user agent does not support image display.

Attribute	Explanation
src=*href*	This attribute specifies the URI for the image. The browser downloads the image from the specified URI 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 URI specified in the SRC attribute, that is, any localsrc attribute specified takes precedence over the image specified in the src attribute.
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 short length that is not 0. Note that if you specify length as a percentage value, the resulting size is based on the available horizontal or vertical space, not on the natural size of the image.
align= (top|middle|bottom)	This attribute specifies image alignment within the text flow with respect to the current insertion point. align has three possible values: bottom — The bottom of the image is vertically aligned with the current baseline. This is the default value. middle — The center of the image is vertically aligned with the center of the current text line. top — The top of the image is vertically aligned with the top of the current text line.
height=*length* width=*length*	These attributes specify the size of an image or object. User agents may scale objects and images to match these values if appropriate. Note that if you specify length as a percentage value, the resulting size is based on the available horizontal or vertical space, not on the natural size of the image.
xml:lang	See "Common attributes" on page 23.

Attribute	Explanation
id	See "Common attributes" on page 23.
class	See "Common attributes" on page 23.

Example [41.]

The following example illustrates the use of the img element. Note the following:

src="bitmaps/moon.wbmp" refers to the file moon.wbm pthat is located in the bitmaps directory in the same domain as the deck in which the img element is included.

One unit of white space is inserted to the left and right (hspace) as well as above and below (vspace) the image.

```
<img src="bitmaps/moon.wbmp" alt="Moon" src="default.wbmp" hspace="1"
vspace="1"/>
```

See Example [45.] on page 81 for another example using the img element.

timer element

Description

The timer element implements a card timer, which can be used to process inactivity or idle time. The timer is initialized and started at card entry and 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 the execution of any task. 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. Note that it is a WML syntax error to have more than one timer in a card.

The timer timeout value is specified in units of one-tenth (1/10) of a second. However, note that you should not expect a particular timer resolution. It is thus recommended that in applications where exact timer resolution is required, you provide the user agent with another means to invoke a timer's task. Note also that if the value of the timeout is not a positive integral number, the user agent ignores the timer element. A timeout value of zero (0) disables the timer.

Contained elements

None.

Syntax

The attributes of the `timer` element are explained in the following table.

Attribute	Explanation
`name=`*`nmtoken`*	This attribute specifies the name of the variable to be set with the value of the timer. The `name` variable's value is used to set the timeout period upon timer initialization. The variable named by the `name` attribute will be set with the current timer value when the card is exited or when the timer expires. For example, if the timer expires, the `name` variable is set to a value of "`0`".
`value=`*`vdata`*	This attribute indicates the default value of the variable named in the `name` attribute. When the timer is initialized and the variable named in the `name` attribute is not set, the `name` variable is assigned the value specified in the `value` attribute. If the `name` variable already contains a value, the `value` attribute is ignored. If the `name` attribute is not specified, the timeout is always initialized to the value specified in the `value` attribute. This attribute is required.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [42.]

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

```
<wml>
   <card ontimer="/next">
      <timer value="100"/>
      <p>
         Hello World!
      </p>
   </card>
</wml>
```

The same example could be implemented as:

```
<wml>
   <card>
      <onevent type="ontimer">
         <go href="/next"/>
      </onevent>
      <timer value="100"/>
      <p>
```

```
         Hello World!
      </p>
   </card>
</wml>
```

The following example illustrates how a timer can initialize and re-use a counter. Each time the card is entered, the timer is reset to the 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 name="t" value="50"/>
      <p>
         Hello World!
      </p>
   </card>
</wml>
```

Text formatting

This section introduces the elements and constructs related to text formatting.

White space

The way that WML handles white space and line breaks is based on XML and assumes the default white space handling rules. This means that the WML user agent converts multiple contiguous spaces, returns and lines into a single space between words.

Emphasis elements

The emphasis elements specify text emphasis markup information. The emphasis tags are explained in the following table.

Tag	Explanation
`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.

Use the `strong` and `em` tags where possible. It is not recommended to use `b`, `i`, and `u` tags except where explicit control over text presentation is required.

Example [43.]

The following WML illustrates the use of text emphasis tags.

```
<wml>

   <card id="card1">
      <p>
         <em>
            A
               <u>
                  Demonstration
               </u>
            of Nokia's
         <i>
          <strong> Wireless Application Protocol<br/> </strong>
       </i>
         <b> Toolkit</b>
         </em>
      </p>
   </card>

</wml>
```

The deck generates the following user interface in the user agent:

Card with text formatting.

Note that the user agent can only display a few lines at a time so you must scroll down to see the last lines of the card.

br element

Description

The `br` element establishes the beginning of a new line. The user agent must break the current line and continue on the following line. User agents should do best effort to support the `br` element in tables (see "table element" on page 79).

Contained elements

None.

Syntax

The line break attributes are explained in the following table.

Attribute	Explanation
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

p element

Description

The `p` element establishes both the line wrap and alignment parameters for a paragraph. If the text alignment is not specified, it defaults to `left`. If the wrap mode is not specified, it is identical to the line-wrap mode of the previous paragraph in the current card. Empty paragraphs, such as an empty element or an element with only insignificant white space, should be considered as insignificant and ignored by visual user agents. Insignificant paragraphs do not impact line-wrap mode. If the first `p` element in a card does not specify a line-wrap or alignment mode, that mode defaults to the initial mode of the card. The user agent must insert a line break into the text flow between significant `p` elements.

You may remove insignificant paragraphs before delivering the document to the user agent.

WML has two line-wrapping modes for visual user agents: breaking (or wrapping) and non-breaking (or non-wrapping). The treatment of a line too long to fit on the screen is specified by the current line-wrap mode.

If `mode="wrap"` is specified, the line is word-wrapped onto multiple lines. In this case, line breaks should be inserted into a text flow as appropriate for presentation on an individual device.

If `mode="nowrap"` is specified, the line is not wrapped. automatically. In this case, the user agent must provide a mechanism to view entire non-wrapped lines, such as horizontal scrolling or some other user-agent specific mechanism.

Any space between words is a legal line break point. The non-breaking space entity (` ` or ` `) indicates that the user agent must not treat the space as a space between words. It is recommended that you use ` ` to prevent unwanted 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 inserts a hyphen character (`&#45;`) at the end of the line. In all other operations, the soft-hyphen entity is ignored. Note also that a user agent may ignore soft-hyphens when formatting text lines.

Contained elements

```
a
anchor
br
do
em|strong|b|i|u|big|small
fieldset
img
input
select
table
```

Syntax

The attributes are explained in the following table.

Attribute	Explanation
`align=(left|right|center)`	This attribute specifies the text alignment mode for the paragraph. You can align the text center, left or right. The default alignment is left. If not explicitly specified, the text alignment is set to the default alignment.
`mode=(wrap|nowrap)`	This attribute specifies the line-wrap mode for the following paragraph. `Wrap` specifies breaking text mode. `Nowrap` specifies non-breaking text mode. If not explicitly specified, the line wrap mode is identical to the line wrap mode of the previous paragraph in the text flow of a card. The default mode for the first paragraph in a card is `wrap`.
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [44.]

The following example illustrates centering and left-justifying text as well as the difference between wrapping and not wrapping text.

```
<p align="center">
   centered text
</p>

<p align="left">
   left-justified
</p>

<p mode="wrap">
   This text is wrapped to the next line.
</p>

<p mode="nowrap">
   This text is truncated.
</p>
```

table element

Description

The `table` element is used together with the `tr` and `td` elements to create sets of aligned columns of text and images in a card. You cannot nest `table` elements. The `table` elements determine the structure of the columns. The elements separate content into columns, but do not specify column or intercolumn widths. The user agent should do its best effort to present the information of the table in a manner appropriate to the device.

The alignment of the text and images within a column is specified by the `align` attribute. You can align the contents of a column center, left, or right. The `align` attribute value is interpreted as a list of alignment designations, one for each column. You specify center alignment with the value "C", left alignment with the value "L", and right alignment with the value "R". The first designator in the list applies to the first column, the second designator to the second column, and so on. The default alignment is applied for any column for which an alignment designation is omitted. For left-to-right languages, the default alignment is left. For right-to-left languages, the default alignment is right.

You must use the `columns` attribute to specify the number of columns for the row set. The user agent must create a row set with exactly the number of columns specified by the `columns` attribute value. If the actual number of columns in a row is less than the value specified in the `columns` attribute, the row must be effectively padded with empty columns. The orientation of the table depends on the language: for left-to-right languages, the leftmost column is the first column in the table. Columns are added to the right side of a row to pad left-to-right tables. Columns are added to the left side of a row to pad right-to-left tables.

If the actual number of columns in a row is greater than the value specified by the `columns` attribute, the extra columns of the row must be aggregated into the last column, such that the row contains exactly the number of columns specified. A single inter-word space must be inserted between two cells that are being aggregated.

Depending on the display characteristics, the user agent may create aligned columns for each table, or may use a single set of aligned columns for all tables 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.

Contained elements

`tr +`

Syntax

The attributes are explained in the following table.

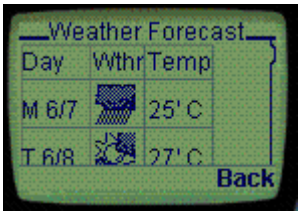
Attribute	Explanation
`title=`*`vdata`*	This attribute specifies a title for this element, which may be used in the presentation of this object.
`align=`*`cdata`*	This attribute specifies the layout of text and images within the columns of a row set. You can align the contents of a column center, left, or right. The attribute value is interpreted as a list of alignment designations, one for each column. You specify center alignment with the value "C", left alignment with the value "L", and right alignment with the value "R".
`columns=`*`number`*	This attribute specifies the number of columns for the row set. The user agent must create a row set with exactly the number of columns specified by the attribute value. Note that it is a WML syntax error to specify a value of zero ("0").
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Example [45.]

The following example contains a card with a table that has six rows and three cells of data in each row. On rows two through 6, the second cell contains a wireless bitmap.

```
<wml>

   <card id="card1" title="Weather Forecast">
      <p>
         <table columns="3">
            <tr>
               <td>Day</td><td>Wthr</td><td>Temp</td>
            </tr>
            <tr>
               <td>M 6/7</td><td><img src="rainy.wbmp" alt="rain"/></td>
               <td>25' C</td>
            </tr>
            <tr>
               <td>T 6/8</td><td><img src="partcldy.wbmp" alt="part
cldy"/></td>
               <td>27' C</td>
            </tr>
            <tr>
               <td>W 6/9</td><td><img src="cloudy.wbmp"
alt="cloudy"/></td>
               <td>24' C</td>
            </tr>
            <tr>
               <td>T 6/10</td><td><img src="rainy.wbmp" alt="rainy"/></td>
               <td>28' C</td>
            </tr>
            <tr>
               <td>F 6/11</td><td><img src="sunny.wbmp" alt="sunny"/></td>
               <td>29' C</td>
            </tr>
         </table>
      </p>
   </card>

</wml>
```

The deck generates the following user interface in the user agent (shown in a 6110 model phone):



tr element

Description

The `tr` element is used as a container to hold a single table row. Table rows may be empty (for example, all the cells in the row are empty). Empty rows are significant and must not be ignored.

Contained elements

`td +`

Syntax

The attributes are explained in the following table.

Attribute	Explanation
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

td element

Description

The `td` element is used as a container to hold a single table cell data within a table row. Table data cells may be empty. Empty cells are significant, and must not be ignored. The user agent should do a best effort to deal with multiple line data cells that may result from using images or line breaks.

Contained elements

a
anchor
br
`em|strong|b|i|u|big|small`
`img`

Syntax

The attributes are explained in the following table.

Attribute	Explanation
`xml:lang`	See "Common attributes" on page 23.
`id`	See "Common attributes" on page 23.
`class`	See "Common attributes" on page 23.

Appendix A

Examples

This Appendix introduces two very general examples of WML.

Using variables

Example [46.]

The following example demonstrates how to reference variables and set variables as a side effect of navigation.

This example illustrates the use of the following WML elements:

```
card
do
go
prev
setvar
onevent
refresh
```

The example illustrates the use of the following attribute:

```
newcontext
```

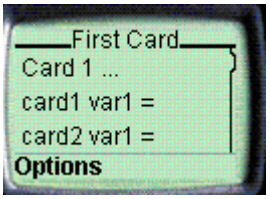
WML code

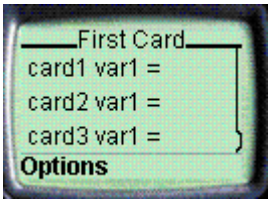
```
<!—deck1.wml -->

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>

   <card id="card1" title="First Card" newcontext="true">
   <p>

      Card 1 ... <br/>

      <!-- The following variables will not be defined
           until the other cards in this deck are entered.
       -->
```

```
      card1 var1 = $(card1_var1) <br/>
      card2 var1 = $(card2_var1) <br/>
      card3 var1 = $(card3_var1) <br/>

      <do type="accept" label="Next Card">
         <go href="#card2">
            <setvar name="card1_var1" value="val_1"/>
         </go>
      </do>

   </p>
   </card>

   <card id="card2" title="Second Card">
   <p>

      Card 2 ... <br/>

      card1 var1 = $(card1_var1) <br/>
      card2 var1 = $(card2_var1) <br/>
      card3 var1 = $(card3_var1) <br/>

      <do type="accept" label="First Card">
         <go href="#card1"/>
      </do>

      <do type="accept" label="Third Card">
         <go href="#card3">
            <setvar name="card2_var1" value="val_2"/>
         </go>
      </do>

      <do type="prev" label="Previous Card">
         <prev/>
      </do>

   </p>
   </card>

   <card id="card3" title="Third Card">

      <onevent type="onenterforward">
         <refresh>
            <setvar name="card3_var1" value="val_3"/>
         </refresh>
      </onevent>
      <p>

         Card 3 ... <br/>

         card1 var1 = $(card1_var1) <br/>
         card2 var1 = $(card2_var1) <br/>
         card3 var1 = $(card3_var1) <br/>

         <do type="prev" label="Previous">
            <prev/>
         </do>

      </p>
   </card>

</wml>
```

Explanation

In the user agent, this example generates the following user interface when you enter the deck (you may need to scroll down to see the entire screen, as shown in this example on a 6150 model phone):

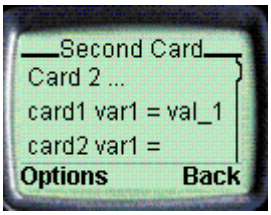


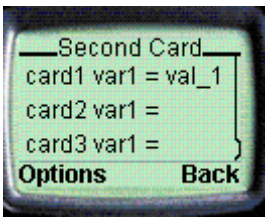


Note that none of the variables has a value at this stage.

By pressing the **Options** button, you can select the **Next Card**.

Press the **Select** button to choose the **Next Card** and go to the second card of the deck.





Note that now the variable `card1 var1` has the value `val_1` but the other two variables do not have values yet.

Pressing **Options** sends you to the options card specified by the `do`elements in `card2`:

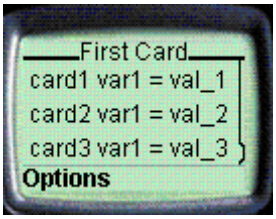
You can now navigate to the first or third card:

Use the button with the up and down arrow to select the card that you want to display.

Press **Select**.

When you enter the third card in the deck, it will display all three variables with their values.

You can now navigate back to the first card by pressing the **Back** button twice. Now the first card contains values for all three variables:



Task shadowing and inter-deck navigation

Example [47.]

The next two example decks demonstrate how to implement card and deck task shadowing and inter-deck navigation in WML services. They also illustrate the use of relative URLs.

The example illustrates the use of the following WML elements:

```
access
head
meta
template
noop
```

WML code

```
<!-- deck2a.wml -->

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">

<wml>
   <template>
      <do type="prev" name="Previous" label="Previous">
         <prev/>
      </do>
   </template>

   <card id="card1" title="First Card" newcontext="true">
   <p>
      Card 1 ... <br/>
      <do type="accept" label="Next Card">
         <go href="#card2"/>
      </do>
      <!-- Must override the DO/PREV in the template to
           prevent the PREV element from going back to
           the previous deck
      -->
      <do type="prev" name="Previous">
         <noop/>
      </do>
   </p>
   </card>
   <card id="card2" title="Second Card">
   <p>
      Card 2 ... <br/>
      <do type="accept" label="Next Card">
         <go href="#card3"/>
      </do>
   </p>
   </card>

   <card id="card3" title="Third Card">
   <p>
      Card 3 ... <br/>
      <!-- NOTE: the following DO element will go to a new
                 deck.  The name of this deck is embedded
               in the URL.
      -->
      <do type="accept" label="Next Deck">
         <go href="deck2b.wml"/>
      </do>
   </p>
   </card>
</wml>
```

Explanation

When you navigate to the first card of the deck, the card displays only the **Next Card** soft key, since the **Previous** item described in the template has been overridden with a **Previous** item in the card specifying a `noop` task. (Note that this example is shown on a 6110 model phone.)

Pressing **Next Card** sends you to the second card of the deck. The card contains the **Next Card** item specified in the card and the **Previous** item specified in the template:

Pressing **Next Card** sends you to the third card. The card contains the **Next Deck** item specified in the card and the **Previous** item specified in the template:

The **Next Deck** menu item allows you to navigate to `deck2b.wml`, which contains a **Previous** soft key, allowing you to navigate to the card you visited last:

Summary of examples

The following table lists the examples used in this guide and the page on which you can them.

Appendix B

WML document type definition

This Appendix provides the WML document type definition (DTD).

```
<!--
Wireless Markup Language (WML) Document Type Definition.

Copyright Wireless Application Protocol Forum Ltd., 1998,1999.
                     All rights reserved.

WML is an XML language.  Typical usage:
   <?xml version="1.0"?>
   <!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
          "http://www.wapforum.org/DTD/wml_1.1.xml">
   <wml>
   ...
   </wml>

Terms and conditions of use are available from the Wireless
Application Protocol Forum Ltd. web site at
http://www.wapforum.org/docs/copyright.htm.
-->

<!ENTITY % length  "CDATA">    <!-- [0-9]+ for pixels or [0-9]+"%" for
                                    percentage length -->
<!ENTITY % vdata   "CDATA">    <!-- attribute value possibly containing
                                    variable references -->
<!ENTITY % HREF    "%vdata;">  <!-- URI, 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 % coreattrs  "id    ID     #IMPLIED
                       class CDATA  #IMPLIED">

<!ENTITY % emph    "em | strong | b | i | u | big | small">
<!ENTITY % layout  "br">

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

<!-- flow covers "card-level" elements, such as text and images -->
<!ENTITY % flow     "%text; | %layout; | img | anchor | a | table">

<!-- 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
  %coreattrs;
  >

<!-- card intrinsic events -->
<!ENTITY % cardev
 "onenterforward  %HREF;          #IMPLIED
  onenterbackward %HREF;          #IMPLIED
  ontimer         %HREF;          #IMPLIED"
  >

<!-- card field types -->
<!ENTITY % fields  "%flow; | input | select | fieldset">

<!ELEMENT card (onevent*, timer?, (do | p)*)>
<!ATTLIST card
  title           %vdata;       #IMPLIED
  newcontext      %boolean;     "false"
  ordered         %boolean;     "true"
  xml:lang        NMTOKEN       #IMPLIED
  %cardev;
  %coreattrs;
  >

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

<!ELEMENT do (%task;)>
<!ATTLIST do
  type         CDATA       #REQUIRED
  label        %vdata;     #IMPLIED
  name         NMTOKEN     #IMPLIED
  optional     %boolean;   "false"
  xml:lang     NMTOKEN     #IMPLIED
  %coreattrs;
  >

<!ELEMENT onevent (%task;)>
<!ATTLIST onevent
  type         CDATA       #REQUIRED
  %coreattrs;
  >

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

<!ELEMENT head ( access | meta )+>
<!ATTLIST head
  %coreattrs;
  >

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

<!ELEMENT access EMPTY>
<!ATTLIST access
  domain       CDATA       #IMPLIED
  path         CDATA       #IMPLIED
  %coreattrs;
  >

<!ELEMENT meta EMPTY>
<!ATTLIST meta
  http-equiv     CDATA      #IMPLIED
  name           CDATA      #IMPLIED
  forua          %boolean;  #IMPLIED
  content        CDATA      #REQUIRED
  scheme         CDATA      #IMPLIED
  %coreattrs;
  >
```

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

<!ELEMENT go (postfield | setvar)*>
<!ATTLIST go
  href            %HREF;           #REQUIRED
  sendreferer     %boolean;       "false"
  method          (post|get)      "get"
  accept-charset  CDATA           #IMPLIED
  %coreattrs;
  >

<!ELEMENT prev (setvar)*>
<!ATTLIST prev
  %coreattrs;
  >

<!ELEMENT refresh (setvar)*>
<!ATTLIST refresh
  %coreattrs;
  >

<!ELEMENT noop EMPTY>
<!ATTLIST noop
  %coreattrs;
  >

<!--================ postfield ================-->

<!ELEMENT postfield EMPTY>
<!ATTLIST postfield
  name         %vdata;           #REQUIRED
  value        %vdata;           #REQUIRED
  %coreattrs;
  >

<!--================ variables ================-->

<!ELEMENT setvar EMPTY>
<!ATTLIST setvar
  name         %vdata;           #REQUIRED
  value        %vdata;           #REQUIRED
  %coreattrs;
  >

<!--================ Card Fields ================-->

<!ELEMENT select (optgroup|option)+>
<!ATTLIST select
  title        %vdata;           #IMPLIED
  name         NMTOKEN           #IMPLIED
  value        %vdata;           #IMPLIED
  iname        NMTOKEN           #IMPLIED
  ivalue       %vdata;           #IMPLIED
  multiple     %boolean;         "false"
  tabindex     %number;          #IMPLIED
  xml:lang     NMTOKEN           #IMPLIED
  %coreattrs;
  >

<!ELEMENT optgroup (optgroup|option)+ >
<!ATTLIST optgroup
  title      %vdata;    #IMPLIED
  xml:lang   NMTOKEN    #IMPLIED
  %coreattrs;
  >

<!ELEMENT option (#PCDATA | onevent)*>
<!ATTLIST option
  value      %vdata;    #IMPLIED
  title      %vdata;    #IMPLIED
  onpick     %HREF;     #IMPLIED
  xml:lang   NMTOKEN    #IMPLIED
  %coreattrs;
  >
```

```
<!ELEMENT input EMPTY>
<!ATTLIST input
  name       NMTOKEN           #REQUIRED
  type       (text|password)   "text"
  value      %vdata;           #IMPLIED
  format     CDATA             #IMPLIED
  emptyok    %boolean;         "false"
  size       %number;          #IMPLIED
  maxlength  %number;          #IMPLIED
  tabindex   %number;          #IMPLIED
  title      %vdata;           #IMPLIED
  xml:lang   NMTOKEN           #IMPLIED
  %coreattrs;
  >

<!ELEMENT fieldset (%fields; | do)* >
<!ATTLIST fieldset
  title          %vdata;      #IMPLIED
  xml:lang       NMTOKEN      #IMPLIED
  %coreattrs;
  >

<!ELEMENT timer EMPTY>
<!ATTLIST timer
  name       NMTOKEN           #IMPLIED
  value      %vdata;           #REQUIRED
  %coreattrs;
  >

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

<!ENTITY % IAlign "(top|middle|bottom)" >

<!ELEMENT img EMPTY>
<!ATTLIST img
  alt        %vdata;      #REQUIRED
  src        %HREF;       #REQUIRED
  localsrc   %vdata;      #IMPLIED
  vspace     %length;     "0"
  hspace     %length;     "0"
  align      %IAlign;     "bottom"
  height     %length;     #IMPLIED
  width      %length;     #IMPLIED
  xml:lang   NMTOKEN      #IMPLIED
  %coreattrs;
  >

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

<!ELEMENT anchor ( #PCDATA | br | img | go | prev | refresh )*>
<!ATTLIST anchor
  title        %vdata;      #IMPLIED
  xml:lang     NMTOKEN      #IMPLIED
  %coreattrs;
  >

<!ELEMENT a ( #PCDATA | br | img )*>
<!ATTLIST a
  href         %HREF;       #REQUIRED
  title        %vdata;      #IMPLIED
  xml:lang     NMTOKEN      #IMPLIED
  %coreattrs;
  >

<!--================ Tables ================-->

<!ELEMENT table (tr)+>
<!ATTLIST table
  title       %vdata;    #IMPLIED
  align       CDATA      #IMPLIED
  columns     %number;   #REQUIRED
  xml:lang    NMTOKEN    #IMPLIED
  %coreattrs;
  >
```

```
<!ELEMENT tr (td)+>
<!ATTLIST tr
  %coreattrs;
  >

<!ELEMENT td ( %text; | %layout; | img | anchor | a )*>
<!ATTLIST td
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

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

<!ELEMENT em     (%flow;)*>
<!ATTLIST em
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT strong (%flow;)*>
<!ATTLIST strong
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT b      (%flow;)*>
<!ATTLIST b
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT i      (%flow;)*>
<!ATTLIST i
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT u      (%flow;)*>
<!ATTLIST u
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT big    (%flow;)*>
<!ATTLIST big
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ELEMENT small  (%flow;)*>
<!ATTLIST small
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!ENTITY % TAlign "(left|right|center)">
<!ENTITY % WrapMode "(wrap|nowrap)" >
<!ELEMENT p (%fields; | do)*>
<!ATTLIST p
  align       %TAlign;     "left"
  mode        %WrapMode;   #IMPLIED
  xml:lang    NMTOKEN      #IMPLIED
  %coreattrs;
  >

<!ELEMENT br EMPTY>
<!ATTLIST br
  xml:lang        NMTOKEN       #IMPLIED
  %coreattrs;
  >

<!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) -->
<!--
Copyright Wireless Application Protocol Forum Ltd., 1998,1999.
                      All rights reserved.
-->
```

Appendix C

WML quick reference

Header

```
<?xml version="1.0"?>
```

Decks and cards

```
<wml xml:lang="lang">
   content
</wml>

<card id="name" title="label" newcontext="boolean" ordered="boolean"
   onenterforward="href" onenterbackward="href" ontimer="href">
      content
</card>

<template onenterforward="href" onenterbackward="href" ontimer="href">
   content
</template>

<head> content </head>

<access domain="domain" path="path" />

<meta content="value" scheme="format"/>
```

Events

```
<do type="type" label="label" name="name" optional="boolean">
   task
</do>

<onevent type="type"> task </onevent>

<postfield name="value" value="value">
```

Tasks

```
<go href="href" sendreferer="boolean" method="method"
   accept-charset="charset" >
      content
</go>

<prev> content </prev>

<noop/>

<refresh> content </refresh>
```

User input

```
<input name="variable" title="label" type="type" value="value"
   value="default" format="mask" emptyok="boolean" size="n"
   maxlength="n" tabindex="n"/>

<select title="label" multiple="boolean" name="variable" value="default"
   iname="index_var" ivalue="default" tabindex="n">
      content
</select>

<option title="label" value="value" onpick="href">
   content
</option>

<optgroup title="label"> content </optgroup>

<fieldset title="label"> content </fieldset>
```

Anchors

```
<anchor title="label"> task text</anchor>

<a title="label"> href="href" text </a>
```

Images

```
<img alt="text" src="url" localsrc="icon" align="alignment"
   height="n" width="n" vspace="n" hspace="n"/>
```

Timers

```
<timer name="variable" value="value"/>
```

Variables

```
<setvar name="name" value="value"/>
```

Layout and formatting

```
<br>

<p align="alignment" mode="wrapmode"/>

<table title="value" align="alignment" columns="number"/>
<tr> </tr>
<td>text layout img anchor a </td>

<em> text </em>
<strong> text </strong>
<b> text </b>
<i> text </i>
<u> text </u>
<big> text </big>
<small> text </small>
```

Special characters

`&quot;`	quotation mark
`&`	ampersand
`&apos;`	apostrophe
`<`	less than
`>`	greater than
` `	non-breaking space
`&shy;`	soft hyphen (discretionary hyphen)

Glossary

The following terms and conventions are used throughout this document.

American Standard Code for Information Interchange (ASCII)

ASCII is a standard developed by the American National Standards Institute (ANSI) to define computer-intelligible values for characters used in text. The ASCII set of 128 characters includes uppercase and lowercase letters of the English alphabet, numbers, punctuation, and 33 control codes (such as tab, bell, carriage return). ASCII uses 7 bits to represent each character. You may see ASCII characters identified by a decimal number from 0 to 127.

The standard ASCII character set uses just 7 bits for each character, consequently one bit of each octet is not used. Larger character sets, known as extended ASCII or high ASCII, use all 8 bits, allowing as many as 128 additional characters to be defined. Numerous extensions to ASCII have been devised and quite a few have become national or international standards. Notable among them is a family of international standards, ISO-8859, that defines extensions appropriate to certain language groups which ASCII alone cannot support. The most important member of this group is ISO-8859-1, known as ISO Latin-1, which provides for the languages of western Europe.

Attribute

A syntactical component of a WML element which is often used to specify a characteristic quality of an element, other than type or content.

Author

An author is a person or program that writes or generates WML, WMLScript or other content.

Bandwidth

Bandwidth is the capacity that a telecommunications medium has for carrying data. For analog or voice communication, bandwidth is measured in the difference between the upper and lower transmission frequencies expressed in cycles per second, or hertz (Hz). For digital communication, bandwidth and transmission speed are usually treated as synonyms and measured in bits per

second. The actual speed or transmission time of any message or file from origin to destination depends on a number of factors. Most Internet transmissions travel at very high speed on fiber optic lines most of the way. Switching en route, lower bandwidths on local loops at both ends, and server processing time add to the overall transmission time.

Byte

A sequence of consecutive bits treated as a unit. On almost all modern computers, a byte is comprised of 8 bits, though other numbers were formerly encountered. To avoid ambiguity, the term octet is used in the language of international standards to refer to an 8-bit unit.

Large amounts of memory are indicated in terms of kilobytes (1,024 bytes), megabytes (1,048,576 bytes), and gigabytes (approximately 1 billion bytes). A disk that can hold 1.44 megabytes, for example, is capable of storing approximately 1.4 million ASCII characters, or about 3,000 pages of information.

Bytecode

Content encoding where the content is typically a set of low-level opcodes, that is, instructions, and operands for a targeted piece of hardware or virtual machine.

Card

A single WML navigational and user interface unit. A card may contain information to present to the user or instructions for gathering user input, for example.

Character Encoding

When used as a verb, character encoding refers to conversion between sequence of characters and a sequence of bytes. When used as a noun, character encoding refers to a method for converting a sequence of bytes to a a sequence of characters. Typically, WML document character encoding is captured in transport headers attributes, meta information placed within a document, or the XML declaration defined by the XML specification.

Client

A device or application that initiates a request for connection with a server.

Common Gateway Interface (CGI)

A programming language that enables you to use forms on your web site.

Concatenation

Concatenating two strings means sticking them together, one after another, to make a new string. For example, the string “foo” concatenated with the string “bar” gives the string “foobar”.

Content

Subject matter stored or generated at a web 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 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.

Document Type Definition (DTD)

The document type definition states which elements can be nested within others. A DTD defines:

— The names and contents of all elements that are permissible in a certain document.

— How often an element may appear.

— The order in which the elements must appear.

— Whether the start or end tag may be omitted.

— The contents of all elements, that is, the names of the other generic identifiers that are allowed to appear inside them.

— The attributes and their default values.

— The names of the reference symbols that may be used.

Element

Elements specify all the markup and structural information for a WML deck. Elements may contain a start tag, content and an end tag.

Extensible Markup 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.

Hypertext transfer protocol (HTTP)

HTTP is the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands. For example, when you enter an URL in your browser, an HTTP command is sent to the web server directing it to retrieve and transmit the requested web page.

JavaScript™

A *de facto* standard language that can be used to add dynamic behaviour to HTML documents.

Markup

Text added to the data of a document to convey information on it. There are four different kinds of markup: descriptive markup (tags), references, markup declarations, and processing instructions.

Rendering

Formatting and presenting information.

Resource

A network data object or service that can be identified by an URL. Resources may be available in multiple representations (for example, 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.

Standardized Generalized Markup Language (SGML)

The Standardized Generalized Markup Language is a general-purpose language for domain-specific markup languages. SGML is defined in the *ISO8879* standard.

Tag

A tag is a generic term for a language element descriptor. WML consists of content surrounded by formatting tags. Each tag is enclosed in a pair of angle brackets:
< and >. Tags are generally used in pairs, one to start the element and one to end it.

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, for example, conversion from UCS-2 to UTF-8.

Unicode

An encoding scheme for written characters and text. Unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters, a huge increase over ASCII's code capacity of 128 characters. Unicode was authored and is maintained by the Unicode Consortium, a group comprised of major corporations and institutions involved in international computing. The character repertory and the codes assigned in Unicode are identical to those specified by *ISO 10646*, the international Universal Character Set (UCS) standard.

The Unicode Standard, Version 2.0 defines codes for characters used in every major language written today. In all, the Unicode standard currently defines codes for nearly 39,000 characters from the world's alphabetic, ideographic and syllabic scripts and symbol collections. The Unicode repertory was derived from many pre-existing character set standards to which previously unstandardized characters have been added. In particular, the first 256 code values are identical to those of ISO 8859-1 extended to 16 bits. Unicode values are displayed as four hex digits preceded by U+. For example, U+0041 is Latin uppercase A.

Uniform Resouce Identifier (URI)

Uniform Resource Identifiers (URI) identify resources in the web: documents, images, downloadable files, services, electronic mailboxes, and other resources. A URI can refer to an Uniform Resource Locator (URL) or an Uniform Resource Name (URN).

Uniform Resource Locator (URL)

URL stands for Uniform Resource Locator and is an address referring to a document on the Internet. The syntax of an URL consists of three elements:

— The protocol, or the communication language, that the URL uses.

— The domain name, or the exclusive name that identifies a web site.

— The pathname of the file to be retrieved.

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 piece of software or physical device that interprets WML, WMLScript, WTAI or other resources. They may include textual browsers, voice browsers and search engines, for example.

Web server

The server on which a given resource resides or is to be created. Often referred to as an origin server or an HTTP server.

Wireless Application Environment (WAE)

The Wireless Application Environment specifies a general-purpose application environment based fundamentally on World Wide Web technologies and philosophies. WAE specifies an environment that allows operators and service providers to build applications and services that can reach a wide variety of different platforms. WAE is part of the Wireless Application Protocol.

Wireless Application Protocol (WAP)

The Wireless Application Protocol specifies an application framework and network protocols for wireless devices such as mobile phones, pagers, and personal digital assistants (PDAs). The WAP specifications extend mobile networking technologies (such as digital data networking standards) and Internet technologies (such as XML, URLs, scripting, and various content formats).

Wireless Markup Language (WML)

The Wireless Markup Language is a markup language based on XML and is intended for use in specifying content and user interface for narrowband devices, including mobile phones and pagers.

Wireless Markup Language Script (WMLScript)

The Wireless Markup Language Script is a scripting language used to program the mobile device. WMLScript is an extended subset of the JavaScript™ scripting language.

Wireless Session Protocol (WSP)

The Wireless Session Protocol provides the upper-level application layer of WAP with a consistent interface for two session services. The first is connection-mode service that operates above a transaction layer protocol, and the second is a connectionless service that operates above a secure or non-secure datagram transport service.

Index

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X