



# Object Model Reference

**Openwave Usability Interface, Java Edition 1.0 Beta**

**Openwave Systems Inc.**  
1400 Seaport Boulevard  
Redwood City, CA 94063 U.S.A.  
<http://www.openwave.com>

Part Number OJAR-10-005  
October 2001

---

---

## **LEGAL NOTICE**

Copyright © 1999–2001 Openwave Systems Inc. All rights reserved.

The contents of this document constitute valuable proprietary and confidential property of Openwave Systems Inc. and are provided subject to specific obligations of confidentiality set forth in one or more binding legal agreements. Any use of this material is limited strictly to the uses specifically authorized in the applicable license agreement(s) pursuant to which such material has been furnished. Any use or disclosure of all or any part of this material not specifically authorized in writing by Openwave Systems Inc. is strictly prohibited.

Openwave, the Openwave logo and the family of terms carrying the “Openwave” prefix are trademarks of Openwave Systems Inc. All other trademarks are the properties of their respective owners.

All other company, brand, and product names are referenced for identification purposes only and may be trademarks that are the sole property of their respective owners.

---

# Contents

---

## **Preface v**

About This Guide v

Related Documentation v

Technical Support v

## **1 Object Model Reference 1**

Anchor 2

Appendix 4

BodyPager 5

Button 8

Caller 9

Card 11

Check 21

ComboMenu 24

Deck 26

DeviceContext 27

DoElement 30

Form 33

Head 42

HRule 45

Image 47

Input 50

Menu 53

Onevent 56

Picker 57

PickerCard 59

Popup 63

Radio 66

SimpleLink 69

Table 71

Task 75

TaskMenu 77

Template 80

Timer 81

---

## Contents

---

---

# Preface

---

## About This Guide

This guide is a reference to the Openwave™ Usability Interface (OUI), wireless application object model, expressed as a Java API. You can use the OUI objects to create a single Wireless Markup Language (WML) application that will operate optimally on various types of Wireless Application Protocol (WAP) devices. This guide assumes familiarity with WML and Java.

## Related Documentation

In addition to this guide, OUI comes with the following documentation:

- *Installation and Integration* describes how to install the OUI software and how to configure it to work with a Java server. It also includes a basic example servlet.
- *Getting Started* introduces OUI and describes how to use it.
- *XHTML Tab Library Reference* is a reference to the XHTML tags you can use in your OUI applications.
- *WML Tag Library Reference* is a reference to the WML tags you can use in your OUI applications.

You will also find valuable the *WML 1.3 Language Reference* and related documentation that come with the Openwave SDK. For a complete list of Openwave documentation for developers, see the Openwave Developer site:

<http://developer.openwave.com>

## Technical Support

Your best resource for information about OUI and other Openwave products, technologies, and solutions related to OUI is the Openwave Developer web site:

<http://developer.openwave.com/>

Openwave updates this site frequently to include late-breaking information.



---

# Object Model Reference

---

You can use the Openwave Usability Interface (OUI) Java API to create and enrich an object model that represents an ideally usable WML application. The objects that you use to build this model in this edition of OUI are instances of Java classes. This book is a reference these objects and their Java methods.

Some OUI objects are direct counterparts of WML elements. Examples include `Anchor`, `Deck`, `Input`, and `Timer`.

Other OUI objects abstract WML at a higher level, and may be rendered differently on different devices. Examples include `BodyPager`, `Menu`, and `PickerCard`.

For information about using OUI, this API, and the WML and XHTML tag libraries to build wireless applications, see the OUI *Getting Started* book and the tag library references.

For more information about WML, see the WML 1.3 *Developer's Guide, Language Reference*, and related documentation that come with the Openwave SDK.

## Anchor

The `Anchor` object encapsulates the WML `<anchor>` element and specifies a link to another card, deck, or resource. `Anchor` is similar to `SimpleLink` in many ways. The main difference is that any task can be associated with `Anchor`, while `SimpleLink` implicitly defines a `<go>` task.

`Anchor` also allows you to take advantage of the `accesskey` attribute, for those browsers that support it. For browsers that don't support it, `accesskey` isn't rendered.

### Example

```
Card myCard = new Card(...);
...
Task myTask = new Task("go", "songlist.asp", "get");

myTask.addPostfield("wholesite", "${wholesite}");

Anchor myAnchor = new Anchor(myTask, "Songlist");
myAnchor.setAccesskey(myCard.getAccessKey()); //unique accesskey provided by Card
...
myCard.addElement(myAnchor);
...
```

### Methods

Method	Description
<code>Anchor(AbsTask absTask, String text)</code>	Creates an instance of the <code>Anchor</code> object in which:  <code>absTask</code> is the task to be executed (such as <code>go</code> , <code>prev</code> , <code>noop</code> , or <code>refresh</code> ).  <code>text</code> is the text the device displays to represent the link to the URL.
<code>void setAccesskey(String accesskey)</code> <code>void setAccesskey(int accesskey)</code>	Direct counterpart of the Openwave extension <code>accesskey</code> attribute. This method maps a single numeric key to the link represented by this object. The parameter must be an integer or an integer passed as a string.
<code>void setText(String text)</code>	Sets the text that the device displays to represent the link. This method overwrites the text specified in the constructor.



Method	Description
<code>void setTitle(String title)</code>	<p>Direct counterpart of the WML <code>title</code> attribute. This method sets the label that identifies the link. If you do not specify the <code>title</code> attribute, the device uses the word <code>Link</code> as the default label. Devices use this attribute in a variety of ways. For example, they may use it to display a tool tip or to issue a voice prompt when the user selects the link. The Openwave text-based browser uses the title as the <code>ACCEPT</code> key label when the user selects the link. To ensure compatibility on a wide range of devices, the title should be five characters, or fewer.</p>

## Appendix

`Appendix` is used with a `ComboMenu` object. It encapsulates any text or input field that you want to append to a menu, for display on text-based browsers.

For more information, see “`ComboMenu`” on page 24.

### Methods

Method	Description
<code>Appendix(String shortTitle)</code>	Creates an instance of the <code>Appendix</code> object where <code>shortTitle</code> becomes the menu item on text-based browsers.
<code>Appendix(String id, String shortTitle)</code>	Creates an instance of the <code>Appendix</code> object in which: <code>id</code> is the identifier for the appendix card <code>shortTitle</code> is the title of the appendix card, which also becomes the last menu item of the <code>ComboMenu</code> object

## BodyPager

The `BodyPager` object does not represent any common WML constructs directly. It is a completely new object, designed to help you fully exploit the capabilities of each class of device. You can use `BodyPager` to split large amounts of text into ordered lists of “chunks,” which are grouped together to form a page.

There are two ways to use `BodyPager`:

- Have `BodyPager` split the text into sensible chunks for you
- Specify the chunks and the number of chunks to display per page yourself

### Example

This example demonstrates the use of `BodyPager` to sensibly split a long document into separate chunks, using the `autoSplitWML` method.

```
public void doGet ( HttpServletRequest request,
                  HttpServletResponse response )
    throws IOException, ServletException
{
    DeviceContext dc = new DeviceContext( request, response );

    String strLongText = new String();
    // Build a long text.
    strLongText = "Helena-Montana -Weather forecasters predicted "+
        "more triple-digit temperatures and lightning storms "+
        "in the Western "+
        "United States on Friday, conditions likely to hamper "+
        "efforts to contain about 60 wildfires "+
        "in 10 Western states." +
        "It's still going to be hot, with scattered "+
        "thunderstorms that won't produce much rain,"+
        "said CNN meteorologist Dave Hennen, "+
        "who predicted similar conditions continuing through the "+
        "weekend. <br/>"+
        "About 700,000 acres are burning throughout the West, "+
        "forcing hundreds of evacuations. "+
        "About half the scorched acreage is in Idaho, mostly in " +
        "national forests away from population centers." +
        "We were warned by weather forecasters that this "+
        "was going to happen this year, said "+
        "National Fire Information Officer Bob Valen. <br/>"+
        "But Friday also brought good news about a recently "+
        "devastating wildfire that threatened ancient "+
        "Indian ruins in a U.S. national park. " +
        "Mesa Verde National Park was reopening " +
        "Friday after firefighter contained a fire that "+
        "eventually scorched more than 22,950 acres and "+
        "came within four "+
        "miles of sandstone ruins built by the Anasazi Indians "+
        "hundreds of years ago.";

    BodyPager myBodyPager = new BodyPager(request);
    // Split the text into chunks.
    myBodyPager.autoSplitWML(strLongText);

    // Set the forward label to "Skip".
    myBodyPager.setTextLinkForward("Skip");
    // Set the exit label to "Done".
    myBodyPager.setTextLinkExit("Done");
    // Set the URL to go to when the user chooses to exit.
    myBodyPager.setURLLinkExit("http://www.openwave.com");

    dc.render(myBodyPager);
}
```

**Methods**

Method	Description
<code>BodyPager(HttpServletRequest request)</code>	Creates an instance of the <code>BodyPager</code> object.
<code>void addChunk(String textChunk)</code>	Adds a chunk of text to be displayed.
<code>void autoSplitWML(String bigText)</code>	Splits the given text into optimally sized chunks.
<code>void clearAll()</code>	Clears the array of text (chunks).
<code>void setCharSet(String charSet)</code>	Specifies the character set to be used for the WML code.
<code>void setChunkSize(chunkSize)</code>	Sets the size (in bytes) for each chunk of text. Default is 420 bytes.
<code>void setChunksPerPage(chunksPerPage)</code>	Sets the number of chunks to be displayed per page.
<code>void setFooterCard (Card newFooterCard)</code>	Sets a footer (text, input field, or image) that appears after the contents of each page generated by <code>BodyPager</code> .
<code>void setHeaderCard (Card newHeaderCard)</code>	Sets a header (text, input field, or image) that appears before the contents of each page generated by <code>BodyPager</code> .
<code>void setPostPercentageText (String postPercentageText)</code>	Sets the text that appears after the percentage (for example, 50% read).
<code>void setPrePercentageText (String prePercentageText)</code>	Sets the text that appears before the percentage (for example, less than 50% read).
<code>void setTextLinkExit(String textLinkExit)</code>	Sets the label for the secondary softkey used to exit from <code>BodyPager</code> .
<code>void setTextLinkForward(String text)</code>	Sets the label for the primary softkey that is used to link to the next page.
<code>void setTitle(String title)</code>	Sets the card title.
<code>void setURLLinkExit(String URLLinkExit)</code>	Sets the URL to open when the user exits <code>BodyPager</code> .
<code>void showPercentage()</code>	Set a flag ( <code>true</code> or <code>false</code> ) to show the percentage of text displayed. Default is <code>false</code> ; the percentage is not displayed.

## Button

You can use Buttons for navigation, although you should use them as little as possible. The alternative is to use menus and card paths. However, you need to implement a `Button` object to create buttons for the graphical Mobile Browser that acts independently of all other navigation patterns. You can also present button as an image.

Text-based browsers render the button as a link. The same is true for the graphical Mobile Browser when the extensions cannot be used.



**NOTE** OUI renders the primary path as a button on graphical Mobile Browsers in cards that contain a user interface widget. A single secondary path is also rendered as a button.

### Example

```
Button myButton = newButton("www.openwave.com", "OPWV");
myCard.addElement(myButton);
```

### Methods

Method	Description
<code>Button(String buttonURL, String label)</code>	Creates an instance of the <code>Button</code> object in which:  <code>buttonURL</code> is the URL to open when the button is chosen.  <code>label</code> is the text on the button (on a graphical Mobile Browser) or representing the link (on a text-based browser).
<code>void setButtonURL(String buttonURL)</code>	Sets the URL to open when the button is chosen.
<code>void setLabel(String label)</code>	Sets the label for the button. This method overwrites the label set in the constructor.
<code>void setUPGUIPic(String UPGUIPic)</code>	Sets the image shown on the button instead of the label (for the graphical Mobile Browser only).

## Caller

You can use the `Caller` object to write applications that make it possible for users to call a phone number that's included in a card, or to provide information about that number if the device doesn't support this feature.

Some handsets support the `mc` (make call) Wireless Telephony Applications Interface (WTAI) function, which allows users to initiate a telephone call from a list of contacts, an order form, or a phone number query. Not all browsers support the `mc` function. The graphical Openwave Mobile Browser supports it, the Nokia browser does not. The WML code for WTAI calls is:

```
<go href="wtai://wp/mc;00455551212"/>
```

The main problem with the `mc` function is trying to write applications for handsets that do not support WTAI. One solution is to create an extra card that contains information about which telephone number the user should dial. This is a poor solution compared to one-click phone calls, but it is the best option available to handsets without WTAI.

The `Caller` object comes in two flavors. With the first you can define some of the text that tells users what they are supposed to do. In this case, OUI builds an extra card behind the scenes and no further action is required from you.

```
Deck myDeck = new Deck();  
...  
  
Caller myCaller = new Caller(myDeck, "00455551212", "Call  
00455551212");
```

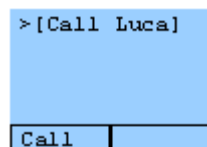
You need to pass the `Deck` object because the `Caller` object automatically creates a card for you. Passing the `Deck` object ensures that there is a place where OUI can place the card.

With the second flavor of the `Caller` object you have more control, but also more responsibility. You can use it to define the URL of a card or a deck to which non-WTAI handsets are directed. In this case, you must make sure that such a card or deck exists. However, because you are creating a fully fledged card, you can make it look exactly the way you want.

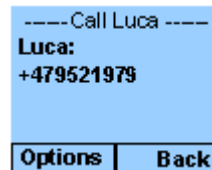
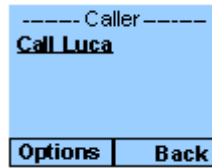
```
Caller myCaller = new Caller("00455551212", "#call");  
...  
Card myCard = new Card("call", "Luca", "Luca's number")  
  
myCard.addText("Luca's number: 00455551212");
```

The `Caller` object is an abstract task that can be used wherever a task can be used, such as inside a `doElement` or an `Anchor`, or associated to the different path activities of a card.

On the Openwave Mobile Browser, activating the link automatically initiates a telephone call to the specified number.



On the Nokia browser, the link needs to be to a new card that contains the number. Nokia users can use the Use Number feature of their phones.



### Methods

Method	Description
<code>Caller(Deck deck, String callNumber, String callText)</code>	Implicit mode: An additional card is added to the deck parameter for browsers that do not support WTAI. Creates an instance of the <code>Caller</code> object in which: <code>deck</code> is the deck where the task belongs. <code>callNumber</code> is the number to call. <code>callText</code> is the description of the number used by browsers that do not support WTAI.
<code>Caller(String callNumber, String callURL)</code>	Explicit mode: You must create and add a card to the deck for browsers that do not support WTAI. Creates an instance of the <code>Caller</code> object in which: <code>callNumber</code> is the number to call for browsers that support WTAI. <code>callURL</code> is the URL for the customized card for browsers that do not support WTAI.
<code>void setNumber(String callNumber)</code>	Sets the number to call. This method overrides the call number specified in the constructor.
<code>void setURL(String callURL)</code>	Sets the URL for the customized card for browsers that do not support WTAI.



## Card

The `Card` object is a direct counterpart of the WML `<card>` element. However, the `Card` object offers additional features that optimize usability across browsers and devices.

The `Card` object includes both standard methods and rendering directives, methods that dictate how a particular attribute or function should be rendered for a device.

### Card Title

Some devices do not render the `title` attribute. The `Card` object resolves this problem by providing a method to enforce the card title on all devices. If the device does not support the `title` attribute, the title is presented as the first line of text on the card. This enhances the usability of your application by describing context-sensitive information in a page.

### Navigation

There are three kinds of navigation: primary, secondary, and side paths. Primary paths are activities that 80% or more of users are likely to perform. Secondary paths are activities that many users (but not the majority) perform often. Side paths are activities that 80% of users will use 20% of the time when they use your application.

The `Card` object makes navigation as intuitive as possible. By using methods to add primary, secondary, and side paths, this object abstracts how navigation can be optimally implemented across different devices. For phones with Openwave browsers, optimal usability is achieved through softkeys, while phones from other vendors are better served with links.

If you prefer, you can also use the `Card` object to force all navigation with links.

### Redefining the `<prev>` Task

You can use the `Card` object to redefine the `<prev>` task by using the `enforceLogicalBack` method. This is useful for devices that do not redefine the `<prev>` task correctly. This method simply creates an event that is triggered when the user navigates to the card in a backward direction. The user is then redirected to a specific card or URL. The redefinition of the `<prev>` task enhances usability for Nokia browsers and maintains good usability on Openwave browsers.

### Enhancing the GUI

A widget is an element of a graphical user interface (GUI) that displays information or provides a specific way for the user to interact with an application. Widgets include the `Button`, `Check`, `Input`, `Picker`, `Popup`, and `Radio` objects.

If there is a widget in a card, the `Card` object optimizes the user interface on graphical Mobile Browsers by rendering the primary path as a button.

## Methods

Method	Description
<code>Card(String id, String shortTitle)</code>	Creates an instance of a card in which: <code>id</code> is the identifier for the card <code>shortTitle</code> is the title of the card. This is a direct counterpart of the WML <code>title</code> attribute.
<code>void addDoElement(DoElement doElement)</code>	Direct counterpart of the WML <code>&lt;do&gt;</code> element. This method adds a task associated with an element in the user interface.
<code>void addElement(WaomElement object)</code>	Appends an existing object to the flow of the card. Not all objects are legal; only such objects as menus, links, user interface widgets, and so on are legal. The <code>DoElement</code> , <code>Timer</code> , and <code>Onevent</code> objects have their own methods.
<code>void addOnevent(Onevent, onevent)</code>	Direct counterpart of the WML <code>&lt;onevent&gt;</code> element. This method adds an event to the card.
<code>void addSecondaryPath(String URL, String shortName)</code>	Adds another URL to be used as secondary navigation for the card. This is rendered as the URL for the secondary softkey on Openwave browsers and as a succeeding link on Nokia browsers.  The parameters are:  <code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code> .  <code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.  You can have more than one secondary path on a card. In this case, the user chooses which of the secondary URLs to go to. This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>addSecondaryPathTask</code> .

Method	Description
<pre>void addSecondaryPath(String URL, String shortName, String longName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. This is rendered as the URL for the secondary softkey on Openwave browsers and as a succeeding link on Nokia browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code> and <code>longName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML label attribute.</p> <p><code>longName</code> is the long label for the URL.</p> <p>When both <code>shortName</code> and <code>longName</code> are specified, Openwave browsers use the <code>shortName</code> as the label for the softkey while Nokia browsers use the <code>longName</code> as the label for the link.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary URLs to go to.</p> <p>This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass along some values to the server by means of postfields, you must use <code>addSecondaryPathTask</code>.</p>
<pre>void addSecondaryPathTask(AbsTask absTask, String shortName)</pre>	<p>Adds a secondary task to the card (see “Task” on page 75). This is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user selects the secondary softkey.</p> <p><code>shortName</code> is the short label for the task.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>

Method	Description
<pre>void addSecondaryPathTask(AbsTask absTask, String shortName, String longName)</pre>	<p>Adds a secondary task to the card (see “Task” on page 75). This is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user selects the secondary softkey.</p> <p><code>shortName</code> is the short label for the task.</p> <p><code>longName</code> is the long label for the task.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>
<pre>void addSidePath(String URL, String shortName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. However, this method causes navigation to be supported in the Options menu on Nokia browsers and through softkeys on Openwave browsers. The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the side paths to go to. This is the short form for defining side path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>addSidePathTask</code>.</p>

Method	Description
<pre>void addSidePath(String URL, String shortName, String longName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. However, this method causes navigation to be supported in the Options menu on Nokia browsers and through softkeys on Openwave browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the side paths to go to. This is the short form for defining a side path activity. It assumes that a URL is all that you need to move on. If you need to pass along some values to the server by means of postfields, you must use <code>addSidePathTask</code>.</p>
<pre>void addSidePathTask(AbsTask absTask, String shortName)</pre>	<p>Adds a side task to the card (see “Task” on page 75). However, this method causes navigation to be supported in the Options menu on Nokia browsers and through softkeys on Openwave browsers.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be executed as identified by <code>shortName</code>.</p> <p><code>shortName</code> is the short label for the task; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>

Method	Description
<pre>void addSidePathTask(AbsTask absTask, String shortName, String longName)</pre>	<p>Adds a side task to the card (see “Task” on page 75). However, this method causes navigation to be supported in the Options menu on Nokia browsers and through softkeys on Openwave browsers.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be executed as identified by <code>shortName</code> and <code>longName</code>.</p> <p><code>shortName</code> is the short label for the task.</p> <p><code>longName</code> is the long label for the task.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>
<pre>void addText(String text)</pre>	<p>Adds the text to be displayed.</p>
<pre>void beginParagraph()</pre>	<p>Sets the start of a paragraph. You must use <code>beginParagraph</code> at the beginning of each card. You can use it several times inside a card, but if you do you must close the previous paragraph explicitly with <code>endParagraph</code>.</p>
<pre>void beginParagraph(int align, int wrap)</pre>	<p>Sets the start of a paragraph in which:</p> <p><code>align</code> is 0 to left align the text, 1 to center the text, 2 to right align the text</p> <p><code>wrap</code> is 0 to not wrap the text (<code>nowrap</code>) and 1 to wrap the text to the next line of the display (<code>wrap</code>).</p> <p>You must use <code>beginParagraph</code> at the beginning of each card. You can use it several times inside a card, but if you do you must close the previous paragraph explicitly with <code>endParagraph</code>.</p>
<pre>void beginParagraph(String align, String mode)</pre>	<p>Sets the start of a paragraph in which:</p> <p><code>align</code> is the alignment of the text: <code>left</code>, <code>center</code>, or <code>right</code></p> <p><code>mode</code> is the <code>wrap</code> or <code>nowrap</code>, to wrap the text to the next line of the display or not.</p> <p>You must use <code>beginParagraph</code> at the beginning of each card. You can use it several times inside a card, but if you do you must close the previous paragraph explicitly with <code>endParagraph</code>.</p>

Method	Description
<code>void endParagraph()</code>	Sets the end of a paragraph. You must use <code>endParagraph</code> at least once at the end of each card.
<code>int getAccesskey()</code>	Returns a unique sequential number, useful for appending anchors or simple links to the flow of a card and for generating incremental values for the <code>accesskey</code> attribute.
<code>String getID()</code>	Returns the ID of the card.
<code>void setId(String id)</code>	Sets the ID of the card.
<code>void setNewContext(boolean newcontext)</code>	Direct counterpart of the WML <code>newcontext</code> attribute. This method sets whether or not to define a new context.
<code>void setOnenterbackward(String onenterbackwardURL)</code>	Direct counterpart of the WML <code>onenterbackward</code> attribute for the <code>&lt;card&gt;</code> element. This method sets the URL to open when the user navigates to the card in a backward direction.
<code>void setOnenterforward(String onenterforwardURL)</code>	Direct counterpart of the WML <code>onenterforward</code> attribute. This method sets the URL to open when the user navigates to the card in a forward direction.
<code>void setOntimer(String ontimerURL)</code>	Direct counterpart of the WML <code>ontimer</code> attribute. This method sets the URL to open when the timer expires.
<code>void setOrdered(boolean ordered)</code>	Direct counterpart of the WML <code>ordered</code> attribute for the <code>&lt;card&gt;</code> element. This method sets the <code>ordered</code> attribute to either <code>true</code> or <code>false</code> .

Method	Description
<pre>void setPrimaryPath(String URL, String shortName)</pre>	<p>Sets the URL to use for the primary navigation of the card. Rendered as the URL for the primary softkey on Openwave browsers and as the first link on Nokia browsers.</p> <p>The parameters are:</p> <p>URL is the URL to open when the user chooses the primary softkey (or the first link for Nokia browsers).</p> <p>shortName is the label for the URL.</p> <p>You can have only one primary path on a card. This is the short form for defining a main path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>setPrimaryPathTask</code>.</p>
<pre>void setPrimaryPath(String URL, String shortName, String longName)</pre>	<p>Sets the URL to use for the primary navigation of the card. Rendered as the URL for the primary softkey on Openwave browsers and as the first link on Nokia browsers.</p> <p>The parameters are:</p> <p>URL is the URL to open when the user chooses the primary softkey (or the first link for Nokia browsers).</p> <p>shortName is the short label for the URL.</p> <p>longName is the long label for the URL</p> <p>You can have only one primary path on a card. This is the short form for defining a main path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>setPrimaryPathTask</code>.</p>



Method	Description
<pre>void setPrimaryPathTask(AbsTask absTask, String shortName)</pre>	<p>Sets the primary task of the card (see “Task” on page 75). Rendered differently on Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the label for the task.</p> <p>You can have only one primary path on a card.</p>
<pre>void setPrimaryPathTask(AbsTask absTask, String shortName, String longName)</pre>	<p>Sets the primary task of the card (see “Task” on page 75). Rendered differently on Openwave and Nokia browsers, depending on the type of the task.</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the short label for the task.</p> <p><code>longName</code> is the long label for the task.</p> <p>You can have only one primary path on a card.</p>
<pre>void setTitle(String title)</pre>	<p>Direct counterpart of the WML <code>title</code> attribute. This method sets the title of the card.</p>
<pre>void setTaskMenu(TaskMenu taskMenu)</pre>	<p>Adds a <code>TaskMenu</code> object to the card.</p>
<pre>void setTimer(Timer timer)</pre>	<p>Adds a <code>Timer</code> object to the card.</p>

### Rendering Directives

Method	Description
<code>void enforceLogicalBack(String BackURL)</code>	Sets the URL to open when the user navigates to the card in a backward direction. This is useful for devices that don't properly redefine <code>&lt;prev /&gt;</code> . This renders as an event of <code>type="onenterbackward"</code> .
<code>void enforceNavigationWithLinks ()</code>	Ensures that primary and secondary paths are rendered as hyperlinks, regardless of the device family.
<code>void enforceTitle()</code>	For devices that don't support the <code>title</code> attribute for cards, inserts the title of the card as the first line of text in the card.
<code>void setEnforceTitle(boolean enforceTitle)</code>	Sets whether to enforce display of card titles for devices that don't support the <code>title</code> attribute.

## Check

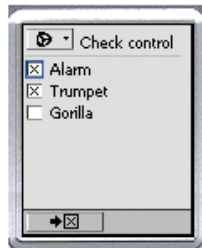
You can use the `Check` object to create checkboxes on graphical browsers. On text-based browsers checkboxes are rendered as multiple-selection lists.

### Example

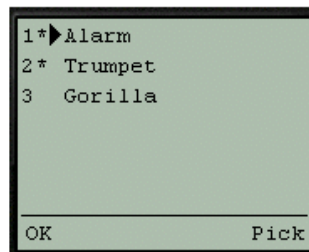
Check objects are rendered differently on different browsers:

```
Check myCheck = new Check("alarm");
myCheck.addEntry("on1", "Alarm");
myCheck.addEntry("on2", "Trumpet");
myCheck.addEntry("on3", "Gorilla");
myCheck.setIvalue("1;2");
```

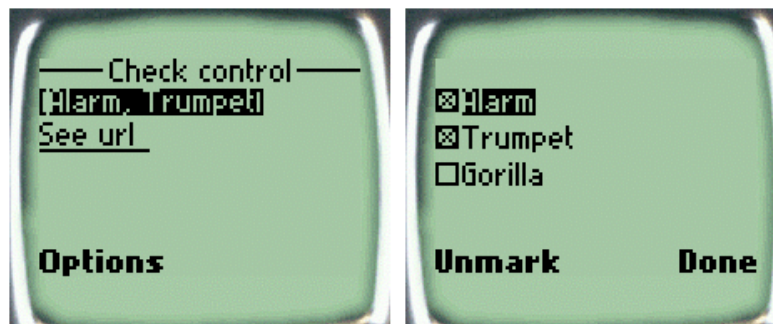
On graphical browsers:



On text-based Openwave browsers:



On Nokia browsers:



## Methods

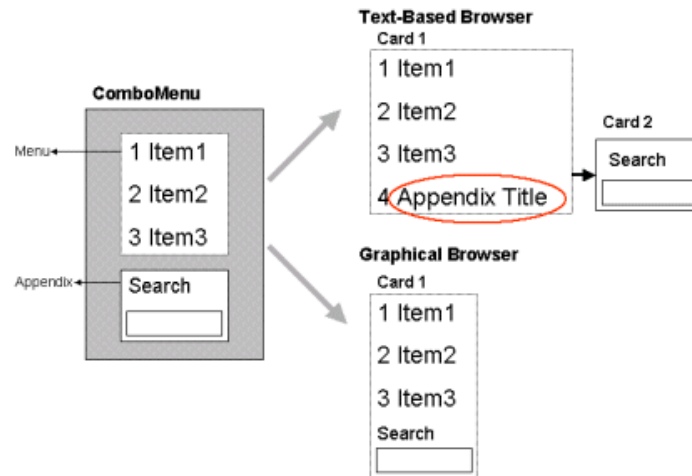
Method	Description
<code>Check(String name)</code>	Creates an instance of the <code>Check</code> object in which <code>name</code> is the identifier for the object. The <code>name</code> variable is the direct counterpart of the WML <code>name</code> attribute for the <code>&lt;select&gt;</code> element.
<code>void addEntry(String key, String value)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value to be contained in the <code>name</code> variable when the user chooses from the list of options.  <code>value</code> is the text describing the option.  If the user selects multiple options, the keys are concatenated by semicolons ( ; ).
<code>void addEntry(String key, String value, String onpick)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value contained in the <code>name</code> variable when the user chooses from the list of options  <code>value</code> is the text describing the option  <code>onpick</code> is the URL to open when the entry is selected.
<code>String getIname()</code>	Returns the name of the WML variable that contains the index value of the option the user selects.
<code>String getIvalue</code>	Returns the index of the default value or values.
<code>String getMultiple()</code>	Returns <code>true</code> if the user can make multiple selections; otherwise returns <code>false</code> .
<code>String getName()</code>	Returns the name of the WML variable that contains the index value of the option the user selects.
<code>String getValue()</code>	Returns the default value or values of the user selection.
<code>String isMultiple()</code>	Returns <code>true</code> if the user can make multiple selections; otherwise returns <code>false</code> .

Method	Description
<code>void setName(String name)</code>	Sets the name of the WML variable to contain the value of the option or options the user selects.
<code>void setValue(String value)</code>	Direct counterpart of the WML <code>value</code> attribute. This method sets the default selection in the list of options by specifying the value associated with the default selection.
<code>void setMultiple(String multiple)</code>	Direct counterpart of the WML <code>multiple</code> attribute. This method sets whether multiple selections can be made. This method accepts <code>true</code> or <code>false</code> .
<code>void setMultiple(boolean multiple)</code>	Direct counterpart of the WML <code>multiple</code> attribute. This method sets whether multiple selections can be made. This method accepts <code>true</code> or <code>false</code> .
<code>void setIvalue(String ivalue)</code>	Direct counterpart of the WML <code>ivalue</code> attribute. This method sets the default selection in the list of options by specifying the index (1, 2, and so on) of the default selection.
<code>void setIname(String iname)</code>	Direct counterpart of the WML <code>iname</code> attribute. This method sets the name of the variable that contains the index value of the option selected. The index value associated with each option comes from its position in the <code>&lt;select&gt;</code> list, starting with 1. If the user has not selected an option, the index value either is 0 or the <code>ivalue</code> .

## ComboMenu

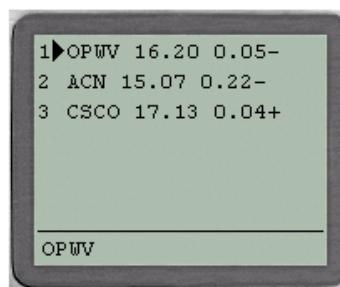
You can use a `ComboMenu` object to append a user interface object to a menu. The appended object is displayed at the end of the menu on the graphical Openwave Mobile Browser, and is implemented as an additional menu item linked to another card on text-based browsers.

For example, you may want to present the user with an advertisement or a search input field at the end of a menu.



If you don't use `ComboMenu`, the appended object is lost.

Text-Based Browser



Graphical Mobile Browser



The `ComboMenu` object consists of the `Menu` and the `Appendix` objects. (For more information, see “Appendix” on page 4 and “Menu” on page 53.)

**NOTE** Because of the nature of elective forms, on Nokia browsers the `ComboMenu` object can be rendered in one card.

### Example

```
Deck myDeck = new Deck();  
Menu myMenu = new Menu();  
...  
Appendix myAppendix = new Appendix("Other Info");  
...  
ComboMenu myCombo = new ComboMenu(myDeck, myMenu, myAppendix)
```

### Methods

Method	Description
<code>ComboMenu(Deck deck, Menu menu, Appendix appendix)</code>	Constructor: Creates an instance of the ComboMenu object.

## Deck

The `Deck` object is a high-level object that is useful as a container of multiple templates and cards.

Plain WML and WML with Openwave extensions have different Document Type Definitions (DTDs). Encapsulating a deck into an object ensures that the most appropriate DTD is delivered to the browser.

### Example

```
myDeck = new Deck()
```

### Methods

Method	Description
<code>Deck()</code>	Creates an instance of the <code>Deck</code> object.
<code>void addCard(AbsCard absCard)</code>	Adds a card to the deck. This is the direct counterpart of the WML <code>&lt;card&gt;</code> element.
<code>void addHead(Head head)</code>	Adds a <code>Head</code> object to the deck.
<code>void addTemplate(Template template)</code>	Adds a <code>Template</code> object to the deck.
<code>void disableNokiaBackNavigation()</code>	Disables defining <code>&lt;prev/&gt;</code> tasks in the deck template for Nokia browsers.
<code>void setCharSet(String charset)</code>	Specifies the character set to be used for the deck.



## DeviceContext

The `DeviceContext` object does not represent a WML construct. It encapsulates the `HTTP REQUEST` headers generated by the browser and often modified by the gateway, and makes that contextual information available to your applications. The headers include information about the device, the browser, the gateway, and related information.

The `DeviceContext` object accepts instructions, actively ensures that everything functions correctly, and provides you with the information that you need to make decisions.

**NOTE** The object model is completely independent of the `DeviceContext` object that does the rendering, so OUI can be implemented in any Java environment. If you need OUI to render an object model for a particular family of devices outside of the servlet framework, you can create a `DeviceContext` object based on a family name. The five family names are `UPText`, `UPGUI`, `Nokia`, `Generic`, and `MSIE5`.

### Example

```
DeviceContext dc = new DeviceContext(request, response);

Deck myDeck = new Deck();
...
    dc.disableExtensions();
...
    dc.render(myDeck);
```

### Methods

Method	Description
<code>DeviceContext(HttpServletRequest request, HttpServletResponse Response)</code>	<p>Creates an instance of the <code>DeviceContext</code> object in which:</p> <ul style="list-style-type: none"><li><code>request</code> is an <code>HTTP</code> request; it contains the <code>URL</code> of the document being requested and all information about the client such as browser type, <code>IP</code> address, <code>MIME</code> type, and so on.</li><li><code>response</code> is an <code>HTTP</code> response; it encapsulates the data being sent back to the client.</li></ul> <p>This method also identifies the <code>WAP</code> gateway, the handset and device, and the device model that requests the <code>URL</code>, and determines the rendering strategy based on the information from the <code>HTTP</code> request.</p>

Method	Description
<code>DeviceContext(String ctxName)</code>	Creates an instance of the <code>DeviceContext</code> object in which: <code>ctxName</code> identifies a particular handset context that you want to create. This method also determines the rendering strategy based on the context that you specified.
<code>void disableContentType()</code>	OUI automatically produces the correct MIME type for the markup it produces (WML and HTML). If you are an experienced developer, you may want to apply your own HTTP header manipulation, making sure that OUI does not interfere. You can use this method to disable automatic MIME type generation.
<code>void disableExtensions()</code>	OUI always tries to use extensions when rendering for the Openwave Mobile Browser. This method forces OUI to gracefully degrade the application to a version that does not deploy extensions.
<code>String getAgentFamilyName()</code>	Returns the browser family of the device: Openwave text-based, Openwave graphical, Nokia, MSIE, or Generic.
<code>String getAgentSubFamilyName()</code>	Returns the browser subfamily of the device: Openwave text-based, Alcatel, Openwave graphical, Nokia 9110, MSIE, or Generic.
<code>String getGatewayVendor()</code>	Returns the vendor of the WAP gateway used by the device.
<code>String getGatewayVersion()</code>	Returns the version of the WAP gateway used by the device.
<code>HandsetContext getHandsetContext()</code>	Returns the handset context of the device, describing the agent family and agent subfamily as well as the specific rendering strategy for the device.
<code>String getProxyVendor()</code>	Returns the vendor of the WAP gateway used by the device.
<code>String getProxyVersion()</code>	Returns the version of the WAP gateway used by the device.
<code>int getSize(WaomElement rootElement)</code>	Returns the length of the generated (string) static WML rendered from the given <code>rootElement</code> .

Method	Description
<code>String getUserAgentString()</code>	Returns the browser used by the device (also known as the user agent).
<code>int getVersionNumber()</code>	Returns the version number of the device model as an integer.
<code>String getVersionString()</code>	Returns the version number of the device model as a string.
<code>boolean isMAG()</code>	Returns <code>true</code> if the HTTP request passed through an Openwave Mobile Access Gateway (formerly called an UP.Link Server); otherwise returns <code>false</code> .
<code>boolean isUpLink()</code>	Returns <code>true</code> if the HTTP request passed through an Openwave Mobile Access Gateway, (formerly called an UP.Link Server); otherwise returns <code>false</code> .
<code>void optimizeForSpeed()</code>	Sets a flag to indicate that validation should be bypassed when rendering the static WML.
<code>void render()</code>	Generates the browser-specific HTML or WML code from the root element of the object. The result is sent back to the browser/device through an HTTP response.
<code>void render(WaomElement rootElement)</code>	Generates the browser-specific HTML or WML code from the given object. The result is sent back to the device through an HTTP response.
<code>String renderToString(WaomElement rootElement)</code>	Generates the browser-specific WML code from the given object. However, the static WML code is not sent back through an HTTP response. Instead, it is returned as a string.

## DoElement

The `DoElement` object represents WML `<do>` elements. This object is mainly used in templates.

`DoElement` objects can be attached to cards, but this is not the recommended way to use them. `DoElements` objects implement navigational elements effectively on some browsers, such as the Openwave Mobile Browser, but they have a secondary role on others, such as the Nokia browser.

For this reason, you should use the `addPrimaryPath`, `addSecondaryPath`, and `addSidePath` `Card` object methods (and their relatives, as described in “Card” on page 11), rather than using `DoElement` objects directly. Those methods maintain the best usability, regardless of the browser.

You should not use `DoElement` objects for primary activities (`type="accept"`), but only to provide common softkey mappings, such as Help, Info, and Menu.

### Example

```
Template myTemplate = new Template();

Task myTask = new Task("go", "main")

DoElement myDoElement = new DoElement(myTask, "options", "main");

myTemplate.addDoElement(myDoElement);
```

### Methods

Method	Description
<code>DoElement(AbsTask absTask, String type)</code>	Creates an instance of the <code>DoElement</code> in which:  <code>absTask</code> is the task to be performed when the <code>&lt;do&gt;</code> element has been activated.  <code>type</code> specifies a hint to the browser about the author's intended use of the element (such as <code>accept</code> , <code>prev</code> , <code>help</code> , <code>reset</code> , <code>options</code> , or <code>delete</code> ). This is a direct counterpart of the WML <code>type</code> attribute.

Method	Description
<code>DoElement(AbsTask absTask, String type, String label)</code>	<p>Creates an instance of the <code>DoElement</code> in which:</p> <p><code>absTask</code> is the task to be performed when the <code>&lt;do&gt;</code> element has been activated .</p> <p><code>type</code> specifies a hint to the browser about the author's intended use of the element (such as <code>accept</code>, <code>prev</code>, <code>help</code>, <code>reset</code>, <code>options</code>, or <code>delete</code>). This is a direct counterpart of the WML <code>type</code> attribute.</p> <p><code>label</code> specifies the text for dynamically labeling the task. To ensure compatibility on a wide range of devices, <code>label</code> should be a maximum of five characters. This is a direct counterpart of the WML <code>label</code> attribute.</p>
<code>DoElement(AbsTask absTask, String type, String label, String name)</code>	<p>Creates an instance of the <code>DoElement</code> in which:</p> <p><code>absTask</code> is the task to be performed when the <code>&lt;do&gt;</code> element has been activated.</p> <p><code>type</code> specifies a hint to the browser about the author's intended use of the element (such as <code>accept</code>, <code>prev</code>, <code>help</code>, <code>reset</code>, <code>options</code>, or <code>delete</code>). This is a direct counterpart of the WML <code>type</code> attribute.</p> <p><code>label</code> specifies the text for dynamically labeling the task. To ensure compatibility on a wide range of devices, <code>label</code> should be a maximum of five characters. This is a direct counterpart of the WML <code>label</code> attribute.</p> <p><code>name</code> specifies a name for the element. This is a direct counterpart of the WML <code>name</code> attribute.</p>
<code>void setLabel(String label)</code>	<p>Direct counterpart of the WML <code>label</code> attribute. This method specifies a textual string for labeling a user interface component. To ensure compatibility on a wide range of devices, <code>label</code> should be a maximum of five characters.</p>

<b>Method</b>	<b>Description</b>
<code>void setName(String name)</code>	Direct counterpart of the WML <code>name</code> attribute. This method specifies a name for the <code>DoElement</code> object.
<code>void setOptional(boolean optional)</code>	Direct counterpart of the WML <code>optional</code> attribute. If <code>true</code> is passed to this method, the browser can ignore the <code>&lt;do&gt;</code> element. By default, the browser does not ignore the <code>&lt;do&gt;</code> element.
<code>void setType(String type)</code>	<p>Direct counterpart of the WML <code>type</code> attribute. This method provides a hint to the browser about the author's intended use of the element (such as <code>accept</code>, <code>prev</code>, <code>help</code>, <code>reset</code>, <code>options</code>, or <code>delete</code>).</p> <p>The most commonly used <code>type</code> values are <code>accept</code> (a task mapped to the primary softkey) and <code>options</code> (mapped to the secondary softkey). Openwave browser software does not currently support the <code>delete</code>, <code>help</code>, and <code>prev</code> <code>type</code> values.</p>

## Form

The `Form` object is very similar to the `Card` object. However, the `Form` object relies on the OUI rendering strategy for optimal usability on current and future browsers.

The `Form` object allows you to focus on what information is needed from the user and not on the syntax of how to build the form. For example, you can rely on OUI to automatically generate paragraph tags when you add a text or an input field to the form.

By providing both `Card` and `Form` objects, OUI strikes a balance between flexibility in developing applications and support for future devices.

If your application requires users to enter a substantial amount of entry, it's better to use the `Form` object.

The `Form` object includes both standard methods and rendering directives, methods that dictate how a particular attribute or function should be rendered for a device.

## Methods

Method	Description
<code>Form(String id, String shortTitle)</code>	Creates an instance of a <code>Form</code> in which: <code>id</code> is the identifier for the card. <code>shortTitle</code> is the title of the card. This is a direct counterpart of the WML <code>title</code> attribute.
<code>void addElement(WaomElement element)</code>	Appends a OUI object to the flow of the card.
<code>void addSecondaryPath(String URL, String shortName)</code>	<p>Adds another URL to be used as secondary navigation for the card. This is rendered as the URL for the secondary softkey on Openwave browsers and as a succeeding link on Nokia browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML label attribute.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary URLs to go to. This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass along some values to the server by means of postfields, you must use <code>addSecondaryPathTask</code>.</p>



Method	Description
<pre>void addSecondaryPath(String URL, String shortName, String longName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. This is rendered as the URL for the secondary softkey on Openwave browsers and as a succeeding link on Nokia browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code> and <code>longName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p><code>longName</code> is the long label for the URL.</p> <p>When both <code>shortName</code> and <code>longName</code> are specified, Openwave browsers use the <code>shortName</code> as the label for the softkey and Nokia browsers use the <code>longName</code> as label for the link. You can have more than one secondary path on a card. In this case, the user chooses which of the secondary URLs to go to.</p> <p>This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>addSecondaryPathTask</code>.</p>
<pre>void addSecondaryPathTask(AbsTask absTask, String shortName)</pre>	<p>Adds a secondary task to the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the label for the task.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>

Method	Description
<pre>void addSecondaryPathTask(AbsTask absTask, String shortName, String longName)</pre>	<p>Adds a secondary task to the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the label for the task.</p> <p><code>longName</code> is the long label for the task.</p> <p>You can have more than one secondary path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>
<pre>void addSidePath(String URL, String shortName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. However, this method causes navigation to be supported in the Options menu on Nokia browsers and through softkeys on Openwave browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the side paths to go to. This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass along some values to the server by means of postfields, you must use <code>addSidePathTask</code>.</p>

Method	Description
<pre>void addSidePath(String URL, String shortName, String longName)</pre>	<p>Adds another URL to be used as secondary navigation for the card. However, this method causes navigation to be supported in the Options menu on Nokia browsers, and through softkeys on Openwave browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the option identified by <code>shortName</code> and <code>longName</code>.</p> <p><code>shortName</code> is the short label for the URL; this is the direct counterpart of the WML <code>label</code> attribute.</p> <p><code>longName</code> is the long label for the URL.</p> <p>When both <code>shortName</code> and <code>longName</code> are specified, Openwave browsers use the <code>shortName</code> as the label for the softkey and Nokia browsers use the <code>longName</code> as the label for the link.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the side paths to go to. This is the short form for defining a secondary path activity. It assumes that a URL is all that you need to move on. If you need to pass along some values to the server by means of post fields, you must use <code>addSidePathTask</code>.</p>
<pre>void addSidePathTask(AbsTask absTask, String shortName)</pre>	<p>Adds a secondary task to the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the label for the task.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>

Method	Description
<code>void addSidePathTask(AbsTask absTask, String shortName, String longName)</code>	<p>Adds a secondary task to the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the user chooses the primary softkey.</p> <p><code>shortName</code> is the short label for the task.</p> <p><code>longName</code> is the long label for the task.</p> <p>You can have more than one side path on a card. In this case, the user chooses which of the secondary tasks to execute.</p>
<code>void addText(String text)</code>	Adds the text to be displayed on the card.
<code>void beginParagraph()</code>	Sets the start of a paragraph.
<code>void beginParagraph(int align, int wrap)</code>	<p>Sets the start of a paragraph in which:</p> <p><code>align</code> is 0 to left align the text, 1 to center the text, 2 to right align the text.</p> <p><code>wrap</code> is 0 to not wrap the text (<code>nowrap</code>) and 1 to wrap the text to the next line of the display (<code>wrap</code>).</p>
<code>void beginParagraph(String align, String mode)</code>	<p>Sets the start of a paragraph in which:</p> <p><code>align</code> is the alignment of the text, <code>left</code>, <code>center</code>, or <code>right</code></p> <p><code>mode</code> is <code>wrap</code> or <code>nowrap</code>, to wrap the text to the next line of the display or not.</p>
<code>void endParagraph()</code>	Sets the end of a paragraph.
<code>int getAccskey()</code>	Returns a unique sequential number, useful for appending anchors or simple links to the flow of a card and for generating incremental values for the <code>accesskey</code> attribute.
<code>void setID(String id)</code>	<p>Sets the ID for the card.</p> <p>This method is the direct counterpart of the WML <code>id</code> attribute.</p>
<code>void setNewContext(boolean newcontext)</code>	<p>Sets whether or not to define a new context.</p> <p>This is a direct counterpart of the WML <code>newcontext</code> attribute.</p>

Method	Description
<code>void setOnenterbackward(String onenterbackwardURL)</code>	<p>Sets the URL to open when the user navigates to the card in a backward direction.</p> <p>This is a direct counterpart of the WML <code>onenterbackward</code> attribute for the <code>&lt;card&gt;</code> element.</p>
<code>void setOnenterforward(String onenterforwardURL)</code>	<p>Sets the URL to open when the user navigates to the card in a forward direction.</p> <p>This is a direct counterpart of the WML <code>onenterforward</code> attribute for the <code>&lt;card&gt;</code> element.</p>
<code>void setOntimer(String ontimerURL)</code>	<p>Sets the URL to open when the timer expires.</p> <p>This is a direct counterpart of the WML <code>ontimer</code> attribute for the <code>&lt;card&gt;</code> element.</p>
<code>void setOrdered(boolean ordered)</code>	<p>Sets the <code>ordered</code> attribute to either <code>true</code> or <code>false</code>.</p> <p>This is the direct counterpart of the WML <code>ordered</code> attribute for the <code>&lt;card&gt;</code> element.</p>
<code>void setPrimaryPath(String URL, String shortName)</code>	<p>Sets the URL to be used for the primary navigation of the card. This is rendered as the URL for the primary softkey on Openwave browsers and as the first link on Nokia browsers.</p> <p>The parameters are:</p> <p><code>URL</code> is the URL to open when the user chooses the primary softkey (or the first link for Nokia browsers).</p> <p><code>shortName</code> is the label for the URL</p> <p>You can have only one primary path on a card. This is the short form for defining a main path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>setPrimaryPathTask</code>.</p>

Method	Description
<pre>void setPrimaryPath(String URL, String shortName, String longName)</pre>	<p>Sets the URL to be used for the primary navigation of the card. This is rendered as the URL for the primary softkey on Openwave browsers and as the first link on Nokia browsers.</p> <p>The parameters are:</p> <p>URL is the URL to open when the user chooses the primary softkey (or the first link for Nokia browsers).</p> <p>shortName is the label for the URL</p> <p>longName is the long label for the URL.</p> <p>You can have only one primary path on a card. This is the short form for defining a main path activity. It assumes that a URL is all that you need to move on. If you need to pass values to the server by means of postfields, you must use <code>setPrimaryPathTask</code>.</p>
<pre>void setPrimaryPathTask(AbsTask absTask, String shortName)</pre>	<p>Sets the primary task of the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p>absTask is the task to be performed when the user chooses the primary softkey.</p> <p>shortName is the label for the task.</p> <p>You can have only one primary path on a card.</p>
<pre>void setPrimaryPathTask(AbsTask absTask, String shortName, String longName)</pre>	<p>Sets the primary task of the card (see “Task” on page 75). It is rendered differently for Openwave and Nokia browsers, depending on the type of the task.</p> <p>The parameters are:</p> <p>absTask is the task to be performed when the user chooses the primary softkey.</p> <p>shortName is the short label for the task.</p> <p>longName is the long label for the task.</p> <p>You can have only one primary path on a card.</p>

Method	Description
<code>void setTaskMenu(TaskMenu taskMenu)</code>	Adds a <code>TaskMenu</code> object to the card (see “TaskMenu” on page 77).
<code>void setTitle(String title)</code>	Sets the title of the card.

### Rendering Directives

Method	Description
<code>void enforceLogicalBack(String BackURL)</code>	Sets the URL to open when the user navigates to the card in a backward direction. This is useful for devices that don't properly redefine <code>&lt;prev /&gt;</code> . This renders as an event of <code>type="onenterbackward"</code> .
<code>void enforceNavigationWithLinks ()</code>	Ensures that primary and secondary paths are rendered as hyperlinks, regardless of the device family.
<code>void enforceTitle()</code>	For devices that don't support the <code>title</code> attribute for cards, inserts the title of the card as the first line of text in the card.
<code>void setEnforceTitle(boolean enforceTitle)</code>	Sets whether to enforce display of card titles for devices that don't support the <code>title</code> attribute.

## Head

The WML `<head>` element specifies information about the deck as a whole, including metadata and access control tags.

The `Head` object is a direct counterpart of the WML `<head>` element. This object is used to exploit advanced functionality of the Openwave platform, such as prefetch, bookmark control, and deck caching (time to live), without breaking the code for other browsers.

**IMPORTANT** Some of this functionality works only for the Openwave Mobile Browser when it connects via an Openwave Mobile Access Gateway, (formerly called the UP.Link Server). Some of the functionality works with the Openwave Mobile Browser regardless of the gateway, and some also works with Nokia phones.

### Example

```
Head myHead = new Head();

myHead.setAccessInfo("stockexchange.com", "customers/");

myHead.setBookmarkURL("http://wap.stockexchange.com/customer/Luca?quote=" + quote);

//cache control
myHead.disableCaching();

//TTL
myHead.setTTL(3600);

//mobile originated prefetch
myHead.setPrefetchURL("http://wap.stockexchange.com/customer/Luca?id=321")
myHead.setPrefetchURL("http://wap.stockexchange.com/customer/Luca?id=341")
```

### Methods

Method	Description
<code>Head()</code>	Creates an instance of the <code>Head</code> object.
<code>void addAccessInfo(String domain, String path)</code>	Direct counterpart of the WML <code>&lt;access&gt;</code> element. This method adds access information to the current deck. This information gives access only to decks originating from the specified domain and path. The <code>domain</code> parameter specifies the domain of other decks that can access cards in the current deck. The default value is the domain of the current deck. The <code>path</code> specifies the root URL of other decks that can access cards in the current deck. The default value is <code>/</code> (the root path of the current deck), which lets any deck within the specified domain access this deck.



Method	Description
<code>boolean addMetaTag(String metaTag)</code>	Direct counterpart of the WML <meta> element.
<code>void disableCaching()</code>	Specifies that the WML deck should not be cached or kept in memory.
<code>ArrayList getAccessInfo()</code>	Returns the domain and URL of the decks that have access to the deck.
<code>String getBookmarkURL()</code>	Returns the URL to be bookmarked, if the current deck is bookmarkable.
<code>boolean getBookmarkable()</code>	Returns <code>true</code> if the current deck is bookmarkable; otherwise returns <code>false</code> .
<code>ArrayList getMetaTags()</code>	Returns the name-value pairs specifying the properties associated with the deck.
<code>boolean getMustRevalidate</code>	Returns <code>true</code> if the Openwave browser must revalidate the TTL; otherwise returns <code>false</code> .
<code>String getPrefetchURL()</code>	Returns the URL to be preloaded to the cache when the deck is accessed.
<code>int getTTL()</code>	Returns the TTL for a deck, that is, the number of seconds that a device keeps the deck in cache memory.
<code>void setBookmarkURL(String bookmarkURL)</code>	Specifies the URL to be bookmarked, if the current deck is not bookmarkable. This feature is supported only on the Openwave Mobile Browser connecting via an Openwave MAG. It renders nothing on Nokia browsers.
<code>void setBookmarkable(boolean bookmarkable)</code>	Specifies whether or not the current deck is bookmarkable. The current deck is bookmarkable by default. This feature is supported only on the Openwave Mobile Browser connecting via an Openwave MAG. It renders nothing on Nokia browsers.
<code>void setMustRevalidate(boolean mustRevalidate)</code>	When passed with a <code>true</code> value, this method forces the Openwave Mobile Browser to revalidate the deck's TTL, even if the user navigates to the deck in the backward direction.
<code>void setPrefetchURL(String prefetchURL)</code>	Sets the URL to be preloaded to the cache when the current deck is accessed. This method is supported only by Openwave browsers.

Method	Description
<code>void setTTL(int TTL)</code>	Sets the TTL, that is, the length of time in seconds that a device keeps the deck in cache memory. This method accepts a value from an integer primitive data type.
<code>void setTTL(Integer TTL)</code>	Sets the TTL, that is, the length of time in seconds that a device keeps the deck in cache memory. This method accepts a value from an integer reference data type.
<code>void setTTL(String TTLStr)</code>	Sets the TTL, that is, the length of time in seconds that a device keeps the deck in cache memory. This method accepts an integer value passed as a string (for example, 360).

## HRule

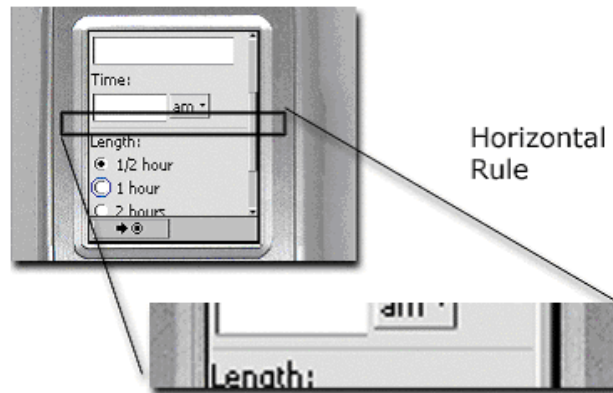
Horizontal rules are an extension to WML. You can use them on the graphical Mobile Browser to draw a horizontal line across the width the device display, to visually segregate elements on a card.

Because horizontal rules break old browsers, the `HRule` object ensures that the horizontal rule is rendered only on the graphical Mobile Browser.

### Example

The following line of code implements the `HRule` object to draw a horizontal line on the graphical Mobile Browser. The `size` and `width` attributes set the dimensions of the horizontal line.

```
HRule myHR = new HRule("20", "30");
```



**Methods**

Method	Description
<code>HRule()</code>	Creates an instance of the <code>HRule</code> object.
<code>HRule(String size, String width)</code>	Creates an instance of the <code>HRule</code> object in which: <code>size</code> is the height in pixels of the line to be drawn. <code>width</code> is the length in pixels of the line to be drawn.
<code>void setSize(String size)</code>	Sets the height in pixels of the line to be drawn. This method can accept only integer values.
<code>void setWidth(String width)</code>	Sets the horizontal length in pixels of the line to be drawn. The values can be either an absolute number of pixels or a percentage of the screen width. If the value of the <code>width</code> attribute exceeds the dimensions of the screen, the line is truncated at the edge of the screen.

## Image

Serving compatible images to different devices can be challenging. For example, Openwave graphical browsers support the PNG format, while Nokia browsers do not.

The `Image` object encapsulates the WML `<img>` element. You can use this object to specify different image types for different classes of devices. This means that you can produce multiple versions of an image and serve the image in the appropriate type to the requesting device. For example, you can serve an image in WBMP format to Nokia and text-based browsers, while serving the same image in PNG format to the Openwave graphical browser.

If one of the main methods to override the default image is not applied, all devices receive the image specified by the constructor.

### Example

```
Image myImage = new Image("UK Compass", "http://www.openwave.com/images/logo.wbmp");
myImage.addUPGUIPic("http://www.openwave.com/images/logo.png");
```

### Methods

Method	Description
<code>Image(String altText, String GenericPicURL)</code>	Creates an instance of the <code>Image</code> object in which:  <code>altText</code> is alternate text to display if the device does not support images or cannot find the specified image.  <code>GenericPicURL</code> is the URL of the image to be displayed unless overridden by one of the member methods.  Unless overridden by one of the other methods, the <code>Image</code> object renders the same <code>&lt;img&gt;</code> tag for each device.
<code>void setAlign(String align)</code>	Sets the alignment of the image (center, left, right).
<code>void setAltText(String altText)</code>	Sets the alternate text to be displayed if the device does not support images or cannot find the specified image.
<code>void setGenericPic(String GenericPicURL)</code>	Sets the URL of the image to be displayed unless overridden by one of the member methods.
<code>void setMSIEPic(String imageURL)</code>	Sets the URL of the image to be displayed for Microsoft Internet Explorer instead of the <code>GenericPicURL</code> .
<code>void setNokiaPic(String imageURL)</code>	Sets the URL of the image to be displayed for Nokia browsers instead of the <code>GenericPicURL</code> .

Method	Description
<code>void setUPGUIPic(String imageURL)</code>	Sets the URL of the image to be displayed for graphical Mobile Browsers instead of the <code>GenericPicURL</code> .
<code>void setUPTextPic(String imageURL)</code>	Sets the URL of the image to be displayed for Openwave text-based browsers instead of the <code>GenericPicURL</code> .
<code>void setHeight(int height)</code>	Sets the height of the image in the display. This method accepts an integer primitive data type. If you specify a height value that is not the same as the actual height of the image, the browser attempts to scale the image to fit the size that you specified. This attribute is currently not supported by Openwave browsers.
<code>void setHeight(String height)</code>	Sets the height of the image in the display. This method accepts an integer parameter passed as a string (that is, 1, 2, and so on). If you specify a height value that is not the same as the actual height of the image, the browser attempts to scale the image to fit the size that you specified. This attribute is currently not supported by Openwave browsers.
<code>void setWidth(int width)</code>	Sets the width of the image in the display. This method accepts an integer primitive data type. If you specify a width value that is not the same as the actual width of the image, the browser attempts to scale the image to fit the size that you specified. This attribute is currently not supported by Openwave browsers.
<code>void setWidth(String width)</code>	Sets the width of the image in the display. This method accepts an integer parameter passed as a string (that is, 1, 2, and so on). If you specify a width value that is not the same as the actual width of the image, the browser attempts to scale the image to fit the size that you specified. This attribute is currently not supported by Openwave browsers.

Method	Description
<code>void setHspace(int hspace)</code>	Sets the amount of space to the left and right of the image where nothing else in the display may encroach. This is useful for creating a border to the left and to the right of the image to keep text away from images. The default setting is 0.
<code>void setVspace(int vspace)</code>	Sets the amount of space at the top and bottom of the image where nothing else in the display may encroach. This is useful for creating a border on top and at the bottom of the image to keep text away from images.
<code>void setLocalsrc(String localsrc)</code>	Direct counterpart of the WML <code>localsrc</code> attribute. This method sets the value of the <code>localsrc</code> attribute. It accepts the name of a known icon to be displayed instead of the <code>GenericPicURL</code> . If the device cannot find the icon in ROM, it attempts to retrieve it from the Openwave MAG. If you specify a valid icon, the device ignores the <code>GenericPicURL</code> and <code>altText</code> (see constructor above), even though they are still required.

## Input

The `Input` object is the direct counterpart of the WML `<input>` element. In addition to the features of its counterpart, you can use the `Input` object to specify different input formats, or masks, for particular browsers, to ensure that users receive adequate feedback.

**NOTE** The `Input` object includes methods that you can use to hide, or mask, users' input in fields on screen, typically by replacing the characters that users enter with other characters, such as asterisks. This is a common technique to hide sensitive information, such as passwords. However, the masked information is not encrypted, so you should not rely on this feature for security.

### Example

```
Input myInput = newInput("text", "nickname", "");
:
myCard.addText("Name?");
myCard.addElement(myInput);
```

### Methods

Method	Description
<code>Input(String type, String name, String value)</code>	Creates an instance of the <code>Input</code> object in which:  <code>type</code> can be <code>text</code> (the default), which allows users to see the characters they enter, or <code>password</code> , which causes the browser to mask the characters that users enter by replacing them with other characters, such as asterisks.  <code>name</code> is the variable associated with the input field where the device stores the text entered by the user.  <code>value</code> is the initial value of the input field.
<code>String getTitle()</code>	Returns the title of the input field.
<code>boolean getEmptyok()</code>	Returns <code>true</code> if the input field can be left blank; otherwise, returns <code>false</code> .
<code>String getFormat()</code>	Returns the format of the data that the user entry must match.
<code>String getName()</code>	Returns the name of the WML variable in which the device stores the text that the user enters.
<code>String getMaxLength()</code>	Returns the maximum number of characters that the user can enter.
<code>String getSize()</code>	Returns the size of the input field as shown in the display.



Method	Description
<code>String getValue()</code>	Returns the text the user entered. If the user did not enter any text, returns the value of the variable named by the WML <code>name</code> attribute.
<code>void setEmptyok(boolean emptyok)</code>	Direct counterpart of the WML <code>emptyok</code> attribute. This method sets the <code>emptyok</code> attribute to either <code>true</code> or <code>false</code> . If <code>empty="true"</code> , the input field can be left blank.
<code>void setFormat(String format)</code>	Direct counterpart of the WML <code>format</code> attribute. This method sets the input field format, or the mask, to specify exactly what characters are allowed in an input string, and in what position.
<code>void setMaxlength(String maxlength)</code>	Direct counterpart of the WML <code>maxlength</code> attribute. This method specifies the maximum number of characters that users can enter in the field.
<code>void setName(String name)</code>	Direct counterpart of the WML <code>name</code> attribute. This method specifies the variable name to be associated with the input field. This variable name is used to store the input from the user.
<code>void setSize(String size)</code>	Direct counterpart of the WML <code>size</code> attribute. This method specifies the size of the input field.  If the <code>size</code> attribute is not specified, the field grows to accommodate all of the characters that the user enters.  If a value is given to the <code>size</code> attribute, when the user enters more characters than can be displayed in the field, the characters scroll off the screen to the left. When the user navigates off of the <code>input</code> element, only the beginning of the input is visible in the field.
<code>void setTitle(String title)</code>	Direct counterpart of the WML <code>title</code> attribute. This method specifies the title of the input field. Depending on the browser, this title may or may not be displayed.

Method	Description
<code>void setType(String type)</code>	<p>Direct counterpart of the WML <code>type</code> attribute. This method specifies the type of input field (<code>text</code> or <code>password</code>). Specifying <code>text</code> allows users to see the characters they enter. Specifying <code>password</code> causes the browser to mask the characters users that enter by replacing them with other characters, such as asterisks.</p>
<code>void setValue(String Value)</code>	<p>Direct counterpart of the WML <code>value</code> attribute. This method specifies the default value of the input field.</p> <p>If the input field is displayed and the variable named in the <code>name</code> attribute is not set, the <code>name</code> variable is assigned the value specified in the <code>value</code> attribute.</p> <p>If the <code>name</code> variable already contains a value, the <code>value</code> attribute is ignored.</p> <p>If the <code>value</code> attribute specifies a value that does not conform to the input mask specified by the <code>format</code> attribute, the browser ignores the <code>value</code> attribute.</p>

## Menu

Menu navigation is implemented with different constructs for different devices or class of devices. The Openwave Mobile Browser works best with the `<select>/<option>` construct, while other browsers work better with variations of the classical list of links.

You can use the `Menu` object to build menus that OUI renders using the construct that is best suited to each browser.

### Example

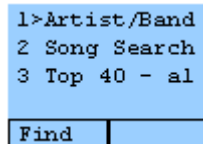
The following code builds a menu using the `Menu` object:

```
myMenu = new Menu();

//Enrich menu.
myMenu.addEntry("#band", "find", "Artist/Band search");
myMenu.addEntry("#song", "songs", "Song search");
myMenu.addEntry("#top", "top", "Top 40 - all genres");

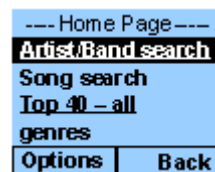
//Deploy menu.
myCard.addElement(myMenu);
```

This is how the menu is rendered on the Openwave Mobile Browser.



```
1>Artist/Band
2 Song Search
3 Top 40 - al
Find
```

This is how the menu is rendered on the Nokia browser.



```
--- Home Page ---
Artist/Band search
Song search
Top 40 - all
genres
Options | Back
```

You can also add a task to the menu:

```
Menu myMenu = new Menu();

//Enrich menu.
myMenu.addEntry("#band", "find", "Artist/Band search");
myMenu.addEntry("#song", "songs", "Song search", "smiley");

myTask = new Caller(myDeck, "00455551234", "CallRadio 1");

myMenu.addTaskEntry(myTask, "call", "Call Radio 1");
```

## Methods

Method	Description
<code>Menu()</code>	Creates an instance of the Menu object.
<code>void addEntry(String url, String title, String text)</code>	<p>Adds a menu item without an icon in which:</p> <p><code>url</code> is the URL to open when this menu item is chosen</p> <p><code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer.</p> <p><code>text</code> is the device displays this text to represent the menu item.</p>
<code>void addEntry(String url, String title, String text, String localIcon)</code>	<p>Adds a menu item with an icon in which:</p> <p><code>url</code> is the URL to open when this menu item is chosen.</p> <p><code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, labels should be five characters, or fewer.</p> <p><code>text</code> is the device displays this text to represent the menu item.</p> <p><code>localIcon</code> is the name or number identifying an icon to be displayed in front of the menu item.</p>
<code>void addTaskEntry(AbsTask absTask, String title, String text)</code>	<p>Adds a menu item without an icon, but navigation implies triggering a task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the selection is chosen.</p> <p><code>title</code> is a label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, labels should be five characters, or fewer.</p> <p><code>text</code> is the device displays this text to represent the menu item.</p>

Method	Description
<pre>void addTaskEntry(AbsTask absTask, String title, String text, String localIcon)</pre>	<p>Adds a menu item with an icon, but navigation implies triggering a task.</p> <p>The parameters are:</p> <p><code>absTask</code> is the task to be performed when the selection is chosen.</p> <p><code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, labels should be five characters, or fewer.</p> <p><code>text</code> is the device displays this text to represent the menu item.</p> <p><code>localIcon</code> is the name or number identifying an icon to be displayed in front of the menu item.</p>

## Onevent

The `Onevent` object encapsulates information about the WML `<onevent>` element. This element is used inside templates and cards.

### Example

```
Template myTemplate = new Template();

Task myTask = new Task("go", "main")

Onevent myOnevent = new Onevent(myTask, "onenterbackward");

myTemplate.addOnevent(myOnevent);
```

### Methods

Method	Description
<code>Onevent(AbsTask absTask, String type)</code>	Creates an instance of the <code>Onevent</code> object in which: <code>absTask</code> is the task to be performed when the event is triggered. <code>type</code> is the type of event ( <code>onpick</code> , <code>onenterforward</code> , <code>onenterbackward</code> , or <code>ontimer</code> ).
<code>void setType(String type)</code>	Direct counterpart of the WML <code>type</code> attribute. This method sets the type of event associated with the task.

## Picker

The `Picker` object encapsulates the WML `<select>` or `<option>` element, for use with the `Form` object without the need for enclosing paragraph tags, and to provide compatibility with future devices. You use the `Picker` object to offer users an interface for selecting one option from a finite set of possible options. Support for this object is similar in all devices.

### Example

```
Picker myPicker = new Picker("animal");

//Enrich picker.
myPicker.addEntry("D", "Dog");
myPicker.addEntry("C", "Cat");
myPicker.addEntry("H", "Horse");
myCard.addElement(myPicker);
```

### Methods

Method	Description
<code>Picker(String name)</code>	Creates an instance of the <code>Picker</code> object in which: <code>name</code> is the value of the name attribute and, ultimately, the name of the corresponding WML variable.
<code>void addEntry(String value, String text[, String onpickURL])</code>	Adds <code>Picker</code> entry in which: <code>value</code> specifies the value to assign to the variable defined in the <code>&lt;select&gt;</code> element name attribute if the user picks the entry. <code>text</code> is the device displays this text to represent the entry. <code>onpickURL</code> is the URL to open when the user picks from the entry.
<code>String getIname()</code>	Returns the name of the WML variable that contains the index value of the selected entry.
<code>String getIvalue()</code>	Returns the index of the default value of the selections.
<code>String getMultiple()</code>	Returns <code>true</code> if the user can select multiple options; otherwise returns <code>false</code> .
<code>String getName()</code>	Returns the name of the WML variable that returns the value of the selection.
<code>String getValue()</code>	Returns the default value or values of the selection.

Method	Description
<code>boolean isMultiple()</code>	Returns <code>true</code> if the user can pick more than one option; otherwise, returns <code>false</code> .
<code>void setName(String iname)</code>	Direct counterpart of the WML <code>iname</code> attribute. This method sets the name of the variable that contains the index value of the selected option.  The index value associated with each option comes from its position in the <code>&lt;select&gt;</code> list, starting with 1. If the user has not selected an option, the index value is either 0 or the <code>ivalue</code> .
<code>void setIvalue(String ivalue)</code>	Direct counterpart of the WML <code>ivalue</code> attribute. This method sets the default selection in the list of options by specifying the index (1, 2, and so on) of the default selection.
<code>void setMultiple(String multiple)</code>	Direct counterpart of the WML <code>multiple</code> attribute. This method sets whether the user can select more than one option from the menu. This method accepts <code>true</code> or <code>false</code> .
<code>void setMultiple(boolean multiple)</code>	Direct counterpart of the WML <code>multiple</code> attribute. This method sets whether the user can select more than one option from the menu. This method accepts <code>true</code> or <code>false</code> .
<code>void setName(String name)</code>	Direct counterpart of the WML <code>name</code> attribute. This method sets the name of the corresponding WML variable to contain the value of the selection.
<code>void setTitle(String title)</code>	Direct counterpart of the WML <code>title</code> attribute. This method sets the label that identifies the option. The Openwave text-based browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer.
<code>void setValue(String value)</code>	Direct counterpart of the WML <code>value</code> attribute.



## PickerCard

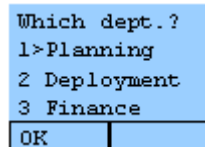
There are cases in which users should be able to set a variable with a single click and then move on. Although this is similar to the functionality offered by `Menu` and `Picker`, those two objects don't maximize usability across browsers for this specific case. The `PickerCard` object offers this specific functionality.

The `PickerCard` object includes both standard methods and rendering directives, methods that dictate how a particular attribute or function should be rendered for a device.

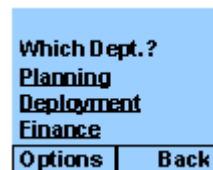
### Example

Suppose that your users have to choose a company department and that you want to store the choice in the WML variable `$dept`.

This is how the list is rendered on the Openwave Mobile Browser.



This is how the list is rendered on the Nokia browser.



Extending the `Menu` object to handle the case in which the programmer wants to set a WML variable would lead to a bloated API. You use the `PickerCard` object instead:

```
myPickerCard = new
PickerCard("picker", "dept", "#nextStage", "next", "Which Dept.?");

myPickerCard.addEntry("plan", "Planning");
myPickerCard.addEntry("deply", "Deployment");
myPickerCard.addEntry("cash", "Finance");

myDeck.addCard(myPickerCard);
```

## Methods

Method	Description
<code>PickerCard(String id, String varName, String nextCardURL, String softkeyLabel, String text)</code>	<p>Creates an instance of the <code>PickerCard</code> object in which:</p> <p><code>id</code> is the identifier for the card.</p> <p><code>varName</code> represents the value of the name attribute and, ultimately, the name of the corresponding WML variable.</p> <p><code>softkeyLabel</code> is the label that appears in the primary softkey on Openwave text-based browsers. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer. Devices that don't support dynamic labeling ignore the label attribute.</p> <p><code>nextCardURL</code> is the URL to open when the user picks from one of the selections.</p> <p><code>text</code> is the device displays this text before the selections.</p>
<code>void addEntry(String value, String text)</code>	<p>Adds a <code>Picker</code> entry in which:</p> <p><code>value</code> specifies the value to assign to the variable defined in the <code>&lt;select&gt;</code> element name attribute if the user selects the option.</p> <p><code>text</code> is the device displays this text to represent the selection item.</p>
<code>void addEntry(String value, String text, String onpick, String title)</code>	<p>Adds a <code>Picker</code> entry in which:</p> <p><code>value</code> specifies the value to assign to the variable defined in the <code>&lt;select&gt;</code> element name attribute if the user selects the option.</p> <p><code>text</code> is the device displays this text to represent the selection item.</p> <p><code>onpick</code> is the URL to open when the user picks a selection. This URL overrides the <code>nextCardURL</code> specified in the constructor.</p> <p><code>title</code> is the label that dynamically appears in the primary softkey for this entry. This parameter overrides the <code>softkeyLabel</code> specified in the constructor.</p>
<code>String getId()</code>	Returns the ID of the card.

Method	Description
<code>String getText()</code>	Returns the text that the device displays in front of the selection.
<code>String getNextCardURL()</code>	Returns the URL to open when the user selects an option.
<code>String getVarName()</code>	Returns the WML variable name associated with the option that the user selects.
<code>void setID(String id)</code>	Sets the ID of the card.
<code>void setNextCardURL(String nextCardURL)</code>	Sets the URL to open when the user picks a selection. This URL is overridden if you specified an <code>onpick</code> URL in the <code>addEntry</code> method.
<code>void setSoftkeyLabel(String softkeyLabel)</code>	Sets a label that appears in the primary softkey. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer. Devices ignore the <code>label</code> attribute if they do not support dynamic labeling. However, this label is overridden if you specify a title in the <code>addEntry</code> method.
<code>void setText(String text)</code>	Sets the text that the device displays in front of the selection.
<code>void setVarName(String varName)</code>	Sets the variable name to receive the value when the user picks from the selection.

## Rendering Directives

Method	Description
<code>void enforceLogicalBack(String BackURL)</code>	Sets the URL to open when the user navigates to the card in a backward direction. This method is useful for devices that don't properly redefine <code>&lt;prev /&gt;</code> . This method renders as an event of <code>type="onenterbackward"</code> .
<code>void enforceNavigationWithLinks()</code>	Ensures that primary and secondary paths are rendered as hyperlinks, regardless of the device family.
<code>void enforceTitle()</code>	For devices that don't support the <code>title</code> attribute for cards, inserts the title of the card as the first line of text in the card.
<code>void setEnforceTitle(boolean enforceTitle)</code>	Sets whether to enforce display of card titles for devices that don't support the <code>title</code> attribute.

## Popup

The graphical Openwave Mobile Browser supports pop-up menus, which you can create with the `Popup` object. Pop-up menus present users with a convenient list of options, rather than a field in which they must enter text. On text-based browsers, a pop-up menu created with the `Popup` object is rendered a menu.

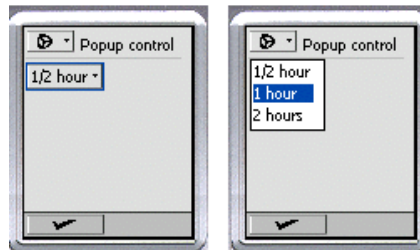
### Example

The following example uses the `Popup` object to create a pop-up menu:

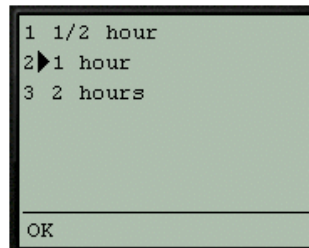
```

Popup myPopup = new Popup("popup");
myPopup.addEntry("on1", "1/2 hour");
myPopup.addEntry("on2", "1 hour");
myPopup.addEntry("on3", "2 hours");
  
```

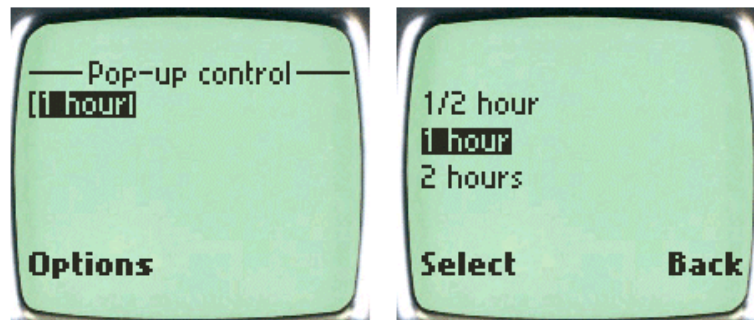
On the graphical Openwave Mobile Browser, the code is rendered as a pop-up menu.



On the text-based Openwave Mobile Browser, the code is rendered as a list of options.



On the Nokia browser, the code is rendered as follows.



## Methods

Method	Description
<code>Popup(String name)</code>	Creates an instance of the <code>Popup</code> object in which:  <code>name</code> is the name of the variable in which the device stores the value of a corresponding selection. This also becomes the value of the WML <code>name</code> attribute.
<code>void addEntry(String key, String value)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value to be contained in the <code>name</code> variable when the user picks from the list of options.  <code>value</code> is the text describing the option.
<code>void addEntry(String key, String value, String onpick)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value to be contained in the <code>name</code> variable when the user picks from the list of options.  <code>value</code> is the text describing the option.  <code>onpick</code> is the URL to open when the user selects the option.
<code>String getIname()</code>	Returns the name of the WML variable that contains the index value of the selected option.
<code>String getIvalue()</code>	Returns the index of the default value of the selections.
<code>String getName()</code>	Returns the name of the WML variable that returns the value of the selection.
<code>String getValue()</code>	Returns the default value or values of the selection.
<code>void setIname(String iname)</code>	Direct counterpart of the WML <code>iname</code> attribute. This method sets the name of the variable that contains the index value of the selected option.  The index value associated with each option comes from its position in the <code>&lt;select&gt;</code> list, starting with 1. If the user has not selected an option, the index value is either 0 or the <code>ivalue</code> .

Method	Description
<code>void setIvalue(String ivalue)</code>	Direct counterpart of the WML <code>ivalue</code> attribute. This method sets the default selection in the list of options by specifying the index (1, 2, and so on) of the default selection.
<code>void setName(String name)</code>	Direct counterpart of the WML <code>name</code> attribute. This method sets the name of the corresponding WML variable to contain the value of the selection.
<code>void setValue(String value)</code>	Direct counterpart of the WML <code>value</code> attribute. This method sets the default selection in the list of options by specifying the value associated with the default selection. If the <code>name</code> attribute already has a value when the user navigates to the <code>&lt;select&gt;</code> element, the <code>value</code> attribute is ignored.

## Radio

You can use the `Radio` object to add radio buttons to your wireless applications for the graphical Openwave Mobile Browser. On text-based browsers, the `Radio` object renders radio buttons with the `<select>` and `<option>` WML elements.

### Example

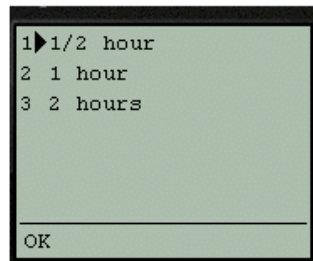
The following example uses the `Radio` object to create a set of radio buttons:

```
Radio myRadio = new Radio("radiovar");  
myRadio.addEntry("on1", "1/2 hour");  
myRadio.addEntry("on2", "1 hour");  
myRadio.addEntry("on3", "2 hours");
```

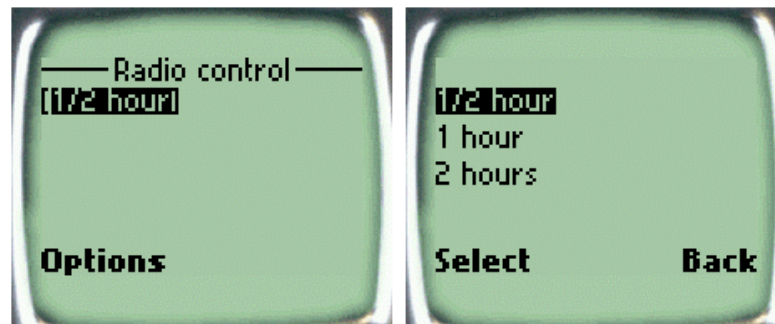
On the graphical Openwave Mobile Browser, the code is rendered like this.



On the text-based Openwave Mobile Browser, the code is rendered like this.



On the Nokia browser, the code is rendered like this.





## Methods

Method	Description
<code>Radio(String name)</code>	Creates an instance of the <code>Popup</code> object in which:  <code>name</code> is the name of the variable in which the device stores the value of a corresponding selection. This also becomes the value of the WML <code>name</code> attribute.
<code>void addEntry(String key, String value)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value to be contained in the <code>name</code> variable when the user picks from the list of options.  <code>value</code> is the text describing the option.
<code>void addEntry(String key, String value, String onpick)</code>	Adds an entry to the list of options in which:  <code>key</code> is the value to be contained in the <code>name</code> variable when the user picks from the list of options.  <code>value</code> is the text describing the option.  <code>onpick</code> is the URL to open when the entry is selected.
<code>String getIname()</code>	Returns the name of the WML variable that contains the index value of the selected option.
<code>String getIvalue()</code>	Returns the index of the default value of the selections.
<code>String getName()</code>	Returns the name of the WML variable that will return the value of the selection.
<code>String getValue()</code>	Returns the default value or values of the selection.
<code>void setIname(String iname)</code>	Direct counterpart of the WML <code>iname</code> attribute. This method sets the name of the variable that contains the index value of the option selected.  The index value associated with each option comes from its position in the <code>&lt;select&gt;</code> list, starting with 1. If the user has not selected an option, the index value is 0 or the <code>ivalue</code> .

<b>Method</b>	<b>Description</b>
<code>void setValue(String ivalue)</code>	Direct counterpart of the WML <code>ivalue</code> attribute. This method sets the default selection in the list of options by specifying the index (1, 2, and so on) of the default selection.
<code>void setName(String value)</code>	Direct counterpart of the WML <code>name</code> attribute. This method sets the name of the corresponding WML variable to contain the value of the selection.
<code>void setValue(String value)</code>	Direct counterpart of the WML <code>value</code> attribute. This method sets the default selection in the list of options by specifying the value associated with the default selection.

## SimpleLink

The `SimpleLink` object is a simple `<a href="url"></a>` link. It is encapsulated in an object to support the `accesskey` attribute, which is an Openwave extension to WML 1.2 that you can use to associate a number between 0 and 9 with the link. This is a keyboard accelerator for power users, who can trigger the link by pressing the corresponding number key on the device keypad.

### Example

```
SimpleLink mySimpleLink = new SimpleLink("band.wml", "the band");  
  
mySimpleLink.setTitle("band");  
mySimpleLink.setAccesskey(1);
```

## Methods

Method	Description
<code>SimpleLink(String url, String text)</code>	Creates an instance of the <code>SimpleLink</code> object in which: <code>url</code> is the URL to open when the link is chosen. <code>text</code> is what the device displays to represent the link.
<code>void setAccesskey(int accesskey)</code>	Sets the <code>accesskey</code> attribute for devices that support it. The number that you specify through this method appears to the left of the link. Pressing the corresponding key on a device that supports this attribute results in immediate navigation. This method requires a primitive integer parameter.
<code>void setAccesskey(String accesskey)</code>	Sets the <code>accesskey</code> attribute for devices that support it. The number that you specify through this method appears to the left of the link. Pressing the corresponding key on a device that supports this attribute results in immediate navigation. This method requires an integer parameter passed as a string.
<code>void setText(String text)</code>	Sets the text that the device displays to represent the link.
<code>void setTitle(String title)</code>	A direct counterpart of the WML <code>title</code> attribute. Sets the label that appears on the primary softkey that is associated with the link.
<code>void setURL(String simpleURL)</code>	A direct counterpart of the WML <code>href</code> attribute of the <code>&lt;a&gt;</code> element. Sets the URL to open when the link is chosen.

## Table

You can use the `Table` object to build tables for devices that support them, and whose contents are preserved when rendered on devices that don't support tables.

Although tables are part of WML 1.1, some browsers, such as the Nokia browser, do not support tables and simply ignore table tags (`table`, `tr`, and `td`). Because this makes the content in tables meaningless on some browsers, many developers don't use tables.

The `Table` object offers three ways of interpreting the meaning of information laid out in a table:

- Row logic: The information displayed in a table makes sense if interpreted on a row basis. In row logic, rows are the subparts of a table that need to be preserved, in the sense that breaking the integrity of a row implies losing information.
- Column logic: The information presented makes sense on a columnar basis. In column logic, columns must be preserved.
- Matrix logic: Both dimensions must be preserved.

OUI supports tables of the first two kinds and makes sure that each device supports tables to the best of its abilities. OUI also attempts to display tables with matrix logic.

### Example

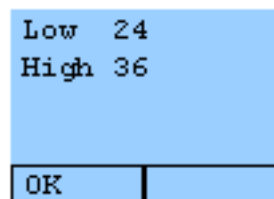
To build the following table with row logic,

LOW	24
HIGH	36

you can write the following code using the `Table` object:

```
Table myTable = new Table(Table.ROW_MODE, 2);  
  
myTable.addRow(new String[] { "Low", "24" });  
myTable.addRow(new String[] { "High", "36" });  
  
myCard.addElement(myTable);
```

On the Openwave Mobile Browser, row logic is rendered like this:



Low	24
High	36

OK

On the Nokia browser, row logic is rendered like this:

----- Table -----	
Low	24
High	36
Options	Back

To build the following table with column logic,

LOW	HIGH
24	36

you can write the following code using the Table object:

```
Table myTable = new Table(Table.COLUMN_MODE , 2);

myTable.addColumn(new String[] { "Low", "24" });
myTable.addColumn(new String[] { "High", "36" });

myCard.addElement(myTable);
```

On the Openwave Mobile Browser, column logic is rendered like this.

Low	High
24	36
OK	

On the Nokia browser, column logic is rendered like this.

----- Table -----	
Low	
24	
High	
36	
Options	Back

To build the following table with matrix logic,

	Low	High
Rome	24	36
Milan	22	31

you can write the following code using the Table object:

```
Table myTable = new Table(Table.MATRIX_MODE, 3);

myTable.setColumnHeaders(new String[] { "", "Low", "High" });
myTable.addRow(new String[] { "Rome", "24", "36" });
myTable.addRow(new String[] { "Milan", "22", "31" });

myCard.addElement(myTable);
```

On the Openwave Mobile Browser, matrix logic is rendered like this.

	Low	High
Rome	24	36
Milan	22	31

OK

On the Nokia browser, matrix logic is rendered like this.

----- Table -----	
<b>Rome</b>	
<b>Low</b>	<b>24</b>
<b>High</b>	<b>36</b>
<b>Milan</b>	
<b>Options</b>	<b>Back</b>

<b>Low</b>	<b>22</b>
<b>High</b>	<b>31</b>

## Methods

The possible values of tableMode are:

- Table.COLUMN\_MODE
- Table.MATRIX\_MODE
- Table.ROW\_MODE

Method	Description
<code>Table(int tableMode, int numberOfColumns)</code>	Creates an instance of the <code>Table</code> object in which:  <code>tableMode</code> is the logic or manner of interpreting the meaning of the information in the table. The value can be 0 for row logic, 1 for column logic, or 2 for matrix logic.  <code>numberOfColumns</code> is the number of columns of the table to be created.  This constructor requires a primitive integer value.
<code>Table(String tableMode, String numberOfColumns)</code>	Creates an instance of the <code>Table</code> object in which:  <code>tableMode</code> is the logic or manner of interpreting the meaning of the information in the table. The values can be <code>row</code> , <code>column</code> , or <code>matrix</code> .  <code>numberOfColumns</code> is the number of columns of the table to be created. This must be an integer value passed as a string.
<code>void addColumn(String[] items)</code>	Adds a column to the table. The table logic (see <code>tableMode</code> in constructor) must be set to <code>column</code> .
<code>void addRow(String[] items)</code>	Adds a row to the table. The table logic (see <code>tableMode</code> in constructor) must be set to <code>row</code> or <code>matrix</code> .
<code>void setAlignment(int align)</code>	Direct counterpart of the WML <code>align</code> attribute. This method sets the text alignment relative to the column. The values can be 0 for left alignment, 1 for center alignment, or 2 for right alignment.
<code>void setAlignment(String alignName)</code>	Direct counterpart of the WML <code>align</code> attribute. This method sets the text alignment relative to the column. The values can be <code>left</code> for left alignment, <code>center</code> for center alignment, or <code>right</code> for right alignment.
<code>void setColumnHeaders(String[] items)</code>	Specifies table headers for tables with matrix logic.
<code>void setTitle(String newTitle)</code>	Direct counterpart of the WML <code>title</code> attribute. This method sets the label for the table.



## Task

The `Task` object abstracts the familiar WML tasks, such as `go`, `prev`, `noop`, and `refresh`.

Tasks can be passed to `Anchor` and `DoElement` objects and to the navigation paths in cards and menus.

### Example

The following code creates a `go` task element with a postfield. The task is associated with the primary path for the card.

```
Task myTask = new Task("go", "#card2", "get");
myTask.addPostfield("firstname", "$firstname");
//activating the task is the primary activity on this card
myCard.addPrimaryPathTask(myTask, "card2", "go to card2");
```

### Methods

Method	Description
<code>Task()</code>	Creates an instance of the <code>Task</code> object.
<code>Task(String taskType)</code>	Creates an instance of the <code>Task</code> object in which: <code>taskType</code> is the type of task to be executed ( <code>go</code> , <code>prev</code> , <code>noop</code> , <code>refresh</code> , and so on).
<code>Task(String taskType, String taskURL)</code>	Creates an instance of the <code>Task</code> object in which: <code>taskType</code> is the type of task to be executed (such as <code>go</code> or <code>spawn</code> ). <code>taskURL</code> is the URL to open when the task is executed. This method is for tasks, such as <code>go</code> , that navigate to another URL.
<code>Task(String taskType, String taskURL, String method)</code>	Creates an instance of the <code>Task</code> object in which: <code>taskType</code> is the type of task to be executed (such as <code>go</code> or <code>spawn</code> ). <code>taskURL</code> is the URL to open when the task is executed. <code>method</code> specifies the HTTP submission method ( <code>get</code> or <code>post</code> ). This method is for tasks, such as <code>go</code> , that navigate to another URL.
<code>void addPostfield(String name, String value)</code>	Adds a WML <code>&lt;postfield&gt;</code> element to the task.

Method	Description
<code>void addSetvar(String name, String value)</code>	Adds a WML <code>&lt;setvar&gt;</code> element to the task.
<code>void setMethod(String method)</code>	Direct counterpart to the <code>method</code> attribute of the WML <code>&lt;go&gt;</code> element. Specifies the HTTP submission method ( <code>get</code> or <code>post</code> ) when passing data through an HTTP request.
<code>void setTaskType(String taskType)</code>	Specifies the type of task to be executed ( <code>go</code> , <code>prev</code> , <code>noop</code> , <code>refresh</code> , and so on).
<code>void setTaskURL(String taskURL)</code>	Specifies the URL to open when the task is executed.

## TaskMenu

The `TaskMenu` object encapsulates card-level menus for the graphical Openwave Mobile Browser, but renders such menus as an extra card on text-based browsers.

The graphical Mobile Browser supports long task menus in the form of a menu associated with the second softkey. On text-based browsers, such menus are isolated in new cards and made easily accessible. This mechanism replaces the traditional way of supporting multiple paths on text-based browsers. Output is enhanced on the graphical Mobile Browser and rendered gracefully on older browsers.

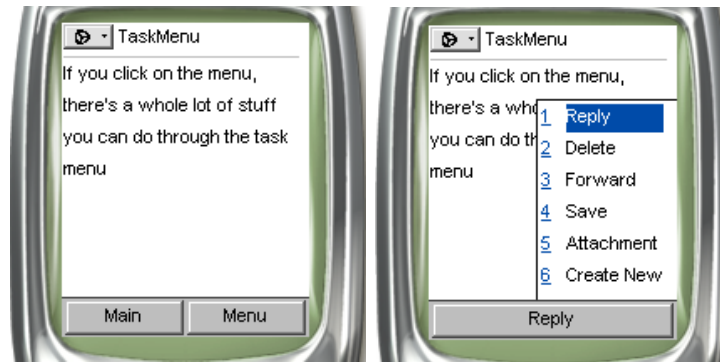
**NOTE** If you use a `TaskMenu` object you cannot define a secondary path, because these features rely on identical WML constructs and cover similar needs.

### Example

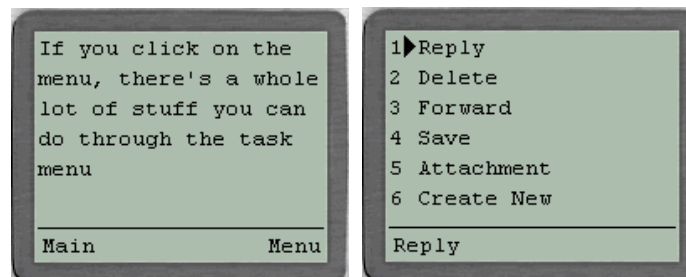
You can use the `TaskMenu` object to create a card-level menu as follows:

```
Card myCard = new Card("taskmenusample", "TaskMenu");
TaskMenu myTaskMenu = new TaskMenu(myDeck, "Menu", "Show Menu");
myTaskMenu.addEntry("#emailreply", "Reply", "Reply");
myTaskMenu.addEntry("#delete", "Delete", "Delete");
myTaskMenu.addEntry("#forward", "Forward", "Fwd");
myTaskMenu.addEntry("#save", "Save", "Save");
myTaskMenu.addEntry("#attach", "Attachment", "Attach");
myTaskMenu.addEntry("#new", "Create New", "New");
myCard.setTaskMenu(myTaskMenu);
```

On the graphical Mobile Browser, the task menu is quickly accessible through the second softkey.



On an Openwave text-based browser, the task menu is rendered as an extra card.



On the Nokia browser, the task menu is rendered in a new card.



### Methods

Method	Description
<code>TaskMenu(Deck deck, String shortTitle, String longTitle)</code>	Creates an instance of the <code>TaskMenu</code> object in which: <code>deck</code> is a WML deck. <code>shortTitle</code> is the short label for the <code>TaskMenu</code> . This text appears as the secondary softkey label for Openwave browsers. <code>longTitle</code> is the long label for the <code>TaskMenu</code> . This text appears as a link for Nokia browsers.
<code>void addEntry(String url, String title, String text)</code>	Adds a menu item without an icon in which: <code>url</code> is the URL to open when this menu item is chosen. <code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer. <code>text</code> is the device displays this text to represent the selection item.

Method	Description
<pre>void addEntry(String url, String title, String text, String localIcon)</pre>	<p>Adds a menu item with an icon in which:</p> <p><code>url</code> is the URL to open when this menu item is chosen.</p> <p><code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer.</p> <p><code>text</code> is the device displays this text to represent the selection item.</p> <p><code>localIcon</code> is the name or number identifying an icon to be displayed in front of the menu item.</p>
<pre>void addTaskEntry(AbsTask absTask, String title, String text, String localIcon)</pre>	<p>Adds a menu item with an icon but navigation implies triggering a task. The parameters are:</p> <p><code>absTask</code> is the task to be performed when the selection is chosen.</p> <p><code>title</code> is the label that identifies the option. The Openwave Mobile Browser uses the title as the ACCEPT key label when the user selects the option. To ensure compatibility on a wide range of devices, the label should be five characters, or fewer.</p> <p><code>text</code> is the text the device displays to represent the selection item.</p> <p><code>localIcon</code> is the name or number identifying an icon to be displayed in front of the menu item.</p>

## Template

The `Template` object encapsulates the WML `<template>` element.

This object defines deck-level event bindings, for example, characteristics that apply to all cards in a deck. However, you can override these characteristics for a particular card by specifying the same event bindings in the `Card` object.

### Example

```
Template myTemplate = new Template();

//Define a task.
Task myTask = new Task("go", "home.wml");
DoElement myDoElement = new DoElement(myTask, "options", "home");

myTemplate.addDoElement(myDoElement);

myDeck.addTemplate(myTemplate);
```

### Methods

Method	Description
<code>Template()</code>	Creates an instance of the <code>Template</code> object.
<code>void addDoElement(DoElement doElement)</code>	Adds a <code>DoElement</code> object (see “ <code>DoElement</code> ” on page 30).
<code>void addOnevent(Onevent doElement)</code>	Adds an <code>Onevent</code> object (see “ <code>Onevent</code> ” on page 56).

## Timer

The `Timer` object encapsulates the WML `<timer>` element. You can add a `Timer` object to a card to set a timer that will trigger an event when the timer expires.

**NOTE** You must qualify the `Timer` class to avoid conflicting with the Java `Timer` utility in `java.util.Timer`.

### Example

```
com.openwave.oui.waomelements.Timer myTimer = new com.openwave.oui.waomelements.Timer()

myTimer.setName("ToCard2");
myTimer.setName("10");
Card myCard = new Card("timersample", "Timer");
myCard.setOnTimer("#someURL");
myCard.setTimer(myTimer);
```

### Methods

Method	Description
<code>Timer()</code>	Creates an instance of the <code>Timer</code> object.
<code>Timer(String time)</code>	Creates an instance of the <code>Timer</code> object in which: <code>time</code> is the direct counterpart of the WML <code>value</code> attribute of the <code>&lt;timer&gt;</code> element. This method specifies the length of time (in units of 1/10 seconds) to wait before triggering an event.
<code>void setName(String name)</code>	Direct counterpart of the <code>name</code> attribute. This method specifies the name of the timer.
<code>void setTime(String time)</code>	Direct counterpart of the <code>value</code> attribute. This method specifies the length of time (in units of 1/10 seconds) to wait before triggering an event.

