OPENWAVE™

Getting Started

Openwave Usability Interface, Java Edition 1.0 Beta

Openwave Systems Inc.
1400 Seaport Boulevard
Redwood City, CA 94063
http://www.openwave.com

Part Number OUGS-10-006
November 2001

Contents

Preface

About This Book

This book shows you how to use the Openwave™ Usability Interface (OUI), Java Edition 1.0 Beta to create a single Wireless Markup Language (WML) application that operates optimally on various types of Wireless Application Protocol (WAP) devices.

Audience

This book is intended for WAP developers who want to create and host WML applications that run well on more than one type of mobile browser.

You can write OUI applications using the Java language or the OUI XHTML or WML tag libraries. Depending on the method you choose, you will need a background in that technology.

Related Documentation

OUI comes with the following documentation:

- *Installation and Integration* describes how to install and configure OUI and the Java server and development software required to host and develop OUI applications. It also includes some basic OUI examples.
- *Getting Started* (this book) describes OUI and how to use it.
- The *Object Model Reference* describes the details about the Java implementation for each OUI object.
- The *WML Tag Library Reference* describes each WML tag library tag and attribute.
- The *XHTML Tag Library Reference* describes each XHTML Mobile Profile tag library tag and attribute.

The Openwave SDK comes with documentation for XHTML-MP, WML and WML script, and related topics. For a complete list of documentation, see:

```
http://developer.openwave.com
```

Technical Support

The best resource for up-to-date information on using OUI is the Openwave Developer site:

`http://developer.openwave.com`

In addition to the downloadable OUI, this site contains a variety of useful resources, including Frequently Asked Questions, bug reporting, technical support, and an interactive developer forum.

Other Resources

WAP

- WAP Forum: `http://www.wapforum.com`
- WML Pulse Europe: `http://wmlpulse-europe.openwave.com`
- WAP FAQs: `http://www.allnetdevices.com/faq/`
- WML Forum: `http://groups.yahoo.com/group/wmlprogramming`
- WMLScript.com: `http://www.wmlscript.com`
- WirelessDeveloper.com: `http://www.wirelessdeveloper.com`
- WirelessDevNet.com: `http://www.wirelessdevnet.com`

Java

- Allaire JRun: `http://www.jrun.com/`
- Jakarta Tomcat: `http://jakarta.apache.org/`
- Resin-CMP: `http://www.caucho.com`
- SUN JSP: `http://java.sun.com/products/jsp`
- SUN Servlets: `http://java.sun.com/products/servlet`

Style and Typographical Conventions

This manual uses different fonts to represent the information that you enter:

- `Text that appears like this` identifies command names, path names, URLs, and specific text that you must enter.
- *`Text that appears like this`* identifies placeholders or variables that you should replace with values appropriate to your environment.

Introducing OUI 1

You can use the Openwave Usability Interface (OUI) to build a single wireless application that delivers the best possible user experience to a wide variety of Wireless Application Protocol (WAP) devices. The current release of OUI is a developer library implemented in Java, plus tag libraries modeled on the Wireless Markup Language (WML) and XHTML Mobile Profile (XHTML-MP).

You can download OUI at no charge from the Openwave Developer Web site:

```
http://developer.openwave.com
```

The best tool for writing OUI applications is the Openwave SDK 5.1 Preview, which is also available free of charge from the Openwave Developer Web site. This edition of the Openwave SDK features an integrated development environment (IDE), with a text editor that recognizes and can highlight the syntax of code you write using the OUI WML and XHTML tag libraries. The IDE also provides a simulator for the graphical Openwave Mobile Browser, which you can use to test your OUI applications.

You host OUI applications on a web server that is configured to support Java Server Pages (JSP) and servlets. One example is the Tomcat Java server from the Apache Software Foundation. Installing and configuring a Java server, a Java SDK, and OUI so you can start developing and hosting OUI applications is described detail in the OUI *Installation and Integration* book.

The WML Dilemma

WML is a markup language based on Extensible Markup Language (XML). It is designed for specifying user interface behavior and displaying content on wireless devices such as phones, pagers, and personal digital assistants (PDAs).

The Wireless Application Protocol (WAP) Forum is an industry organization dedicated to developing open standards for wireless communication. It has provided a formal specification for WML, which includes the Document Type Definition (DTD) for WML. See the WAP Forum web site for full WAP specifications:

```
http://www.wapforum.org
```

In practice, the manufacturers of WML browsers and the devices they run on have interpreted the WML standard very differently, so WML applications that run well on one device are often unusable or simply fail on other devices.

This has left developers with difficult choices: Write WML applications that work well on one device but work poorly (if at all) on others, or write and maintain two or more versions of the same application. A third choice, writing a lowest-common-denominator application that works on all devices has proved equally frustrating: The set of common features is small, and shrinks each time a new device is introduced.

The OUI Solution

OUI solves this dilemma by offering a set of high-level APIs that encapsulate the building blocks of ideally usable WML applications.

When you build a OUI application out of these building blocks and a user connects to it, OUI recognizes the details of the user's WAP browser, device, and gateway and delivers the most appropriate and usable WML code: You don't need to write and maintain multiple versions of your applications to offer the best user experience to a wide array of mobile devices.

OUI is optimized for the current generations of graphical and text-based Openwave Mobile Browsers (4.x and 5.x) and for Nokia WAP 1.1 browsers. For current or future devices that OUI doesn't recognize, OUI delivers a generic version of WML that ensures your applications will run. As new WAP devices appear on the market, new OUI libraries can support them without your having to change your current code: Your applications will simply run better.

An Example with Menus

Menus are a feature of WML applications that are interpreted quite differently on different browsers.

The WML code segment shown in Listing 1-1 provides optimum usability for menu navigation on Nokia browsers.

Listing 1-1.

```
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"

"http://www.wapforum.org/DTD/wml_1.1.xml">

<!--  Control examples: Menu Navigation on Nokia  -->

<wml>

 <card  id="testcard" title="short">
  <p align="left" mode="wrap" >
    <a title="Weather" href="#weath">Weather</a>
    <a title="Eat" href="#eat">Restaurants</a>
    <a title="Movie" href="#cinema">Movie Theaters</a>
    <a title="Gamble" href="#gamble">Casinos</a>
    <a title="Zoo" href="#zoo">Zoologic Park</a>
    <a title="Fun" href="#fun">Fun for family</a>
  </p>
 </card>
</wml>
```

When this code is run on an Openwave Mobile Browser, users are deprived of a useful shortcut to access menu items: they cannot use the numeric keypad to jump to an option. Instead, they must scroll down to reach it.

To optimize the WML code for Openwave Mobile Browser users, you can take the approach shown in Listing 1-2.

Listing 1-2.

```
<?xml version="1.0" ?>
<!DOCTYPE wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
"http://www.phone.com/dtd/wml11.dtd">


<!--  Control examples: Menu Navigation on the Openwave Mobile
Browser  -->

<wml>
 <card  id="testcard" title="short">
  <p align="left" mode="nowrap" >
   <select>
    <option title="Weather" onpick="#weath">
     <img alt="" src="" localsrc="sun" />Weather
    </option>
    <option title="Eat" onpick="#eat">
      <img alt="" src="" localsrc="meal1" />Restaurants
    </option>
    <option title="Movie" onpick="#cinema">
      <img alt="" src="" localsrc="camcorder" />Movie Theaters
    </option>
    <option title="Gamble" onpick="#gamble">
      <img alt="" src="" localsrc="dice" />Casinos
    </option>
    <option title="Zoo" onpick="#zoo">
      <img alt="" src="" localsrc="camera1" />Zoologic Park
    </option>
    <option title="Fun" onpick="#fun">
      <img alt="" src="" localsrc="family" />Fun for family
    </option>
   </select>  </p>
 </card>
</wml>
```

Unfortunately, this code degrades usability on the Nokia browser to an unacceptable level: The Nokia browser actually breaks on this code because of the in-line images, which depend on WML extensions. If the images were removed the Nokia would not break, but usability would be poor. For this reason, many developers use hyperlinks to build menus. They adopt the “generic WML” model—that is, coding for a generic browser.

However, this approach has some serious problems. It deprives the owners of advanced WAP devices of many of the possibilities offered by their handsets. This is a direct consequence of the “least common denominator” approach. In particular, the generic approach prevents users from fully exploiting the capabilities of the Openwave Mobile Browser.

This is not the only serious problem. Let’s assume that you have chosen the generic approach for your application and that a new WAP device is released onto the market. The fact that you have already compromised in terms of user interface capabilities does not protect you from new devices with different interpretations that require you to change your code again and possibly further degrade the overall usability of your application.

OUI provides a solution that allows you to create and maintain a single version of your application while supporting optimal usability on a multitude of devices.

Consider the segment of OUI WML tag library code shown in Listing 1-3:

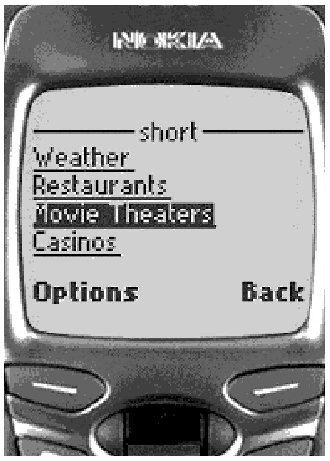
Listing 1-3.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>
   <oui:card id="testcard" title="short">
      <oui:p align="left" mode="nowrap">
         <oui:menu>
            <oui:menu_item title="Weather" href="#weath" text="Weather" icon="sun"/>
            <oui:menu_item title="Eat" href="#eat" text="Restaurants" icon="meal1"/>
            <oui:menu_item title="Movie" href="#cinema" text="Movie
               Theaters"icon="camcorder" />
            <oui:menu_item title="Gamble" href="#gamble" text="Casinos"icon="dice" />
            <oui:menu_item title="Zoo" href="#zoo" text="Zoologic Park"icon="camera1" />
            <oui:menu_item title="Fun" href="#fun" text="Fun for family"icon="family" />
         </oui:menu>
      </oui:p>
   </oui:card>
</oui:wml>
```

The `Menu` object is a OUI abstraction representing a navigation menu. The object is created and progressively enriched with information about the menu items.

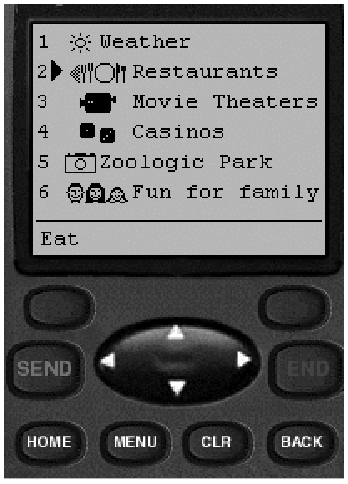
When this file is served to a user, OUI determines the requesting device and returns one of the WML versions shown in Listing 1-1 and Listing 1-2, depending on the family of the device: Hyperlinks for Nokia handsets, `<select>/<option>` for the Openwave Mobile Browser, and hyperlinks followed by line breaks for generic WAP devices.

This application produces the following output on a Nokia handset.



This illustration shows the best output that the handset can manage when implementing menu-like navigation.

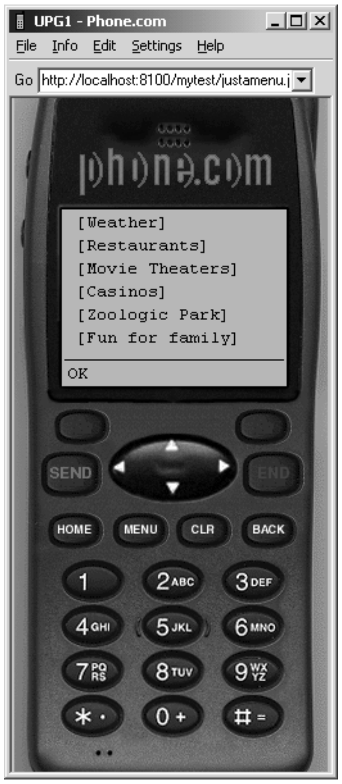
The Openwave Mobile Browser is better served by `<select>/<option>` elements (and possibly some in-line icons), as shown in the following figure.



```
1  Weather
2  Restaurants
3  Movie Theaters
4  Casinos
5  Zoologic Park
6  Fun for family
Eat
```

This is a great improvement over the generic WML, which many developers use in GSM areas.

Compare the previous figure with what generic WML would produce on the same phone.



```
[Weather]
[Restaurants]
[Movie Theaters]
[Casinos]
[Zoologic Park]
[Fun for family]
OK
```

UPG1 - Phone.com
File Info Edit Settings Help
Go http://localhost:8100/mytest/justamenu.j
phone.com
SEND
END
HOME
MENU
CLR
BACK

This is what Openwave Mobile Browser users experience in GSM areas. Usability is much lower than the level that the handsets are actually capable of providing.

Getting Started

The current version of OUI offers you a number of choices for how you write your OUI applications:

- You can write your applications in Java, to take advantage of the powerful Java programing environment.
- You can write your applications using the OUI WML tag library, a quick and easy way to start writing OUI applications, particularly if you're already familiar with WML.
- You can write your applications using the OUI XHTML tag library, which is also much simpler than the OUI Java APIs, and which you can use to experiment with XHTML Mobile Profile, the new mark-up standard specified by the WAP forum.
- You can write applications that mix these approaches, taking advantage of the strengths of each.

The next chapter in this book, "Creating a OUI Service,", gets you started writing a simple application using each of these approaches.

The final chapter in this book, "OUI Application Fundamentals," offers greater detail about the most important building blocks of OUI applications.

Creating a OUI Service 2

The information in this chapter will help you begin to create and preview OUI applications and services.

Installing OUI and a Java Server

To use the Openwave Usability Interface (OUI), you must first install and configure OUI and a Java Application Server. See OUI *Installation and Integration* for instructions.

Openwave SDK

The Openwave SDK 5.1 Preview provides an Integrated Development Environment (IDE) that helps you develop OUI services. The IDE integrates a number of mobile browser simulators that allow you to preview your services on several phone models. It also provides file headers for new JSP files and syntax coloring for OUI tag library files in the editor.

You can download the Openwave SDK 5.1 Preview from the Openwave Developer web site:

```
http://developer.openwave.com
```

OUI Programming

You can write OUI applications using the Java language or the OUI XHTML or WML tag libraries. OUI applications are implemented as Java Server Pages (JSP) or Java servlets. You can use the examples in the this chapter to learn about OUI and to determine the OUI programming tools that make the most sense for your needs.

In each example, a different OUI technology is used to create the same application for a mobile browser device.

WML Tag Library

The following example shows a “Hello World” application using the OUI WML tag library.

1 **In the application server `webapps` directory, create a new directory called `testing`.**

   For Tomcat:

   ```
   %TOMCAT_HOME%\webapps\ROOT\testing
   ```

2 **Enter the following code in a text editor (such as the Openwave IDE).**

   ```
   <%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
   <oui:wml>
       <oui:card title="WML Greeting" id="main">
         <oui:p mode="nowrap">
           Hello World!
         </oui:p>
       </oui:card>
   </oui:wml>
   ```

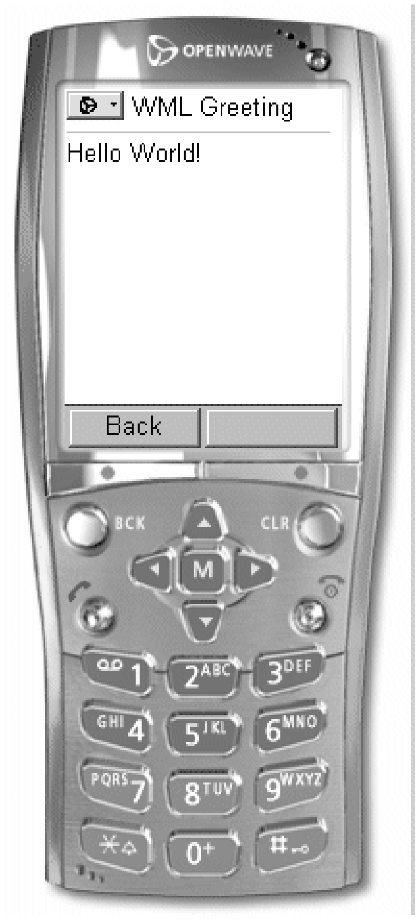
3 **Save the file in the new `testing` directory as `wmlHello.jsp`.**

4 **Open the file in a web browser or in the Openwave IDE.**

   For Tomcat, the URL is:

   ```
   http://localhost:8080/testing/wmlHello.jsp
   ```

Figure 2-1. WML JSP display



XHTML Tag Library

1. **Enter the following code in a text editor (such as the Openwave IDE).**

```
<%@ taglib uri="/WEB-INF/tld/xhtmloui.tld" prefix="oui" %>
<oui:html xmlns="http://www.w3.org/1999/xhtml">
    <oui:head>
      <oui:title>XHTML Greeting</oui:title>
    </oui:head>
    <oui:body>
      <oui:p align="center">
        Hello World!
      </oui:p>
    </oui:body>
</oui:html>
```

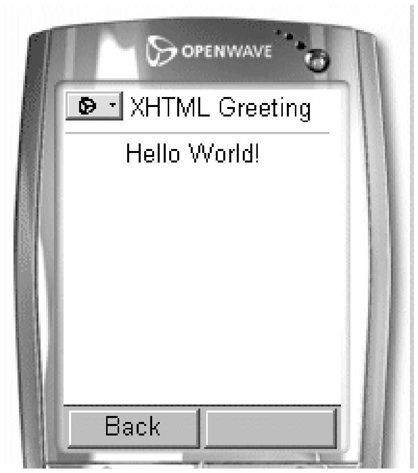
2. **Save the file in the** `testing` **directory as** `xhtmlHello.jsp`**.**

3 Open the file in a web browser or in the Openwave IDE.

For Tomcat, the URL is:

`http://localhost:8080/TestXhtml/xhtmlHello.jsp`

Figure 2-2. XHTML JSP display



Hello World!

Java Servlets

The following example shows you how to create a “Hello World” application that is a Java servlet.

1 **Enter the following code in a text editor.**

```
import java.util.*;
import com.openwave.oui.framework.*;
import com.openwave.oui.waomelements.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

    public void doGet(HttpServletRequest request,
        HttpServletResponse response)
        throws javax.servlet.ServletException,
        java.io.IOException {

        performTask(request, response);
    }
    public void doPost(HttpServletRequest request,
        HttpServletResponse response)
        throws javax.servlet.ServletException,
        java.io.IOException {

        performTask(request, response);
    }
    public void performTask(HttpServletRequest request,
        HttpServletResponse response) {

        try {
            DeviceContext dc = new DeviceContext
                (request,response);
            Deck myDeck = new Deck();
            Card myCard = new Card("hello","A Greeting");
            myCard.setTitle("A Greeting");
            myCard.beginParagraph();
            myCard.addText("Hello World!");
            myCard.endParagraph();
            myDeck.addCard(myCard);
            dc.render(myDeck);
        }
        catch (Throwable theException) {
            theException.printStackTrace();
        }
    }
}
```

2 **Save the file as `HelloWorld.java` in the directory where the Java server stores Java class files.**

For Tomcat, the class directory is:

```
%TOMCAT_HOME\webapps\ROOT\WEB-INF\classes
```

3 **Choose Start > Run.**

4 **In the Run dialog box, enter** `cmd` **and click OK.**

5 **In the DOS command prompt window, change to the directory in which the Java server stores Java class files.**

For Tomcat, Enter:

```
cd %TOMCAT_HOME%\webapps\ROOT\WEB-INF\classes
```

6 **Enter**

```
javac HelloWorld.java
```

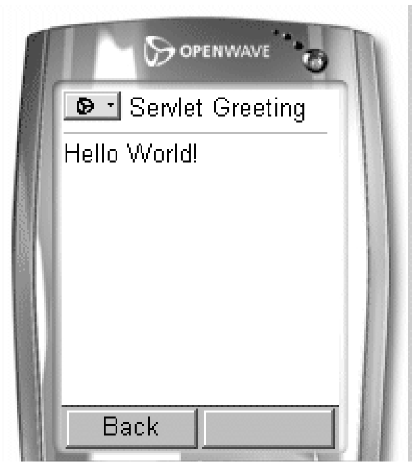
The class directory now contains the `HelloWorld.class` file, which can be accessed from a web browser or the SDK simulator. When accessing this file, don’t include the `.class` extension in the URL.

7 **Open the file in a web browser or in the Openwave IDE.**

For Tomcat, the URL is:

```
http://localhost:8080/servlet/HelloWorld
```

Figure 2-3. WML servlet display



Hello World!

Previewing Your OUI Application

There are a number of ways to preview your OUI application.

Using the Openwave SDK

When developing OUI applications, you should use the Openwave SDK 5.1 Preview, which you can download without charge at:

```
http://developer.openwave.com
```

When cross-testing, you should use SDKs associated with the other WAP devices for which you are developing.

It is highly recommended that you also test your final applications on real devices running their respective browsers. This procedure helps to establish minor or subtle differences between simulators and actual devices and it ensures that your applications operate correctly on real devices.

Examples of using the Openwave SDK to preview OUI applications are shown in Figure 2-1, Figure 2-2, and Figure 2-3.

Using Other SDKs

Because OUI produces wireless services that can be displayed by many WAP compliant mobile devices, it is useful to test your applications on SDKs that simulate these other devices. Many of these can be obtained at no charge.

Using Mobile Browser Devices

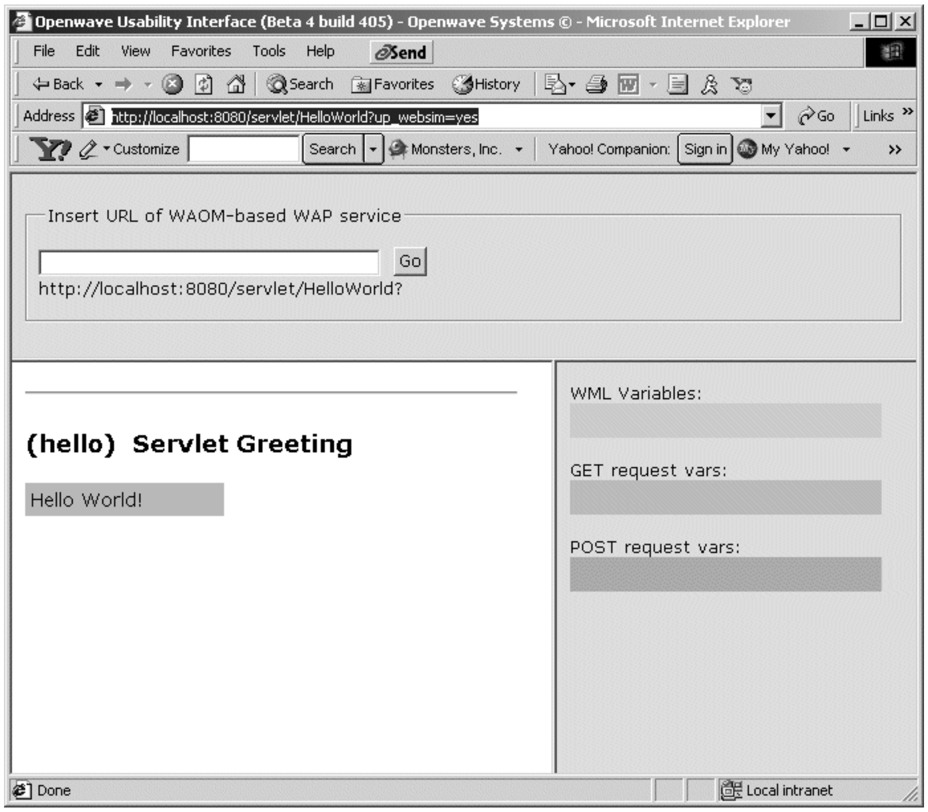
While simulators and development environments are a good way to test your OUI application, the best way is to use actual mobile devices. To test on a mobile device, you must have access to a server approved by the carrier to handle web services for the device. Make sure that the Java server is properly installed and your OUI applications are placed in the proper directories. Then you can use the mobile browser in the device to navigate to your OUI application.

Using MS Internet Explorer 5.x

One advantage of using OUI is that you can preview your applications in one of the most commonly available web browsers: Microsoft Internet Explorer 5.x (IE).

Pointing that browser at the application URL produces the result shown in Figure 2-4. The OUI application functions much as it would in a mobile browser.

Figure 2-4. Previewing using MS IE


Openwave Usability Interface (Beta 4 build 405) - Openwave Systems © - Microsoft Internet Explorer
File Edit View Favorites Tools Help Send
Back Search Favorites History
Address http://localhost:8080/servlet/HelloWorld?up_websim=yes
Go Links
Customize Search Monsters, Inc. Yahoo! Companion: Sign in My Yahoo!
Done
Local intranet

Insert URL of WAOM-based WAP service

Go

http://localhost:8080/servlet/HelloWorld?

(hello) Servlet Greeting

Hello World!

WML Variables:

GET request vars:

POST request vars:

If there are errors in your Java code, Explorer displays Java error codes that aren't available in a mobile browser. You may find this feature of OUI valuable in debugging your applications and detecting server-side problems.

Raw Mode: An Extra Operation Mode

Through its abstractions, OUI encourages the creation of applications that offer a high level of usability. However, in certain cases you may require full control of WML.

OUI is flexible enough to address such cases, by providing a mechanism called *Raw Mode.*

All OUI objects support the Raw Mode and configuring any object in Raw Mode results in the following:

1 When rendering, the object mode is ignored.

2 The differences between various devices are ignored.

3 The particular segment of WML markup associated with the object.

Use Raw Mode sparingly—not as part of your everyday OUI programming. Do not resort to the programming style shown in Listing 2-1unless you are an advanced programmer and very familiar with OUI.

Listing 2-1. Building a task using extensions not supported by the object model

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

Deck myDeck = new Deck();

Card myCard = new Card("Raw","Task is Raw");
myCard.beginParagraph();

myCard.addText("Just checking raw mode on task primary path ");

Task myTask = new Task();
myTask.setToRawMode();

myTask.setDirectDefinition("<spawn
           href=\"http://www.openwave.com\"><catch
/></spawn>");

myCard.addPrimaryPathTask(myTask,"Spawn","Spawn");

myCard.endParagraph();
myDeck.addCard(myCard);

dc.render(myDeck);
```

OUI Application Fundamentals 3

This chapter provides more information about the main OUI abstractions, along with some examples using the WML tag library (and in one case, the XHTML tag library). It should help you start experimenting with OUI.

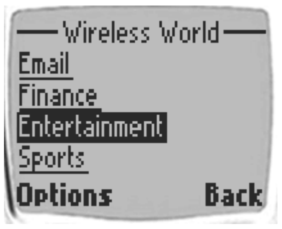
Menus

In a way, menus are the most obvious OUI abstraction. The Nokia WML model forces developers to use lists of hyperlinks. Openwave phones are better served by the `<select>/<option>` WML construct, because it provides users with numbered lists and numeric key accelerators. In addition, the Openwave Mobile Browser is the only widely deployed WAP browser that allows in-line icons on menus. This last feature is typically left unexploited in Europe for the sake of interoperability across different phones.

OUI menus make it possible for you to write applications with advanced features, and the library makes sure that your applications degrade gracefully on devices that do not fully support those features.

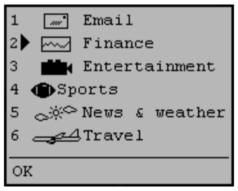
```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>
 <oui:card id="start" title="Wireless World">
  <oui:p align="left" mode="nowrap">
   <oui:menu>
     <oui:menu_item href="ema.jsp" text="Email" icon="envelope1" />
     <oui:menu_item href="fin.jsp" text="Finance" icon="graph1" />
     <oui:menu_item href="ent.jsp" text="Entertainment" icon="videocam" />
     <oui:menu_item href="spo.jsp" text="Sports" icon="football" />
     <oui:menu_item href="new.jsp" text="News &amp; weather" icon="partcloudy" />
     <oui:menu_item href="tra.jsp" text="Travel" icon="plane" />
     <oui:menu_item href="sho.jsp" text="Shopping" icon="dollarsign" />
     <oui:menu_item href="oth.jsp" text="Other" icon="folder1" />
   </oui:menu>
  </oui:p>
 </oui:card>
</oui:wml>
```

The sample code is rendered like this for the Nokia browser:

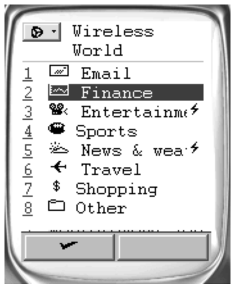


Openwave phones, though, have a different way to render menus: numbered lists (note the in-line icons).

The sample code is rendered like this for text-based Openwave Mobile Browsers:



The sample code is rendered like this for graphical Openwave Mobile Browsers:



Please observe that in-line icons are not just a fancy add-on: they improve users' interactive experiences. For example, you can use icons on the first page of a calendar application to visualize that an appointment is taking place within 16 hours. What user could resist the temptation to hit your application at least once each day?

Forms and Primary Paths

Forms are sets of widgets that collect user input. Nokia browsers and text-based and graphical Openwave Mobile Browsers support forms differently (see “Elective Forms vs. Wizards” in the *Openwave Usability Guidelines*).

With the OUI Form object, you assemble your input elements and picker objects (a picker object is essentially a `<select>` construct with the single purpose of picking an item out of a list).

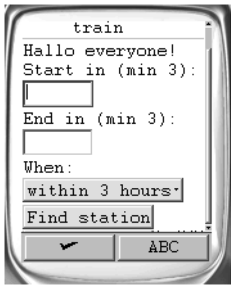
```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>
  <oui:form id="train" title="Find your train">

    Start in (min 3):
    <oui:input type="text" size="6" title="from" name="startstation" value="" />


    End in (min 3):
    <oui:input type="text" size="6" title="to" name="endstation" value="" />

    When:
      <oui:picker name="when" title="when">
        <oui:option value="3">within 3 hours</oui:option>
        <oui:option value="9">within 9 hours</oui:option>
        <oui:option value="0">whenever</oui:option>
      </oui:picker>

//Primary Path
    <oui:primary_path short_label="Find" long_label="Find station">
      <oui:go href="search.jsp">
        <oui:postfield name="startstation" value="$startstation" />
        <oui:postfield name="endstation" value="$endstation" />
        <oui:postfield name="when" value="$when" />
      </oui:go>
    </oui:primary_path>

  </oui:form>
</oui:wml>
```

If you know WML, this code should make sense: A form with three form elements. The only part that may be unclear is the primary path. According to the Openwave usability guidelines, a primary path is the activity that most users are likely to perform when accessing a certain card. On the Openwave Mobile Browser, primary paths should be associated with the main softkey (the Accept key) through the `<do type="accept">` WML construct. On browsers without softkey support, primary paths are often more usable if implemented with a hyperlink. As a consequence, there is no generic WML best practice that guarantees satisfactory results for all phones.

OUI fixes this by letting you specify the intended main activity for each card. This abstraction has been conveniently called “primary path”.

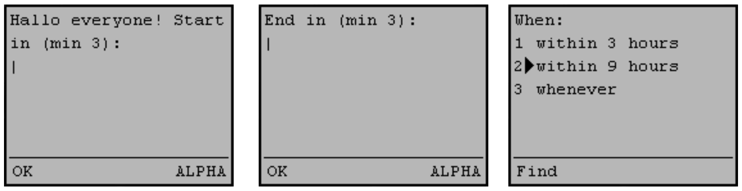
NOTE A primary path can contain either a URL or a more complex task (as in the example above). In addition, you can specify two labels: long and short. The short one is used for softkeys; the long one is used for hyperlinks.

The sample code is rendered like this for the Nokia browser:



```
train
Hallo everyone!
Start in (min 3):
|
End in (min 3):

When:
within 3 hours
Find station
```

The sample code is rendered like this for text-based Openwave Mobile Browsers:

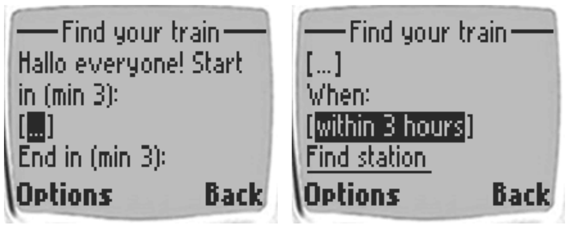


```
Hallo everyone! Start
in (min 3):
|
OK                ALPHA
```

```
End in (min 3):
|
OK                ALPHA
```

```
When:
1 within 3 hours
2 within 9 hours
3 whenever
Find
```

The sample code is rendered like this for graphical Openwave Mobile Browsers:



```
Find your train
Hallo everyone! Start
in (min 3):
[...]
End in (min 3):
Options          Back
```

```
Find your train
[...]
When:
[within 3 hours]
Find station
Options          Back
```

XHTML Tag Library

While this doesn’t dig deeply into the details of the XHTML tab library, here is how the from in the previous example is implemented using it:

```
<%@ taglib uri="/WEB-INF/tld/xhtmloui.tld" prefix="oui" %>

<oui:html xmlns="http://www.w3.org/1999/xhtml">
  <oui:head>
    <oui:title>Find your train</oui:title>
  </oui:head>
  <oui:body>
    <oui:form action="search.jsp">

    Start in (min 3):
    <oui:input type="text" size="6" name="startstation" value="" />
      <oui:br/>
    End in (min 3):
    <oui:input type="text" size="6" name="endstation" value="" />
      <oui:br/>
    When:<oui:br/>

    <oui:select name="when">
      <oui:option value="3">within 3 hours</oui:option>
      <oui:option value="9">within 9 hours</oui:option>
      <oui:option value="0">whenever</oui:option>
    </oui:select>

    <oui:input type="submit" value="Find station" name="submit"/>

    </oui:form>
  </oui:body>
</oui:html>
```

Of course OUI still delivers WML to actual WAP devices, but that doesn’t prevent you from experimenting with a new technology.

Secondary Paths and Side Paths

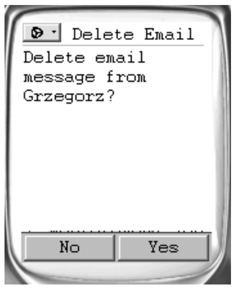
Primary paths were discussed in the previous sections. You should always code your applications so that there is only one primary path. Of course, in many cases you need to offer users multiple choices. Secondary paths serve this purpose. In general, a secondary path is an activity that many users perform occasionally.

You can code as many secondary path as you want in each card. Even none. If you are coding more than four or five secondary paths for a single card, though, you ask yourself whether you’ve picked the best possible structure: clogging the interface of a mobile application is a very bad idea.

Here is an example of a secondary path:

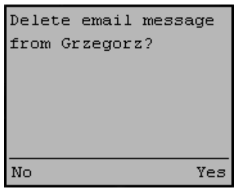
```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="delete" title="Delete Email">
    <oui:p>
      Delete email message from Grzegorz?

    <oui:primary_path short_label="No">
      <oui:prev />
    </oui:primary_path>

    <oui:secondary_path short_label="Yes">
      <oui:go href="delete.jsp">
        <oui:postfield name="emailid" value="573ad8sd9f994da8798" />
      </oui:go>
    </oui:secondary_path>

    </oui:p>
  </oui:card>
</oui:wml>
```

The sample code is rendered like this for graphical Openwave Mobile Browsers:



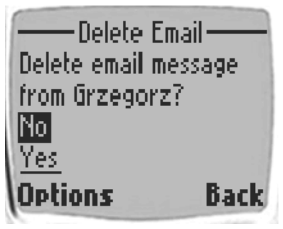
Delete email message from Grzegorz?

The sample code is rendered like this for text-based Openwave Mobile Browsers:



Delete email message from Grzegorz?

The sample code is rendered like this for graphical Nokia:


Delete Email
Delete email message
from Grzegorz?
No
Yes

OUI also supports the concept of side paths, that is, activities that a few users will perform sparingly.

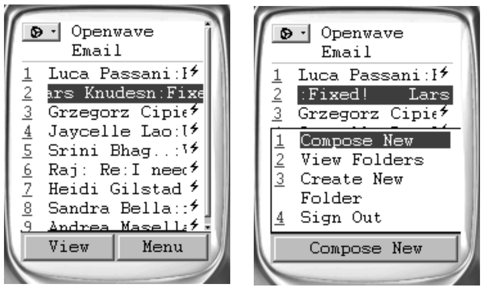
TaskMenus

The new graphical Openwave Mobile Browser has a graphical widget that increases usability, the task menu. A task menu is a bit like the windows Start button. With the task menu, users don’t lose the context of the card they are operating, since the content of the card is only partially covered by the menu. A task menu is an excellent way to group numerous secondary paths. When you use a task menu, you get the advanced behavior for the graphical browser. Text-based browsers show the menu in a separate card; access to that card is implemented as a secondary path for the card with the menu.

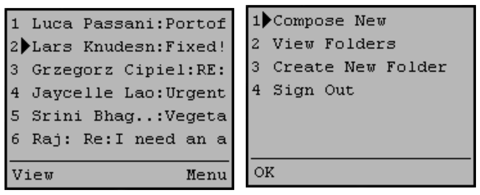
NOTE OUI does not let you implement a secondary path and a task menu for the same card.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>
  <oui:card id="Inbox" title="Openwave Email">
    <oui:p mode="nowrap">
      <oui:menu>
        <oui:menu_item href="ema.jsp?344b53" title="View" text="Luca
           Passani:Portofino" />
        <oui:menu_item href="ema.jsp?345a29" title="View" text="Lars Knudesn:
           Fixed!" />
        <oui:menu_item href="ema.jsp?345a49" title="View" text="Grzegorz
            Cipiel:RE:Installer.." />
        <oui:menu_item href="ema.jsp?345a45" title="View" text="Jaycelle Lao:
           Urgent Issue  " />
        <oui:menu_item href="ema.jsp?345a59" title="View" text="Srini
           Bhag..:Vegetarian recipies." />
        <oui:menu_item href="ema.jsp?345a59" title="View" text="Raj: Re:I need an
           answer" />
        <oui:menu_item href="ema.jsp?345a23" title="View" text="Heidi Gilstad:
           RE:Copenhagen" />
        <oui:menu_item href="ema.jsp?345a24" title="View" text="Sandra Bella:is
           Tivoli open?" />
        <oui:menu_item href="ema.jsp?345a25" title="View" text="Andrea
            Masella:Stasera? " />
      </oui:menu>
    </oui:p>
      <oui:task_menu short_label="Menu" long_label="Show Menu">
        <oui:menu_item href="#compose" text="Compose New"/>
        <oui:menu_item href="#viewfldr" text="View Folders"/>
        <oui:menu_item href="#newfldr" text="Create New Folder"/>
        <oui:menu_item href="#signout" text="Sign Out"/>
      </oui:task_menu>
  </oui:card>
</oui:wml>
```

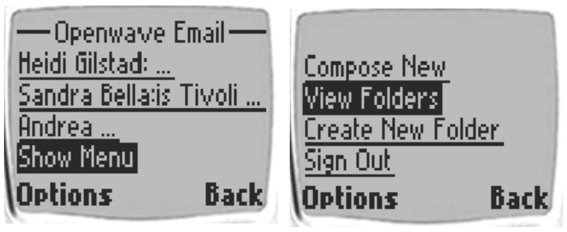
The sample code is rendered like this for graphical Openwave Mobile Browsers:



The sample code is rendered like this for text-based Openwave Mobile Browsers:



The sample code is rendered like this for Nokia browsers:



ComboMenu

Another nice feature of the new graphical Openwave Mobile Browser is its ability possibility to display multiple widgets in the same card. One commonly used configuration is a Menu followed by another bit of WML code or user-input constructs. Unfortunately, the WML code to achieve this effect on the graphical browser does not degrade gracefully on a text-based browser.

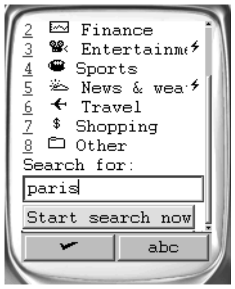
OUI offers a solution in the form of a ComboMenu object. A ComboMenu requires two components:

- A Menu
- An Appendix: Some UI controls (or just a bit of WML code) conveniently wrapped into an object called Appendix.

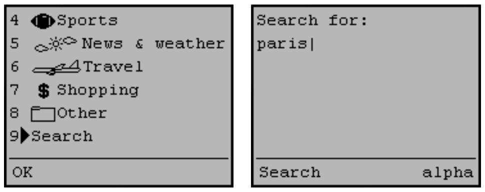
When rendering, graphical browsers receive the fully fledged composite GUI. Text-based browsers receive a card with a menu and a card with the controls in the appendix. A new menu item is conveniently added as the last one of the menu to point to the new card for the textual phone.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="start" title="Wireless World">
    <oui:p align="left" mode="nowrap">
      <oui:combo_menu>
        <oui:menu>
          <oui:menu_item href="ema.jsp" text="Email" icon="envelope1" />
          <oui:menu_item href="fin.jsp" text="Finance" icon="graph1" />
          <oui:menu_item href="ent.jsp" text="Entertainment" icon="videocam" />
          <oui:menu_item href="spo.jsp" text="Sports" icon="football" />
          <oui:menu_item href="new.jsp" text="News &amp; weather" icon="partcloudy" />
          <oui:menu_item href="tra.jsp" text="Travel" icon="plane" />
          <oui:menu_item href="sho.jsp" text="Shopping" icon="dollarsign" />
          <oui:menu_item href="oth.jsp" text="Other" icon="folder1" />
        </oui:menu>

        <oui:appendix title="Search">
          <oui:p>
            Search for:
            <oui:input type="text" name="keyword" value="" title="keyword" />
          </oui:p>
          <oui:primary_path href="" short_label="Search" long_label="Search now" />
        </oui:appendix>
      </oui:combo_menu>

    </oui:p>
  </oui:card>
</oui:wml>
```

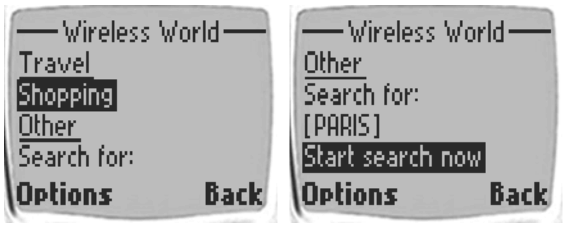
Observe the power of the graphical Openwave Mobile Browser:



The Openwave textual browsers are given a new card. The last menu item points to it.



Nokia browsers support elective forms, so they do not need a new card.



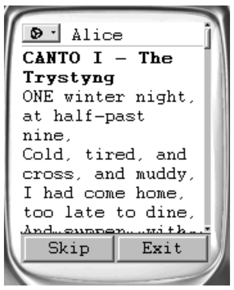
Publishing Large Chunks of Text: BodyPager

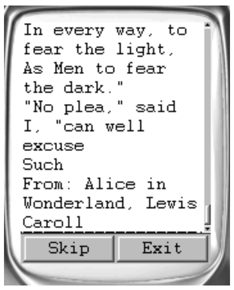
The maximum deck size in Europe is about 1.3 kilobytes. Unfortunately, determining the amount of text that ends up on decks is not always simple. OUI offers a solution through the BodyPager object. In the presence of large amounts of text, the BodyPager splits the content in conveniently sized pages and makes sure that navigation across pages is supported automatically. As an example, here is an excerpt from Lewis Carroll's *Alice In Wonderland*. This example also includes header and footer information, using the `<bp_header>` and `<bp_footer>` tags.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>

<oui:body_pager title="Alice" text_link_forward="Skip" text_link_exit="Exit"
   url_link_exit="main.jsp">

<oui:bp_header>
<oui:p mode="wrap">
<b>CANTO I - The Trystyng</b>
</oui:p>
</oui:bp_header>
ONE winter night, at half-past nine,<br/>
Cold, tired, and cross, and muddy,<br/>
I had come home, too late to dine,<br/>
And supper, with cigars and wine,<br/>
Was waiting in the study.<br/>
<br/><br/>
There was a strangeness in the room,<br/>
And Something white and wavy<br/>
Was standing near me in the gloom -<br/>
I took it for the carpet-broom<br/>
Left by that careless slavey.<br/>
<br/><br/>
But presently the Thing began<br/>
To shiver and to sneeze:<br/>
On which I said "Come, come, my man!<br/>
That's a most inconsiderate plan.<br/>
Less noise there, if you please!"<br/>
<br/><br/>
"I've caught a cold," the Thing replies,<br/>
"Out there upon the landing."<br/>
I turned to look in some surprise,<br/>
And there, before my very eyes,<br/>
A little Ghost was standing!<br/>
<br/><br/>
He trembled when he caught my eye,<br/>
And got behind a chair.<br/>
"How came you here," I said, "and why?<br/>
I never saw a thing so shy.<br/>
Come out!  Don't shiver there!"<br/>
<br/><br/>

...

"ONE slice!  And may I ask you for<br/>
Another drop of gravy?"<br/>
```

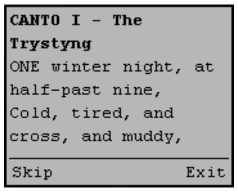
```
I sat and looked at him in awe,<br/>
For certainly I never saw<br/>
A thing so white and wavy.<br/>
<br/><br/>
And still he seemed to grow more white,<br/>
More vapoury, and wavier -<br/>
Seen in the dim and flickering light,<br/>
As he proceeded to recite<br/>
His "Maxims of Behaviour."<br/>
<oui:bp_footer>
<oui:p>
From: Alice in Wonderland, Lewis Carroll
</oui:p>
</oui:bp_footer>
</oui:body_pager>
```

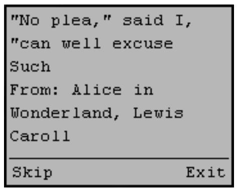
The sample code is rendered like this for graphical Openwave Mobile Browsers:


Alice
CANTO I – The Trystyng
ONE winter night, at half-past nine,
Cold, tired, and cross, and muddy,
I had come home, too late to dine,
Skip Exit

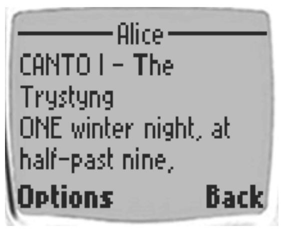

In every way, to fear the light,
As Men to fear the dark."
"No plea," said I, "can well excuse
Such
From: Alice in Wonderland, Lewis Caroll
Skip Exit

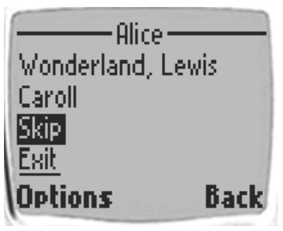
The sample code is rendered like this for text-based Openwave Mobile Browsers:


CANTO I – The Trystyng
ONE winter night, at half-past nine,
Cold, tired, and cross, and muddy,
Skip Exit


"No plea," said I, "can well excuse
Such
From: Alice in Wonderland, Lewis Caroll
Skip Exit

The lack of soft key support on Nokia devices forces users to scroll to the bottom of the page to find navigation.


Alice
CANTO I – The Trystyng
ONE winter night, at half-past nine,
Options Back


Alice
Wonderland, Lewis Caroll
Skip
Exit
Options Back

NOTE Openwave browsers introduce the optimization of preloading the first page, whenever the gateway allows this extension to pass through.

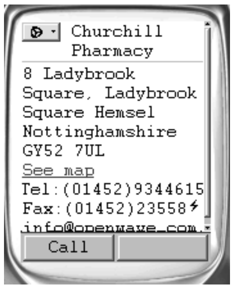
Making Phone Calls: Caller (and Rendering Directives)

The lack of WTAI support in many phones has prevented developers from deploying this useful WAP feature. The OUI library offers an abstraction called Caller. A Caller will resolve into a WTAI call for browsers that support it. Other browsers will receive a new card containing the telephone number on a page of its own. This allows users of non-WTAI compliant devices to start a phone call through proprietary mechanisms (notably, the “use-number” feature on Nokia phones).

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="pharmacy" title="Churchill Pharmacy">
    oui:rendering_directive apply_to="card">
      <oui:method name="enforce_title" />
    </oui:rendering_directive>

      <oui:p>
        8 Ladybrook Square,
        Ladybrook Square
        Hemsel
        Nottinghamshire
        GY52 7UL
          <br/>
        <a href="map.jsp?id=32647287328" title="map">See map</a>
      </p>
      <p mode="nowrap">
        Tel:(01452)9344615582<br/>
        Fax:(01452)2355810158<br/>
        info@openwave.com

    <oui:primary_path short_label="Call">
      <oui:caller phone_number="014529344615582" text="Call Churchill Pharmacy" />
    </oui:primary_path>

      </oui:p>
  </oui:card>
</oui:wml>
```

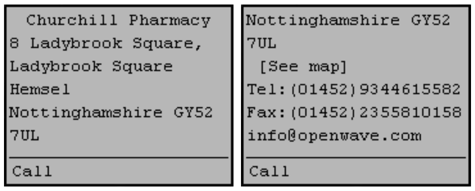
Another OUI feature that you'll find handy is the rendering directive. You can think of a rendering directive like an option to a compiler. You use them to tell OUI to divert from its default behavior in some specific cases. In the code above, you see an example of the `enforce_title` directive. Some browsers honor the value of the `title` attribute on each card. Some don’t. There are cases when you want the title to be displayed anyway. Putting the title in the body of the card is not the best solution, since you will see the title repeated on devices that support real titles. The rendering directive delegates the problem to OUI. In practice, when using the rendering directive, each browser will display the title once (and only once).

The sample code is rendered like this for graphical Openwave Mobile Browsers:



```
Churchill
Pharmacy
8 Ladybrook
Square, Ladybrook
Square Hemsel
Nottinghamshire
GY52 7UL
See map
Tel:(01452)9344615
Fax:(01452)23558
info@openwave.com
Call
```

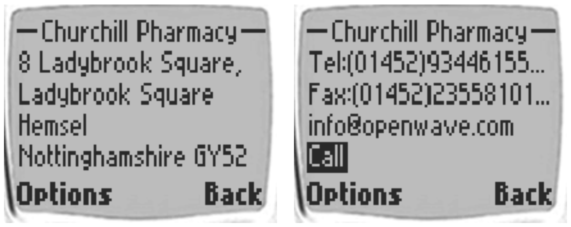
The sample code is rendered like this for text-based Openwave Mobile Browsers:



```
 Churchill Pharmacy
8 Ladybrook Square,
Ladybrook Square
Hemsel
Nottinghamshire GY52
7UL
Call
```

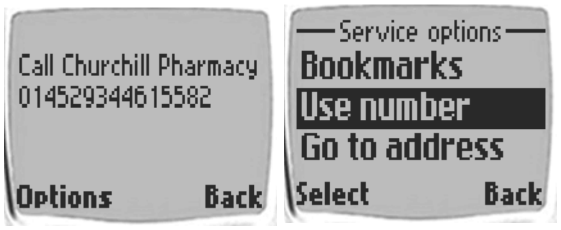
```
Nottinghamshire GY52
7UL
 [See map]
Tel:(01452)9344615582
Fax:(01452)2355810158
info@openwave.com
Call
```

The sample code is rendered like this for Nokia browsers:



```
—Churchill Pharmacy—
8 Ladybrook Square,
Ladybrook Square
Hemsel
Nottinghamshire GY52
Options          Back
```

```
—Churchill Pharmacy—
Tel:(01452)93446155...
Fax:(01452)23558101...
info@openwave.com
Call
Options          Back
```



```
Call Churchill Pharmacy
01452934461 5582
Options          Back
```

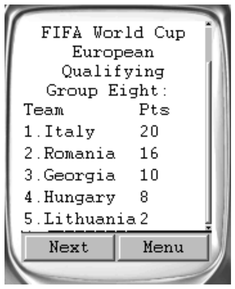
```
—Service options—
Bookmarks
Use number
Go to address
Select           Back
```

WML Tables on All Browsers: Table

Some devices do not properly support WML tables. This has prevented developers from using this valuable feature. OUI supports tables. WML tables are produced for devices that support them. Other devices are presented the same information using a convenient lay-out that does not require the feature tags.

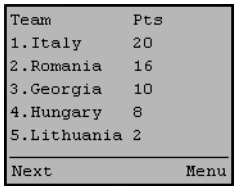
```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="starttable" title="FIFA World Cup">
    <oui:p align="center">
      FIFA World Cup European Qualifying<br/>
      Group Eight:
    </oui:p>
    <oui:p align="left">

    <oui:table logic="row" columns="2">
      <oui:tr>
        <oui:td>Team </oui:td>
        <oui:td>Pts</oui:td>
      </oui:tr>
      <oui:tr>
        <oui:td>1.Italy </oui:td>
        <oui:td>20</oui:td>
      </oui:tr>
      <oui:tr>
        <oui:td>2.Romania</oui:td>
        <oui:td>16</oui:td>
      </oui:tr>
      <oui:tr>
        <oui:td>3.Georgia</oui:td>
        <oui:td>10</oui:td>
      </oui:tr>
      <oui:tr>
        <oui:td>4.Hungary</oui:td>
        <oui:td>8</oui:td>
      </oui:tr>
      oui:tr>
        <oui:td>5.Lithuania</oui:td>
        <oui:td>2</oui:td>
      </oui:tr>
    </oui:table>

    <oui:primary_path href="viewgroup.jsp?id=9" short_label="Next"
      long_label="See Group 9" />
    <oui:secondary_path href="eurozone.jsp" short_label="Up"
      long_label="Europeanzone" />
    <oui:secondary_path href="viewgroup.jsp?id=7" short_label="Back"
      long_label="See Group 7" />

    </oui:p>
  </oui:card>
</oui:wml>
```

The sample code is rendered like this for graphical Openwave Mobile Browsers:



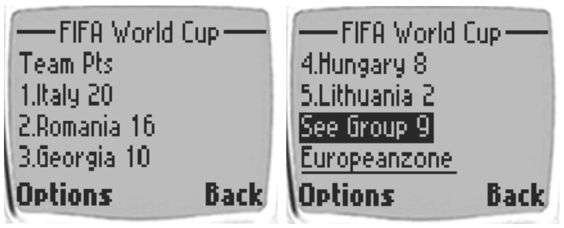
```
  FIFA World Cup
     European
    Qualifying
   Group Eight:
Team          Pts
1.Italy       20
2.Romania     16
3.Georgia     10
4.Hungary     8
5.Lithuania   2
  Next      Menu
```

The sample code is rendered like this for text-based Openwave Mobile Browsers:



```
Team         Pts
1.Italy      20
2.Romania    16
3.Georgia    10
4.Hungary    8
5.Lithuania  2
Next            Menu
```

The sample code is rendered like this for Nokia browsers:



```
—FIFA World Cup—
Team Pts
1.Italy 20
2.Romania 16
3.Georgia 10
Options      Back
```

```
—FIFA World Cup—
4.Hungary 8
5.Lithuania 2
See Group 9
Europeanzone
Options      Back
```

Older Nokia browsers do not support tables.

Reducing Perceived Latency: Mobile-Originated Prefetch

Mobile-originated prefetch is the capability some browsers have to preload one or more decks, under the assumption that users are likely to navigate there soon anyway. This optimization goes a long way into improving application latency, as perceived by the user. Unfortunately, some devices (and gateways) break on this extension. OUI offers a solution by making sure that this feature is automatically discarded for device-gateway combinations that can’t handle it.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>
  <oui:head>
    <oui:prefetch href="nextemailmsbbit.jsp" />
  </oui:head>

  <oui:card id="mop" title="Long Email message">
    <oui:p align="left" mode="nowrap">
      From: Domenico &lt;domenico@openwave.com&gt;<br/>
      To:Luca Passani &lt;luca@openwave.com&gt;<br/>
      Date:Wed,10 Oct 2001<br/>
    </oui:p>
    <oui:p mode="wrap">
      Subject: It will take some time to read the whole message..<br/>

      Hallo Luca, it's time to substantiate your guesses with facts.
      Here is my analysis of the situation. I am sorry
      that this message is a bit long, but woul'll find a lot
      of useful info. So please make yourself
      comfortable and start reading<br/>
      <br/>
...
      <br/>
    <oui:primary_path href="nextemailmsbbit.jsp" short_label="More"
      long_label="24% read. Read more" />

    <oui:task_menu short_label="Menu" long_label="Show Menu">
      <oui:menu_item href="reply.jsp" text="Reply"/>
      <oui:menu_item href="reply.jsp" text="Reply All"/>
      <oui:menu_item href="delete.jsp" text="Delete"/>
      <oui:menu_item href="forwars.jsp" text="Forward"/>
      <oui:menu_item href="viewfold.jsp" text="Move to Folder"/>
      <oui:menu_item href="newfldr.jsp" text="Create New Folder"/>
      <oui:menu_item href="signout.jsp" text="Sign Out"/>
    </oui:task_menu>

    </oui:p>
  </oui:card>
</oui:wml>
```

Prefetch is not a concept you can show with static images—a video might do the job!

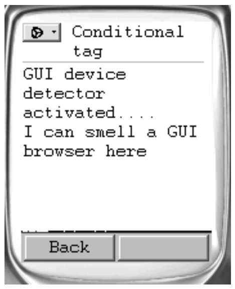
Application-level Optimization: Conditional Tags

OUI performs a lot of optimization behind the scenes. This is core usability support. If you want to use OUI to optimize usability even more, you can do it at the application level. OUI offers a great deal of support for this through conditional tags: `<oui:if>`, `<oui:condition>`, and `<oui:condition_list>`.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="cond" title="Conditional tag">
    <oui:p align="left">
      GUI device detector activated...<br/>

    <oui:if agentfamily="upgui">
      I can smell a GUI browser here
    </oui:if>

    <oui:if agentfamily="upgui" negate="true">
      I cannot smell any GUI browser here
    </oui:if>

</oui:wml>
```

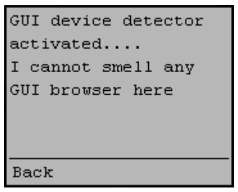
The implication is that you can add or remove specific items for specific devices (or classes of devices), whenever you are not satisfied with the default behavior of the library. You just need a few touches here and there.

The sample code is rendered like this for graphical Openwave Mobile Browsers:



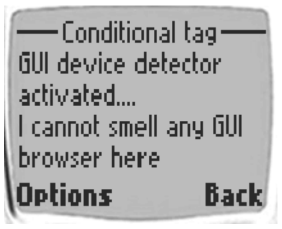
```
Conditional
tag
GUI device
detector
activated....
I can smell a GUI
browser here
```

The sample code is rendered like this for text-based Openwave Mobile Browsers:



```
GUI device detector
activated....
I cannot smell any
GUI browser here
```

The sample code is rendered like this for Nokia browsers:


Conditional tag
GUI device detector activated....
I cannot smell any GUI browser here

Of course, conditional tags come in a more complex format, too:

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="condtag" title="Conditional Tags">
    <oui:p align="left">
      Openwave Detector activated...

      <oui:if>
        <oui:conditionlist logic="AND">
          <oui:condition gateway_vendor="openwave" />
        <oui:conditionlist logic="OR">
          <oui:condition agentfamily="upgui" />
          <oui:condition agentfamily="uptext" />
        </oui:conditionlist>
        </oui:conditionlist>

      <oui:then>
        You have an Openwave browser and you are going through
        an Openwave gateway
      </oui:then>

      <oui:else>
        Either you browser or you gateway are not
        Openwave. It may be both.
      </oui:else>

      </oui:if>

    </oui:p>
  </oui:card>
</oui:wml>
```

Conditions can depend on:

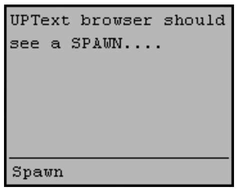
- Gateway vendor name
- User agent string
- Class of device
- Version of the browser
- Any combination of the above

Raw Mode: When All Else Fails

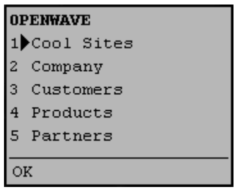
Ideally, OUI contains all you need to build good wireless applications. However, there may be cases where a developer needs to hack bits of mark-up in ways that are not natively supported by the library. In those cases, it is possible to resort to the so-called *raw mode*.

```
<%@ taglib uri="/WEB-INF/tld/oui.tld" prefix="oui" %>
<oui:wml>

  <oui:card id="raw" title="Raw Mode">

    <oui:if agentfamily="upgui">
      <oui:raw_mode element="do">
        <do type="accept" label="Spawn">
          <spawn href="http://www.openwave.com" />
        </do>
      </oui:raw_mode>
    </oui:if>

    <oui:p align="left">
      UPText browser should see a SPAWN....<br/>  </oui:p>
    </oui:card>

</oui:wml>
```

The sample code is rendered like this for text-based Openwave Mobile Browsers:


UPText browser should
see a SPAWN....

Spawn


OPENWAVE
1 Cool Sites
2 Company
3 Customers
4 Products
5 Partners
OK