



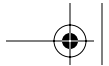
WIRELESS CLIENTS



Topics in This Chapter

- “Rendering Technologies for Mobile Clients” on page 505
- “MIDP Basics” on page 507
- “Mobile Communication and Control Flow” on page 515
- “Component Implementation for Mobile Clients” on page 517
- “The Battleship Game” on page 528





Chapter 11



In recent years, there has been much excitement about accessing web pages through mobile devices such as cell phones or personal digital assistants. These devices have small screens and limited keyboards, making it difficult or impossible to display and navigate regular web pages.

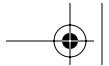
In this chapter, we show you how to write JSF applications that support alternative clients. We focus on cell phones and the Java 2 Platform Micro Edition (J2ME) technology, but the same principles apply to other client technologies.

Rendering Technologies for Mobile Clients

To focus on a specific—and very useful—scenario, let us suppose that we want to deploy a web-based application on a cell phone. Four rendering technologies are supported to various degrees at the time at which we write this book.

1. WAP/WML: The Wireless Access Protocol and the Wireless Markup Language
2. XHTML: The XML representation of HTML, or a subset thereof
3. XUL: The XML User Interface Language
4. J2ME: The Java 2 Micro Edition





WAP is a protocol stack for wireless devices, and WML is a markup language that is loosely equivalent to HTML. However, WML is optimized for cell phones with a small display. WAP/WML enjoys wide device support, but it has had only limited commercial success. WML user interfaces tend to be rather cumbersome and unattractive. We do not discuss WAP/WML in this chapter, but you will find that the techniques that we introduce can be easily modified to render WML.

Mobile phones that render an XHTML subset in “micro browsers” are becoming more common. There is no technical challenge in rendering XHTML for phones in a JSF implementation. You merely need to keep in mind that pages destined for cell phones need to be fairly simple, with small amounts of text and graphics.

XUL is an XML dialect for defining interactive user interfaces. The poster child of XUL is the Mozilla browser (as well as its mail and composer components). Micro implementations of XUL are available on handheld devices, but they are not common. The web site <http://xul.sourceforge.net> contains a good overview of XUL technologies.



NOTE: The JSF reference implementation contains a XUL demo. The purpose of that demo is *not* to show how to deliver an XUL application from a web server. Instead, the demo proves that you can host the JSF engine inside an XUL container, *replacing* the servlet container.

The Java 2 Platform Micro Edition (J2ME) is a version of Java that is optimized for small devices. J2ME affords different flavors for cell phones, cable television boxes, car computers, and so on. Cell phones are supported by the MIDP (Mobile Information Device Profile) library. That library is tailored for devices with limited memory, small screens, and a numeric keypad.

MIDP applications can use the power of Java to produce user interfaces that take optimal advantage of the limited screen size. For example, the “Smart-Ticket” application from the Java blueprint series paints a seating chart for a theater, allowing the customer to pick a seat (see Figure 11–1). It would be difficult to realize such an effective interface on a micro browser.

Network transmission is far slower and more expensive in cell phones than in desktops. For that reason, MIDP applications typically place more application logic onto the client than a browser-based application would. For example, validation and simple navigation logic should be handled on the client.



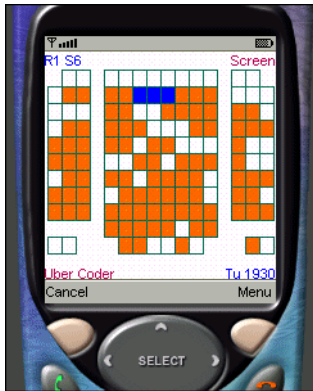
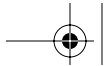


Figure 11-1 The SmartTicket application

In this chapter, we focus on the development of J2ME-based applications. The server-side techniques that are necessary to support these applications can easily be adapted to support other mobile technologies.

MIDP Basics

In this section, we briefly review the building of a MIDP application. For more information, see *Jonathan Knudsen, Wireless Java: Developing with J2ME, Second Edition, Apress 2003*.

Canvases and Forms

MIDP supports a graphic library that is similar to the AWT from the prehistoric time before Swing. That library is even simpler than AWT, and it is intended for small screens and numeric keypads.

A MIDP application extends the `MIDlet` class and minimally extends the three methods `startApp`, `pauseApp`, and `destroyApp`.

The user interface is composed of *displayables*, each of which fills the screen of the device. The most common displayables are the `Canvas` and `Form` classes. A canvas is used for painting of a full-size drawing (such as a theater seating chart or a game board). To create a drawing, subclass `Canvas` and override the `paint` method:

```
public MyCanvas extends Canvas {
    public void paint(Graphics g) {
        g.drawLine(x1, y1, x2, y2);
        ...
    }
    ...
}
```





A form contains a vertically arranged sequence of Item objects. Items are user interface components such as TextField and ChoiceGroup. Figure 11–2 shows a typical form.



Figure 11–2 A MIDP Form

To define a form, simply construct a form object and append items.

```
Form myForm = new Form("Search Flight");
TextField text = new TextField("Airport", "JFK", 3,
    TextField.ANY);
ChoiceGroup choices = new ChoiceGroup("Time", Choice.EXCLUSIVE);
choices.append("am", null /* no image */);
choices.append("pm", null);
myForm.append(text);
myForm.append(choices);
```

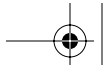
You switch to a particular displayable by calling the setCurrent method of the Display class:

```
public class MyMIDlet extends MIDlet {
    public void startApp() {
        ...
        Display display = Display.getDisplay(this);
        display.setCurrent(myForm);
    }
    ...
}
```

Commands and Keys

To switch between displayables and to initiate network activity, you define commands. A command has a name, a semantic hint that describes the nature





of the command (such as OK, BACK, or EXIT), and a priority. The MIDP environment binds the command to a key or a voice action. Most cell phones have two “soft keys” below the display; these can be mapped to arbitrary commands (see Figure 11–3). If a screen has many commands, the MIDP environment constructs a menu and programs a key to pop up the menu. The priority value gives a hint to the environment whether a command should be bound to an easily accessible key or whether it can be buried inside a menu.



NOTE: The MIDP environment is in charge of binding commands to keys or menus. Using a high priority value does not guarantee that a command is bound to a key.

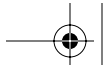


Figure 11–3 The Soft Keys of a Cell Phone

Each displayable needs to specify a `CommandListener`. When the user executes the command, the `commandAction` method of the listener is executed. As with Swing applications, you can specify a separate listener for each command, or you can provide a single listener that dispatches all commands. For simplicity, we do the latter in our sample application.

```
public class MyMIDlet extends MIDlet implements CommandListener {
    private Command nextCommand;
    private Command exitCommand;
    ...
    public void startApp() {
```





```
        nextCommand = new Command("Next", Command.OK, 0);
        exitCommand = new Command("Exit", Command.EXIT, 1);
        myscreen.addCommand(nextCommand);
        myscreen.setCommandListener(this);
        ...
    }

    public void commandAction(Command c, Displayable d) {
        if (c == nextCommand) doNext();
        else if (c == exitCommand) notifyDestroyed();
        else ...
    }

    private void doNext() { ... }
    ...
}
```

In a canvas, you can listen to arbitrary keystrokes. However, not all devices have dedicated cursor keys, so it is best to let the MIDP environment map keys to *game actions* such as LEFT or FIRE.

```
public class MyCanvas extends Canvas {
    ...
    public void keyPressed(int keyCode) {
        int action = getGameAction(keyCode);
        if (action == LEFT) ...
        else ...
    }
}
```

Networking

MIDP contains an `HttpConnection` class that is similar to its J2SE counterpart. This class manages the details of the HTTP protocol, such as request and response headers. Follow these steps to get information from a web server.

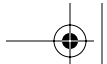
First, you obtain an `HttpConnection` object:

```
HttpConnection conn = (HttpConnection) Connector.open(url);
```

Next, set the request headers:

```
conn.setRequestMethod(HttpConnection.POST);
conn.setRequestProperty("User-Agent",
    "Profile/MIDP-2.0 Configuration/CLDC-1.0");
conn.setRequestProperty("Content-Type",
    "application/x-www-form-urlencoded");
```





Now get the output stream of the connection and send your data to the stream. If you issue a POST command, you need to send URL-encoded name/value pairs. If you issue a GET command, you need not send any data.



CAUTION: If you send POST data to a web server, remember to set the content type to `application/x-www-form-urlencoded`.

Here we read the POST data from a hash table. Unfortunately, MIDP has no built-in method for URL encoding, so we had to write our own. You can find the code in Listing 11–19 at the end of this chapter.

```
Hashtable request = ...;
OutputStream out = conn.openOutputStream();
Enumeration keys = request.keys();
while (keys.hasMoreElements()) {
    String key = (String) keys.nextElement();
    String value = (String) request.get(key);
    urlEncode(key, out);
    out.write('=');
    urlEncode(value, out);
    if (keys.hasMoreElements()) out.write('&');
}
```

Getting the response code automatically closes the output stream.

```
int rc = conn.getResponseCode();
if (rc != HttpURLConnection.HTTP_OK)
    throw new IOException("HTTP response code: " + rc);
```

You can now read the response headers and the server reply.

```
String cookie = conn.getHeaderField("Set-cookie");
int length = conn.getLength();
InputStream in = conn.openInputStream();
byte[] data = new byte[length];
in.read(data, 0, length);
```

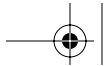
If the server does not report the content length, you will need to manually read until EOF—see the code in Listing 11–1 on page 520.

Close the connection when you are done.

```
conn.close();
```

The principal challenge with MIDP networking is to analyze the response data. If the server sends HTML or XML, the client needs to parse the response. The





current version of MIDP has no support for XML. We examine this issue on page 515.

Multithreading

When a MIDP application makes a network connection, it needs to use a separate thread. Particularly on a cell phone, network connections can be slow and flaky. Moreover, you cannot freeze the user-interface thread, since the UI may pop up a permission dialog when the network connection is about to be initiated (see Figure 11-4).

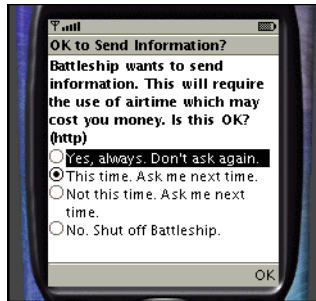


Figure 11-4 Network Permission Dialog

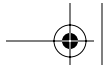
It would be nice if the MIDP library had explicit support for this issue, but unfortunately you need to implement the threading yourself. On a resource-constrained virtual machine, thread creation is expensive, so you want to create one thread for all your connections.

Here is the outline of the threading code:

```
public class MyMIDlet extends MIDlet {
    public void startApp() {
        worker = new ConnectionWorker();
        workerThread = new Thread(worker);
        workerThread.start();
        waitForm = new Form("Waiting...");
        ...
    }
    ...
    public void connect(String url, Hashtable request) {
        display.setCurrent(waitForm);
        worker.connect(url, request);
    }

    public void connectionCompleted(data[] response) {
        // analyze response
    }
}
```





```
        // switch to the next screen
    }
    ...
    private class ConnectionWorker implements Runnable {
        private String url;
        private Hashtable request;
        private byte[] data;
        private boolean busy;

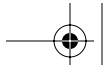
        public synchronized void run() {
            try {
                for (;;) {
                    while (!busy) wait();
                    try { data = post(); }
                    catch (IOException ex) { ... }
                    busy = false;
                    connectionCompleted(data);
                }
            }
            catch (InterruptedException ex) {}
        }

        public synchronized void connect(String url, Hashtable request) {
            this.url = url;
            this.request = request;
            busy = true;
            notify();
        }
        private byte[] post(String url, Hashtable request) { ... }
    }
}
```

The user interface thread calls the connect method, which notifies the worker thread that another connection job is available. As long as there is no possibility that the user interface issues multiple connection commands at the same time, this simple synchronization scheme suffices.

In our sample application, we avoid all the nasty issues of synchronization and cancellation by switching to a wait screen whenever the network connection is in progress. The run method calls the connectionCompleted method of the midlet after the data has been read from the network connection. (We made the connection worker into an inner class to simplify this callback. Note that the callback runs on the worker thread. It should only configure and display the next screen.) For a more sophisticated implementation, with an animated wait screen and an option to cancel the connection, see the excellent article <http://developers.sun.com/techtopics/mobility/midp/articles/threading>.





The MIDP Emulator

Sun Microsystems makes available a convenient toolkit for testing wireless applications (see Figure 11-5). The toolkit includes an environment for compiling and packaging wireless applications, as well as emulators for cell phones and other handheld devices. You can download the wireless toolkit from <http://java.sun.com/products/j2mewtoolkit/>.

Installing and using the toolkit is straightforward. A tutorial is available at <http://developers.sun.com/techtopics/mobility/midp/articles/wtoolkit/>.

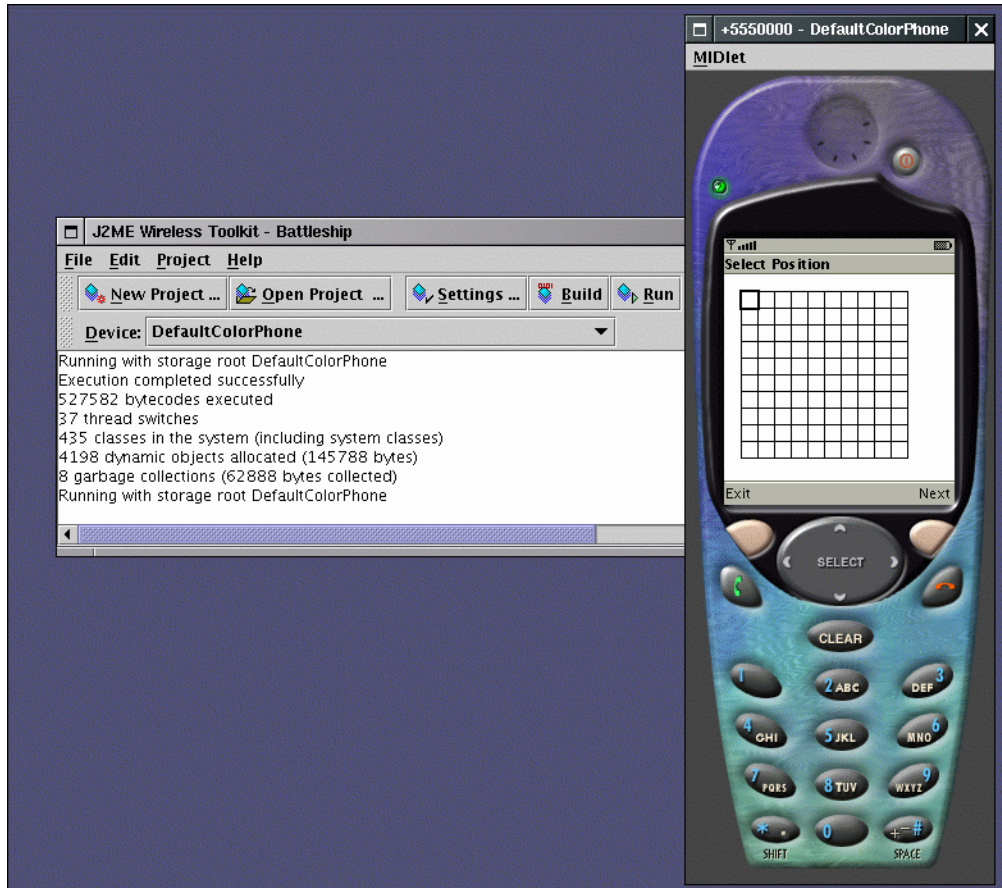
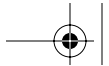


Figure 11-5 Wireless Toolkit





Mobile Communication and Control Flow

Now that you have seen the basics of MIDP programming, let us consider the communication between a MIDP program and a JavaServer Faces application. What data needs to be transferred, and how should the data be formatted?

Let's start at the beginning. The midlet requests a page from the JSF application. Had the request come from a browser, the result would be a page description in HTML. However, a midlet doesn't render HTML. It renders a set of predefined forms and canvases. Thus, the JSF application should tell the midlet

- which form to display next
- how to populate the form items (text fields, choice groups, etc.)

If the next displayable is a canvas, the midlet needs information that is customized for the canvas (such as the seating pattern and the set of available seats for a seating chart).

How should that information be formatted? There are a number of choices.

- An XML document
- A serialized Java object
- An ad hoc format

Unfortunately, the current version of MIDP does not include an XML parser. Lightweight XML parsers that you can bundle with your application are available. A popular choice is the kXML parser, available at <http://kxml.org>. However, parsers with a small footprint tend to have painful APIs, lacking amenities such as XPath. And XML files tend to be verbose, with higher transmission costs than strictly necessary. Many MIDP developers shy away from XML for these reasons.

Communication through Java serialization certainly avoids all parsing overhead since the Java library already knows how to do the parsing. However, the CLDC technology, which underlies MIDP on cell phones, does not currently support serialization. It would also take quite a bit of effort to produce JSF renderers for this purpose.

In our sample application, we use an ad hoc format. We simply send a set of URL encoded name/value pairs, such as

```
form=login  
user=John+0%2e+Public  
password=
```





This format is easily achieved with custom JSF components whose renderers produce the name/value pairs.

For instance, the preceding example is the output of the JSF page

```
<j2me:form id="login">
  <j2me:input id="uname" value="#{user.name}"/>
  <j2me:input id="password" value="#{user.password}"/>
</j2me:form>
```

The indentation in the JSF page produces additional white space that the client must ignore.

Note that the `j2me:form` and `j2me:input` components are not a part of any standard. Later in this chapter we show you how to implement them.



NOTE: You may wonder why we don't simply use the time-honored `Properties` format to encode name/value pairs. The reason is that the MIDP library does not include the `Properties` class, so we would have to produce our own decoder. Moreover, the `Properties` format doesn't escape trailing spaces.

Conversely, the MIDP application needs to send user input to the web application. The easiest method is a simple HTTP POST of URL-encoded form data. A JSF application is prepared to process POST data and to present the request parameters to the various component decoders.

For example, if the user fills in a login screen, the client simply posts data such as

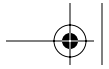
```
form=login&uname=John+Q%2e+Public&password=secret
```

You may wonder why you should bother with JSF when the entire presentation logic appears to be located in the client. Perhaps the client should communicate directly with the server-side business logic, using a protocol such as RMI or SOAP?

Actually, as you will see in our sample application, JSF adds a lot of value. JSF has a well-developed model for the binding between presentation and business logic. It is better to reuse this model than to try to reinvent it. Moreover, many navigation decisions are best handled on the server. For example, if a customer places an order and an item is out of stock, the server is best equipped to determine the impact on navigation. JSF has a good navigation model, so why not take advantage of it?

Of course, some of the presentation logic is best handled by the client. Simple validation and navigation between data entry screens need not involve the server at all. This is not a problem. The midlet can gather user data, using as





many screens and validation steps as necessary, and then post all data to the server in one step.

In summary, here are the main points to keep in mind when you are developing a MIDP application with JSF.

- The midlet draws predefined screens rather than rendering arbitrary markup.
- As with HTML rendering, the JSF components are proxies for the client-side components.
- The controller is split between the midlet and the JSF application. Fine-grained navigation and validation decisions are handled by the midlet, and all other presentation logic is handled by the JSF application in the usual way.

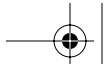
Component Implementation for Mobile Clients

The JSF specification specifies a standard set of presentation-neutral components such as `UIInput`, `UIForm`, and `UICommand`, and an HTML-specific set of tags and renderers. There is no standard implementation for non-HTML clients. Thus, we need to develop our own set of tags and renderers to communicate with a midlet.

Let us first review what exactly happens when you put an HTML component tag such as `h:inputText` into your JSF page.

- The standard tag library file `html_basic.tld` maps the `inputText` tag to the class `com.sun.faces.taglib.html_basic.InputTextTag`. The tag handler class is called whenever the JSF file parser encounters an `inputText` tag. The tag handler stashes away the tag attributes (such as `value` and `validator`) inside the associated component object.
- The `getComponentType` method of that class returns the string `"javax.faces.HtmlInputText"`, and the `getRendererType` method returns the string `"javax.faces.Text"`.
- The file `jsf-ri-config.xml` maps these strings to classes `javax.faces.component.html.HtmlInputText` and `com.sun.faces.renderkit.html_basic.TextRenderer`.
- The `HtmlInputText` class is a subclass of `UIInput`. It merely adds getter and setter methods for the "pass through" attributes (such as `onmouseover`, `alt`, and so on). The `UIInput` class carries out validation, manages value change events, and updates model values.
- The `com.sun.faces.renderkit.html_basic.TextRenderer` class does a fair amount of heavy lifting. Its `decode` method fetches the value that the user supplied





and sets it as the current value of the component. Its encode methods produce a string of the form `<input type="text" .../>`. Moreover, these methods handle data conversion and value binding lookup.

When we implement our own components, we can reuse component classes such as `UIInput`. However, the JSF framework gives very limited support for tags and renderers. Much of the essential work is carried out by classes in the `com.sun.faces` packages. We need to replicate that work in our own tags and renderers.

We designed a tag library with five tags: `input`, `select`, `output`, `form`, and `command`. On the JSF side, they correspond to the `UIInput`, `UISelectOne`, `UIOutput`, `UIForm`, and `UICommand` classes. On the MIDP side, the first two correspond to `TextField` and `ChoiceGroup` items. The `form` tag identifies the form that the client should display. The client uses the `command` tag to invoke JSF actions.

Let's walk through the `input` tag in detail. Consider the following tag:

```
<j2me:input id="uname" value="#{user.name}"
  validator="#{loginform.authenticate}"/>
```

The tag class for the `input` tag processes the `id`, `value`, and `validator` attributes and set the appropriate values in the `UIInput` component.

When the client requests this page for the first time, the renderer for this tag produces a string such as

```
uname=John+Q%2e+Public
```

The renderer simply looks up the ID and the value of the `UIInput` component. When the MIDP client receives this string, it places the value of the `uname` key inside the matching text field. Note again that the client does its own rendering. The server doesn't tell it where to place the text field, only what value it should contain.

When the client posts the form contents to the server, it sends POST data that contain the new value of the text field, such as

```
...&uname=Jane+Doe&...
```

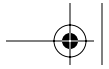
Now the renderer for the `input` tag kicks in again, fetching the request parameter value and setting the value of the `UIInput` component.

We don't need to worry about invoking the validator or evaluating the value binding expression—those tasks are handled by the JSF framework.

Now let's look at the code in detail.

As always, we start with a tag library descriptor file (see Listing 11-1). That file lists the tags, their valid attributes, and the handler classes.





For simplicity, all tag handlers extend the `J2meComponentTag` class shown in Listing 11–3. That class processes the value, action, and validator attributes. The id and binding attributes are handled by the `UIComponentTag` superclass.

The `InputTag` class (Listing 11–4) merely defines the `getComponentType` and `getRendererType` methods. The first method returns the string "Input", which is mapped by the standard JSF configuration to the `UIInput` component. The second method returns the string `J2meText`, which we will map to the `TextRenderer` class of Listing 11–5.

The `decode` method of the renderer simply sets the new value of the component:

```
Map requestMap = context.getExternalContext()
    .getRequestParameterMap();
if (requestMap.containsKey(id)) {
    String newValue = (String) requestMap.get(id);
    ((ValueHolder) component).setValue(newValue);
}
```

The `encodeBegin` method writes out the current value:

```
ResponseWriter writer = context.getResponseWriter();
String id = component.getId();
String value = ((ValueHolder) component).getValue().toString();
writer.write(id + "=" + URLEncoder.encode(value, "UTF8") + "\n");
```

Moreover, the `encodeBegin` method produces a name/value pair with all messages that may have queued up for this component. For example, if during validation, an error was found in the `uname` field, the resulting output would be

```
uname.messages=No+such+user
```

It is up to the client to decide what to do with these messages.

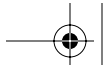
The remaining component implementations are similar. A few issues are worth noting.

- The renderer for the `command` tag queues an action event instead of setting a component value. The associated `UICommand` component automatically processes the event, using its `action` property. See Listing 11–6.
- The renderer for the `selectOne` tag (shown in Listing 11–7) encodes the labels for the choices in the following format:

```
direction.label.0=horizontal
direction.label.1=vertical
```

It needs to carry out this in the `encodeEnd` method rather than `encodeBegin` since the enclosed `f:selectItem` or `f:selectItems` values are not available at the beginning!





- Finally, the form renderer needs to call the `setSubmitted` method of the `UIForm` and to indicate whether the form is requested for the first time or whether it is posted again with new values. (See Listing 11-8.) There is a technical reason for this requirement. When the form is rendered for the first time, the component values may be defaults that do not pass validation. Validation should only occur when the renderer processes form data that are posted from the client. The `submitted` property regulates this behavior. We simply require the client to include the form ID when posting form data.

As you can see, implementing these renderers is fairly straightforward. Moreover, you gain some insight into the rendering process that is helpful for understanding any JSF application.

To separate the configuration information for these renderers from the application-specific configuration, we use an auxiliary file `j2me-config.xml` (see Listing 11-2). To add this file to your web application, include the following parameter definition in your `web.xml` file:

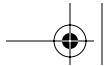
```
<context-param>
  <param-name>
    javax.faces.application.CONFIG_FILES</param-name>
  <param-value>
    /WEB-INF/faces-config.xml,
    /WEB-INF/j2me-config.xml
  </param-value>
</context-param>
```

In the next section, we put our tags to work in a complete application.

Listing 11-1 phonebattle/WEB-INF/j2me.tld

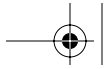
```
1. <?xml version="1.0" encoding="ISO-8859-1" ?>
2. <!DOCTYPE taglib
3. PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
4. "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">
5. <taglib>
6.   <tlib-version>0.03</tlib-version>
7.   <jsp-version>1.2</jsp-version>
8.   <short-name>j2me</short-name>
9.   <uri>http://corejsf.com/j2me</uri>
10.  <description>
11.    This tag library contains J2ME component tags.
12.  </description>
13.
```



**Listing 11-1** phonebattle/WEB-INF/j2me.tld (cont.)

```
14. <tag>
15.   <name>form</name>
16.   <tag-class>com.corejsf.j2me.FormTag</tag-class>
17.   <attribute>
18.     <name>id</name>
19.   </attribute>
20. </tag>
21. <tag>
22.   <name>input</name>
23.   <tag-class>com.corejsf.j2me.InputTag</tag-class>
24.   <attribute>
25.     <name>id</name>
26.   </attribute>
27.   <attribute>
28.     <name>value</name>
29.   </attribute>
30.   <attribute>
31.     <name>validator</name>
32.   </attribute>
33. </tag>
34. <tag>
35.   <name>output</name>
36.   <tag-class>com.corejsf.j2me.OutputTag</tag-class>
37.   <attribute>
38.     <name>id</name>
39.   </attribute>
40.   <attribute>
41.     <name>value</name>
42.   </attribute>
43. </tag>
44. <tag>
45.   <name>selectOne</name>
46.   <tag-class>com.corejsf.j2me.SelectOneTag</tag-class>
47.   <attribute>
48.     <name>id</name>
49.   </attribute>
50.   <attribute>
51.     <name>binding</name>
52.   </attribute>
53.   <attribute>
54.     <name>value</name>
55.   </attribute>
56. </tag>
57. <tag>
```

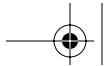


**Listing 11-1** phonebattle/WEB-INF/j2me.tld (cont.)

```
59.     <name>command</name>
60.     <tag-class>com.corejsf.j2me.CommandTag</tag-class>
61.     <attribute>
62.         <name>id</name>
63.     </attribute>
64.     <attribute>
65.         <name>action</name>
66.     </attribute>
67. </tag>
1. </taglib>
```

Listing 11-2 phonebattle/WEB-INF/j2me-config.xml

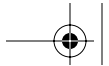
```
1. <?xml version="1.0"?>
2.
3. <!DOCTYPE faces-config PUBLIC
4.     "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
5.     "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
6.
7. <faces-config>
8.     <render-kit>
9.         <renderer>
10.            <component-family>javax.faces.Input</component-family>
11.            <renderer-type>com.corejsf.j2me.Text</renderer-type>
12.            <renderer-class>com.corejsf.j2me.TextRenderer</renderer-class>
13.        </renderer>
14.        <renderer>
15.            <component-family>javax.faces.Output</component-family>
16.            <renderer-type>com.corejsf.j2me.Text</renderer-type>
17.            <renderer-class>com.corejsf.j2me.TextRenderer</renderer-class>
18.        </renderer>
19.        <renderer>
20.            <component-family>javax.faces.Form</component-family>
21.            <renderer-type>com.corejsf.j2me.Form</renderer-type>
22.            <renderer-class>com.corejsf.j2me.FormRenderer</renderer-class>
23.        </renderer>
24.        <renderer>
25.            <component-family>javax.faces.SelectOne</component-family>
26.            <renderer-type>com.corejsf.j2me.Choice</renderer-type>
27.            <renderer-class>com.corejsf.j2me.ChoiceRenderer</renderer-class>
28.        </renderer>
29.        <renderer>
30.            <component-family>javax.faces.Command</component-family>
31.            <renderer-type>com.corejsf.j2me.Command</renderer-type>
32.            <renderer-class>com.corejsf.j2me.CommandRenderer</renderer-class>
33.        </renderer>
34.    </render-kit>
35. </faces-config>
```

**Listing 11-3** phonebattle/WEB-INF/classes/com/corejsf/j2me/J2meComponentTag.java

```
1. package com.corejsf.j2me;
2.
3. import javax.faces.component.UIComponent;
4. import javax.faces.webapp.UIComponentTag;
5.
6. public abstract class J2meComponentTag extends UIComponentTag {
7.     private String value;
8.     private String action;
9.     private String validator;
10.
11.     // PROPERTY: value
12.     public void setValue(String newValue) { value = newValue; }
13.
14.     // PROPERTY: action
15.     public void setAction(String newValue) { action = newValue; }
16.
17.     // PROPERTY: validator
18.     public void setValidator(String newValue) { validator = newValue; }
19.
20.     public void setProperties(UIComponent component) {
21.         super.setProperties(component);
22.         com.corejsf.util.Tags.setString(component, "value", value);
23.         com.corejsf.util.Tags.setAction(component, action);
24.         com.corejsf.util.Tags.setValidator(component, validator);
25.     }
26.
27.     public void release() {
28.         value = null;
29.         validator = null;
30.         action = null;
31.     }
32. }
```

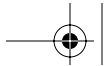
Listing 11-4 phonebattle/WEB-INF/classes/com/corejsf/j2me/InputTag.java

```
1. package com.corejsf.j2me;
2.
3.
4. public class InputTag extends J2meComponentTag {
5.     public String getComponentType() { return "javax.faces.Input"; }
6.     public String getRendererType() { return "com.corejsf.j2me.Text"; }
7. }
```

**Listing 11-5** phonebattle/WEB-INF/classes/com/corejsf/j2me/TextRenderer.java

```
1. package com.corejsf.j2me;
2.
3. import java.io.IOException;
4. import java.net.URLEncoder;
5. import java.util.Map;
6. import javax.faces.component.UIComponent;
7. import javax.faces.component.ValueHolder;
8. import javax.faces.context.FacesContext;
9. import javax.faces.context.ResponseWriter;
10. import javax.faces.render.Renderer;
11.
12. public class TextRenderer extends Renderer {
13.     public void encodeBegin(FacesContext context, UIComponent component)
14.         throws IOException {
15.         ResponseWriter writer = context.getResponseWriter();
16.         String id = component.getId();
17.         String value = "" + ((ValueHolder) component).getValue();
18.         writer.write(id + "=" + URLEncoder.encode(value, "UTF8") + "\n");
19.     }
20.
21.     public void decode(FacesContext context, UIComponent component) {
22.         if (context == null || component == null) return;
23.
24.         String id = component.getId();
25.         Map requestMap
26.             = context.getExternalContext().getRequestParameterMap();
27.         if (requestMap.containsKey(id)
28.             && component instanceof ValueHolder) {
29.             String newValue = (String) requestMap.get(id);
30.             ((ValueHolder) component).setValue(newValue);
31.         }
32.     }
33. }
```



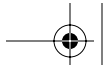
**Listing 11-6**phonebattle/WEB-INF/classes/com/corejsf/j2me/
CommandRenderer.java

```
1. package com.corejsf.j2me;
2.
3. import java.util.Map;
4. import javax.faces.component.UIComponent;
5. import javax.faces.context.FacesContext;
6. import javax.faces.event.ActionEvent;
7. import javax.faces.render.Renderer;
8.
9. public class CommandRenderer extends Renderer {
10.     public void decode(FacesContext context, UIComponent component) {
11.         if (context == null || component == null) return;
12.
13.         String id = component.getId();
14.         Map requestMap
15.             = context.getExternalContext().getRequestParameterMap();
16.         if (requestMap.containsKey(id)) {
17.             component.queueEvent(new ActionEvent(component));
18.         }
19.     }
20. }
```

Listing 11-7

phonebattle/WEB-INF/classes/com/corejsf/j2me/ChoiceRenderer.java

```
1. package com.corejsf.j2me;
2.
3. import java.io.IOException;
4. import java.net.URLEncoder;
5. import java.util.List;
6. import java.util.Map;
7. import javax.faces.component.UIComponent;
8. import javax.faces.component.EditableValueHolder;
9. import javax.faces.component.ValueHolder;
10. import javax.faces.context.FacesContext;
11. import javax.faces.context.ResponseWriter;
12. import javax.faces.model.SelectItem;
13. import javax.faces.render.Renderer;
14.
15. public class ChoiceRenderer extends Renderer {
16.     public void encodeEnd(FacesContext context, UIComponent component)
17.         throws IOException {
18.         ResponseWriter writer = context.getResponseWriter();
19.         EditableValueHolder input = (EditableValueHolder) component;
```

**Listing 11-7**phonebattle/WEB-INF/classes/com/corejsf/j2me/ChoiceRenderer.java
(cont.)

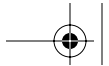
```
20.     String id = component.getId();
21.     List items = com.corejsf.util.Renderers.getSelectItems(component);
22.     String value = input.getValue().toString();
23.     String label = findLabel(items, value);
24.     writer.write(id + "=" + URLEncoder.encode(label, "UTF8") + "\n");
25.     for (int i = 0; i < items.size(); i++) {
26.         SelectItem item = (SelectItem) items.get(i);
27.         writer.write(id + ".label." + i
28.             + "=" + URLEncoder.encode(item.getLabel(), "UTF8") + "\n");
29.     }
30. }
31.
32. public void decode(FacesContext context, UIComponent component) {
33.     if (context == null || component == null) return;
34.
35.     String id = component.getId();
36.     Map requestMap
37.         = context.getExternalContext().getRequestParameterMap();
38.     if (requestMap.containsKey(id)
39.         && component instanceof ValueHolder) {
40.         String label = (String) requestMap.get(id);
41.         List items = com.corejsf.util.Renderers.getSelectItems(component);
42.         Object value = findValue(items, label);
43.         ((ValueHolder) component).setValue(value);
44.     }
45. }
46.
47. private static Object findValue(List list, String label) {
48.     for (int i = 0; i < list.size(); i++) {
49.         SelectItem item = (SelectItem) list.get(i);
50.         if (item.getLabel().equals(label)) return item.getValue();
51.     }
52.     return null;
53. }
54.
55. private static String findLabel(List list, Object value) {
56.     for (int i = 0; i < list.size(); i++) {
57.         SelectItem item = (SelectItem) list.get(i);
58.         if (item.getValue().equals(value)) return item.getLabel();
59.     }
60.     return null;
61. }
62. }
```



**Listing 11-8** phonebattle/WEB-INF/classes/com/corejsf/j2me/FormRenderer.java

```
63. package com.corejsf.j2me;
64.
65. import java.io.IOException;
66. import java.net.URLEncoder;
67. import java.util.Iterator;
68. import java.util.Map;
69. import javax.faces.application.FacesMessage;
70. import javax.faces.component.UIComponent;
71. import javax.faces.component.UIForm;
72. import javax.faces.context.FacesContext;
73. import javax.faces.context.ResponseWriter;
74. import javax.faces.render.Renderer;
75.
76. public class FormRenderer extends Renderer {
77.     public void encodeBegin(FacesContext context,
78.         UIComponent component) throws IOException {
79.         ResponseWriter writer = context.getResponseWriter();
80.         writer.write("form=" + component.getId() + "\n");
81.
82.         Iterator ids = context.getClientIdsWithMessages();
83.         while (ids.hasNext()) {
84.             String id = (String) ids.next();
85.             Iterator messages = context.getMessages(id);
86.             String msg = null;
87.             while (messages.hasNext()) {
88.                 FacesMessage m = (FacesMessage) messages.next();
89.                 if (msg == null) msg = m.getSummary();
90.                 else msg = msg + "," + m.getSummary();
91.             }
92.             if (msg != null) {
93.                 writer.write("messages");
94.                 if (id != null) writer.write("." + id);
95.                 writer.write("=" + URLEncoder.encode(msg, "UTF8") + "\n");
96.             }
97.         }
98.     }
99.
100.    public void decode(FacesContext context, UIComponent component) {
101.        Map map = context.getExternalContext().getRequestParameterMap();
102.        ((UIForm)component).setSubmitted(
103.            component.getId().equals(map.get("form")));
104.    }
```





The Battleship Game

In this section, we put together a simple wireless application that plays the Battleship game against a simulated opponent on a server. We chose this game because it demonstrates a mix of text and graphical components on the handheld device. As a practical matter, customers would probably not want to pay for airtime to play such a simple game against a remote computer. However, with the same implementation techniques, you can implement more complex games and business applications with rich user interfaces.

The Game Rules

In Battleship, each player has a battleground—a rectangular grid. At the start of the game, the players arrange sets of ships on their battlegrounds. In the United States, the traditional arrangement is a 10 by 10 grid, with one ship each of sizes 2, 4, and 5, and two ships of size 3. Players do not see each other's battleground. Players take turns, firing at each other's battlegrounds (see Figure 11-6). The other player announces whether a shot hits a ship or misses. The first player who sinks all ships of the opponent wins the game.

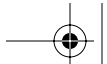


Figure 11-6 The Battleship Game

The User Interface

In the setup phase, the player selects the starting position of the next ship, then moves to a form for choosing the ship's direction and size (see Figure 11-7). Clicking the soft button labeled "Add" sends the form data to the server (see Figure 11-8). The process repeats until all ships are placed.





NOTE: It would have been nicer to put the direction and size items on the same screen as the grid. This is possible in MIDP 2.0, by implementing a CustomItem. For simplicity, we do not use this technique.

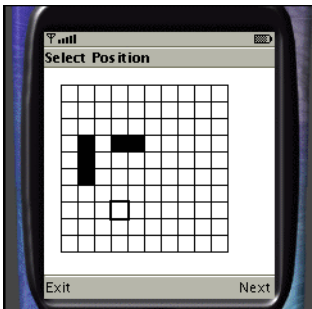


Figure 11-7 Selecting the Ship Position



Figure 11-8 Adding a Ship

In each turn of the battle, the player first sees the damage done by the opponent and then gets to fire a shot by selecting the target position and pressing the soft button labeled “Fire” (see Figure 11-9). When the game is over, the winner is announced (see Figure 11-10).

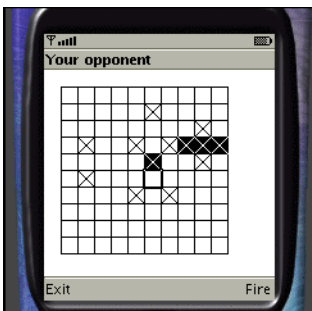


Figure 11-9 Firing a Shot

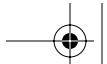


Figure 11-10 Winning the Game

Implementation

To implement this game, we use two standard components on the server side: `UIInput` (for the position) and `UISelectOne` (for the direction and size). We also use





a custom `BattleGround` component, for displaying boats and firing shots. A `GameBean` manages the battlegrounds of the two opponents, and a `SetupForm` coordinates the components used during game setup. The `SetupForm` validates the placement of a new boat, and it determines when all boats have been placed. We do not discuss the code of these classes in detail; much of it is concerned with the tedium of arranging the boats and playing the game. See Listings 11–14 through 11–18 at the end of this section. Figure 11–11 shows the files that make up the application.

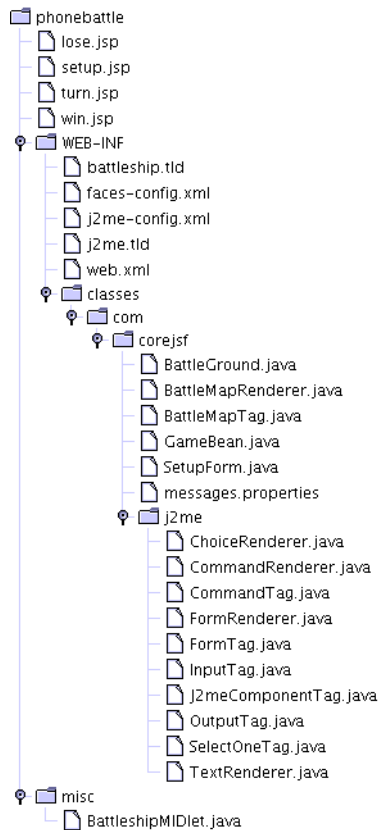
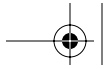


Figure 11–11 Files of the Wireless Battleship Program

Let us turn to the interesting part of the implementation, starting with the first page, `setup.jsp` (Listing 11–9). The page contains four components: a map of the battleground, a text field for the position of the next boat, and two selection components for the boat’s size and position.



**Listing 11-9** phonebattle/setup.jsp

```

1. <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2. <%@ taglib uri="http://corejsf.com/j2me" prefix="j2me" %>
3. <%@ taglib uri="http://corejsf.com/battleship" prefix="battleship" %>
4. <f:view>
5.   <j2me:form id="setup">
6.     <battleship:map id="own" value="#{game.own}" own="true"
7.       validator="#{setupform.validate}"/>
8.     <j2me:selectOne id="direction"
9.       binding="#{setupform.directionComponent}"
10.      value="#{setupform.horizontal}">
11.       <f:selectItem itemValue="true" itemLabel="horizontal"/>
12.       <f:selectItem itemValue="false" itemLabel="vertical"/>
13.     </j2me:selectOne>
14.     <j2me:selectOne id="size"
15.       binding="#{setupform.sizeComponent}"
16.       value="#{setupform.size}">
17.       <f:selectItems value="#{game.own.availableSizes}"/>
18.     </j2me:selectOne>
19.     <j2me:command id="submit" action="#{setupform.submitAction}"/>
20.   </j2me:form>
21. </f:view>

```

Note that the client UI shows this information on two separate screens.

Also note that the two selection components are managed by the SetupForm class. This arrangement simplifies the implementation of the validate method. (Since the validator is attached to the text input component, it is passed as a parameter to the validate call.)

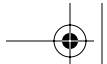
We discussed the renderers for the standard components in the preceding section. Let us briefly look at the renderer for the battleground (Listing 11-18 on page 546). The encodeBegin method produces a string that describes the battle map. The own property determines whether to include boats that haven't yet been hit. They are shown only if own is true. Each position is encoded as one of the following values:

- 0 = water, not hit, or unknown if not owner
- 1 = ship, not hit
- 2 = water, hit
- 3 = ship, hit

Rows are separated by URL-encoded spaces. A typical rendering result looks like this:

```
own=0010000000+0010000000+000001100+...+0000000000
```





When the user selects a position, the client sends back a request value of the form

```
...&opponent=C4&...
```

The `decode` method calls the `setCurrent` method of the `battleground` and fires an action event. In the setup phase, the event handler adds a boat at the specified position. In the play phase, the event handler fires a shot at that position.

When all boats have been added, the navigation handler selects the `turn.jsp` page (Listing 11–10). It simply shows both boards. The action event handler of the second board triggers the `game.move` method. That method fires on the opponent's board, lets the opponent fire on the player's board, and checks whether either contender has won the game.

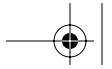
Listing 11–10 phonebattle/turn.jsp

```
1. <%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
2. <%@ taglib uri="http://corejsf.com/j2me" prefix="j2me" %>
3. <%@ taglib uri="http://corejsf.com/battleship" prefix="battleship" %>
4. <f:view>
5.   <j2me:form id="turn">
6.     <battleship:map id="own" value="#{game.own}" own="true"/>
7.     <battleship:map id="opponent" value="#{game.opponent}" own="false"/>
8.     <j2me:command id="fire" action="#{game.move}"/>
9.   </j2me:form>
10. </f:view>
```

If a player has won, then either the `win.jsp` or `lose.jsp` page is displayed. (See Listings 11–11 and 11–12 for these pages.) Note that these pages, unlike all other pages of this application, actually render some contents. They set a value for the `result` key, which is displayed by the client.

Listing 11–11 phonebattle/win.jsp

```
1. <%@ taglib uri="http://java.sun.com/jsp/core" prefix="f" %>
2. <%@ taglib uri="http://corejsf.com/j2me" prefix="j2me" %>
3. <f:view>
4.   <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
5.   <j2me:form id="win">
6.     <j2me:output id="result" value="#{msgs.youWon}"/>
7.     <j2me:command id="newgame" action="#{game.initialize}"/>
8.   </j2me:form>
9. </f:view>
```



Listing 11-12 phonebattle/lose.jsp

```

1. <%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
2. <%@ taglib uri="http://corejsf.com/j2me" prefix="j2me" %>
3. <f:view>
4.   <f:loadBundle basename="com.corejsf.messages" var="msgs"/>
5.   <j2me:form id="lose">
6.     <j2me:output id="result" value="#{msgs.youLost}"/>
7.     <j2me:command id="newgame" action="#{game.initialize}"/>
8.   </j2me:form>
9. </f:view>

```

Listing 11-13 contains the navigation handler; see Figure 11-12 for the navigation map.

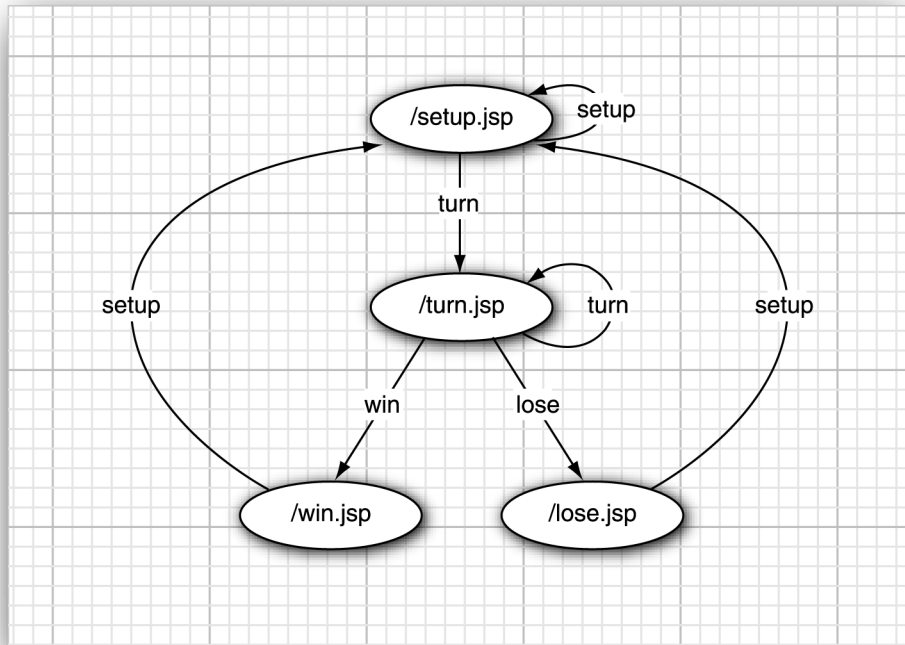
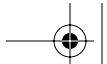


Figure 11-12 Navigation Map of the Battleship Application

The client is shown in Listing 11-19. Its implementation is fairly straightforward. From the perspective of interfacing a midlet with JSF, the most important piece is the handling of the server response. The `connectionCompleted` method is





called after the server response has been decoded and placed in a hash table. The value of the form key determines the next form to be displayed.

```
public void connectionCompleted(Hashtable response) {
    webform = (String) response.get("form");
    if (webform.equals("add")) showAdd(response);
    else if (webform.equals("turn")) showTurn(response);
    else if (webform.equals("win")) showGameOver(response);
    else if (webform.equals("lose")) showGameOver(response);
}
```

The `connectionCompleted` method simply branches to a separate method for each form. Each method is responsible for applying the response parameters and switching the display. For example, here is the `showGameOver` method:

```
public void showGameOver(Hashtable response) {
    result.setText((String) response.get("result"));
    display.setCurrent(gameOverForm);
}
```

Conversely, when the server is contacted, the contents of the various components are packaged in a request data table. For example, the following method is called when the user adds a new boat.

```
public void doAdd() {
    Hashtable request = new Hashtable();
    request.put("size", size.getString(size.getSelectedIndex()));
    request.put("direction",
        direction.getString(direction.getSelectedIndex()));
    request.put("position", position.getString());
    request.put("form", "setup");
    request.put("submit", "");
    connect("setup.jsp", request);
}
```

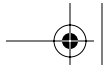
Note the form and submit parameters. The form parameter is required so that the `FormRenderer` treats the submitted values as updates to an existing form, as discussed in the preceding section. The submit parameter must be present so that the renderer of the server-side `UICommand` component fires an action event.



TIP: It can be frustrating to debug the server-side portion of a mobile application through the phone emulator. We found it much easier to use a browser (see Figure 11–13). Simply make a GET request (such as `http://localhost:8080/phonebattle/setup.faces?form=setup&direction=vertical&own=A4&size=3&submit=`) and see the server response (or, sadly, more often than not, a stack trace).

As you can see, it is reasonably simple to combine the MIDP and JSF technologies to develop mobile applications. Here is the approach that we have taken in this chapter:





- Use HTTP POST to send data from a midlet to the web server.
- Encode the navigation and rendering information in URL-encoded name/value pairs instead of HTML.
- Provide a simple JSF tag library that renders the standard JSF components in this format.

We hope that someday a standard mechanism will be provided for sending data from a JSF application to a midlet.

Once the plumbing is in place, there is very little difference between wireless and browser clients on the JSF side. Of course, the client midlet needs to render the UI, but that's the MIDP way.



NOTE: It is a simple matter to reimplement the battleship game for a browser client (see Figure 11–14). All the “business logic” in the form and game bean is unchanged. The only real work is to implement an HTML renderer for the battle map. You can find the code for this application on the web site for this book.

Given the similarities between the phone and web versions, you may wonder whether you could produce a unified version of the code and automatically switch renderers, depending on the User-agent field of the request. We don't think that is very practical. Page layout is always going to be different for different devices.

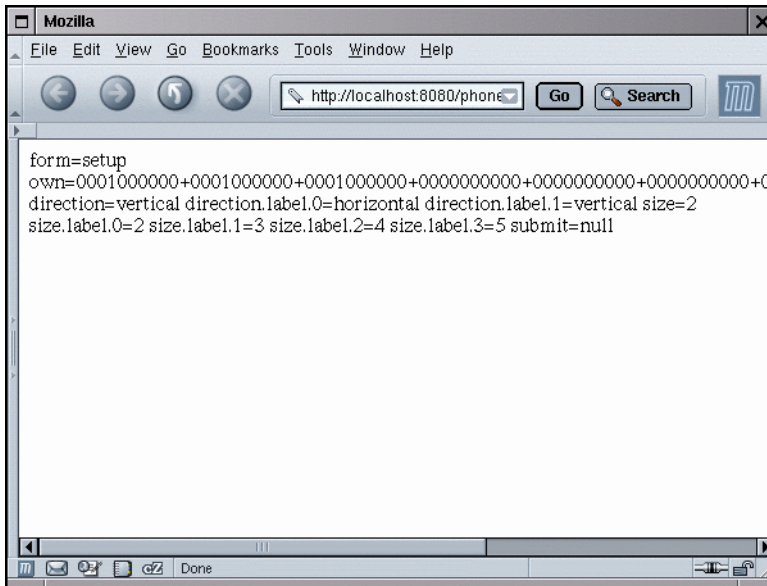


Figure 11–13 Debugging a Wireless Application with a Browser



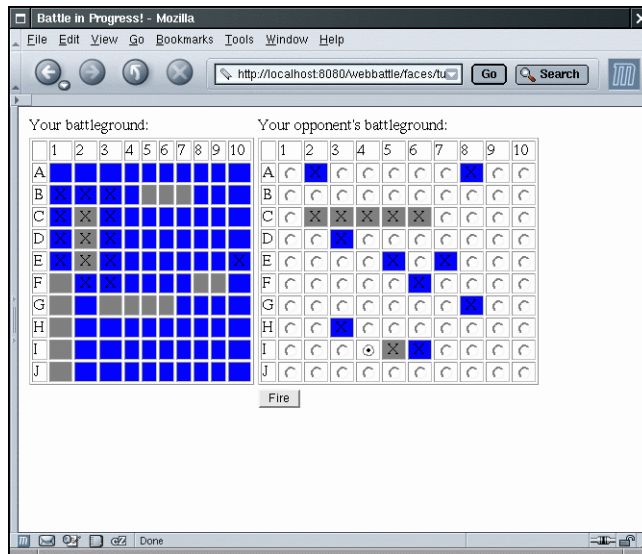
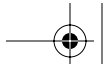


Figure 11-14 A Browser-Based Battle Game

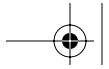
Listing 11-13 phonebattle/WEB-INF/faces-config.xml

```

1. <?xml version="1.0"?>
2.
3. <!DOCTYPE faces-config PUBLIC
4.   "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN"
5.   "http://java.sun.com/dtd/web-facesconfig_1_0.dtd">
6.
7. <faces-config>
8.
9.   <navigation-rule>
10.    <from-view-id>/setup.jsp</from-view-id>
11.    <navigation-case>
12.      <from-outcome>setup</from-outcome>
13.      <to-view-id>/setup.jsp</to-view-id>
14.    </navigation-case>
15.    <navigation-case>
16.      <from-outcome>turn</from-outcome>
17.      <to-view-id>/turn.jsp</to-view-id>
18.    </navigation-case>
19.  </navigation-rule>
20.  <navigation-rule>
21.    <from-view-id>/turn.jsp</from-view-id>
22.    <navigation-case>
23.      <from-outcome>turn</from-outcome>
24.      <to-view-id>/turn.jsp</to-view-id>

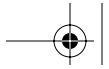
```



**Listing 11-13** phonebattle/WEB-INF/faces-config.xml (cont.)

```
25.     </navigation-case>
26.     <navigation-case>
27.         <from-outcome>win</from-outcome>
28.         <to-view-id>/win.jsp</to-view-id>
29.     </navigation-case>
30.     <navigation-case>
31.         <from-outcome>lose</from-outcome>
32.         <to-view-id>/lose.jsp</to-view-id>
33.     </navigation-case>
34. </navigation-rule>
35. <navigation-rule>
36.     <from-view-id>/win.jsp</from-view-id>
37.     <navigation-case>
38.         <from-outcome>setup</from-outcome>
39.         <to-view-id>/setup.jsp</to-view-id>
40.     </navigation-case>
41. </navigation-rule>
42. <navigation-rule>
43.     <from-view-id>/lose.jsp</from-view-id>
44.     <navigation-case>
45.         <from-outcome>setup</from-outcome>
46.         <to-view-id>/setup.jsp</to-view-id>
47.     </navigation-case>
48. </navigation-rule>
49.
50. <component>
51.     <component-type>com.corejsf.BattleMap</component-type>
52.     <component-class>javax.faces.component.UIInput</component-class>
53. </component>
54.
55. <render-kit>
56.     <renderer>
57.         <component-family>javax.faces.Input</component-family>
58.         <renderer-type>com.corejsf.BattleMap</renderer-type>
59.         <renderer-class>com.corejsf.BattleMapRenderer</renderer-class>
60.     </renderer>
61. </render-kit>
62.
63. <managed-bean>
64.     <managed-bean-name>game</managed-bean-name>
65.     <managed-bean-class>com.corejsf.GameBean</managed-bean-class>
66.     <managed-bean-scope>session</managed-bean-scope>
67. </managed-bean>
68. <managed-bean>
69.     <managed-bean-name>setupform</managed-bean-name>
```



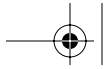
**Listing 11-13** phonebattle/WEB-INF/faces-config.xml (cont.)

```
70.     <managed-bean-class>com.corejsf.SetupForm</managed-bean-class>
71.     <managed-bean-scope>session</managed-bean-scope>
72.     <managed-property>
73.         <property-name>battleGround</property-name>
74.         <value>#{game.own}</value>
75.     </managed-property>
76. </managed-bean>
77. </faces-config>
```

Listing 11-14 phonebattle/WEB-INF/classes/com/corejsf/BattleGround.java

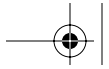
```
1. package com.corejsf;
2.
3. import java.util.ArrayList;
4. import java.util.List;
5. import java.util.Random;
6. import javax.faces.model.SelectItem;
7.
8. public class BattleGround {
9.     public static final int OCCUPIED = 1;
10.    public static final int HIT = 2;
11.
12.    private int rows;
13.    private int columns;
14.    private int currentRow;
15.    private int currentColumn;
16.    private int[][] positions;
17.    private int[] sizes;
18.    private static final int[] INITIAL_SIZES = { 2, 3, 3, 4, 5 };
19.    private static Random generator = new Random();
20.
21.    // PROPERTY: rows
22.    public void setRows(int newValue) { rows = newValue; }
23.    public int getRows() { return rows; }
24.
25.    // PROPERTY: columns
26.    public void setColumns(int newValue) { columns = newValue; }
27.    public int getColumns() { return columns; }
28.
29.    public void initialize() {
30.        sizes = (int[]) INITIAL_SIZES.clone();
31.        positions = new int[rows][columns];
32.    }
33.}
```



**Listing 11-14** phonebattle/WEB-INF/classes/com/corejsf/BattleGround.java

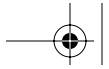
```
34. public void initializeRandomly() {
35.     initialize();
36.     for (int i = 0; i < sizes.length; i++)
37.         addRandomBoat(sizes[i]);
38. }
39.
40. public int getValueAt(int i, int j) {
41.     if (positions == null) return 0;
42.     if (0 <= i && i < rows && 0 <= j && j < columns)
43.         return positions[i][j];
44.     else
45.         return 0;
46. }
47.
48. public void setCurrent(String pos) {
49.     if (pos == null || pos.length() < 2)
50.         throw new IllegalArgumentException();
51.     int r = pos.charAt(0) - 'A';
52.     int c = Integer.parseInt(pos.substring(1)) - 1;
53.     if (r < 0 || r >= rows || c < 0 || c >= columns)
54.         throw new IllegalArgumentException();
55.     currentRow = r;
56.     currentColumn = c;
57. }
58.
59. public void fire() {
60.     if (positions == null) return;
61.     positions[currentRow][currentColumn] |= HIT;
62. }
63.
64. public void addBoat(int size, boolean horizontal) {
65.     addBoat(size, currentRow, currentColumn, horizontal);
66. }
67.
68. public boolean boatFits(int size, boolean horizontal) {
69.     return boatFits(size, currentRow, currentColumn, horizontal);
70. }
71.
72. public void makeRandomMove() {
73.     // try to find a neighbor of an occupied+hit cell that hasn't
74.     // been fired on
75.     for (int i = 0; i < rows; i++)
76.         for (int j = 0; j < columns; j++)
77.             if (positions[i][j] == (OCCUPIED | HIT))
78.                 for (int m = i - 1; m <= i + 1; m++)
```



**Listing 11-14** phonebattle/WEB-INF/classes/com/corejsf/BattleGround.java

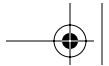
```
79.         for (int n = j - 1; n <= j + 1; n++)
80.             if (m >= 0 && m < rows && n >= 0 && n < columns
81.                 && (positions[m][n] & HIT) == 0) {
82.                 positions[m][n] |= HIT;
83.                 return;
84.             }
85. // pick a random cell that hasn't yet been hit
86. int m = generator.nextInt(rows);
87. int n = generator.nextInt(columns);
88. for (int i = 0; i < rows; i++)
89.     for (int j = 0; j < columns; j++) {
90.         int r = (i + m) % rows;
91.         int s = (j + n) % columns;
92.         if ((positions[r][s] & HIT) == 0) {
93.             positions[r][s] |= HIT;
94.             return;
95.         }
96.     }
97. }
98.
99. public List getAvailableSizes() {
100.     List availableSizes = new ArrayList();
101.     for (int i = 0; i < sizes.length; i++)
102.         if (sizes[i] > 0) {
103.             // is it a duplicate?
104.             boolean found = false;
105.             for (int j = 0; j < i && !found; j++)
106.                 if (sizes[i] == sizes[j]) found = true;
107.             if (!found) {
108.                 String sz = "" + sizes[i];
109.                 availableSizes.add(new SelectItem(sz, sz, null));
110.             }
111.         }
112.     return availableSizes;
113. }
114.
115. public boolean isGameOver() {
116.     for (int i = 0; i < rows; i++)
117.         for (int j = 0; j < columns; j++)
118.             if (positions[i][j] == OCCUPIED /* and not hit */)
119.                 return false;
120.     return true;
121. }
122.
123. private void addBoat(int size, int i, int j, boolean horizontal) {
```



**Listing 11-14** phonebattle/WEB-INF/classes/com/corejsf/BattleGround.java

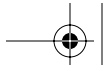
```
124.     if (!boatFits(size, i, j, horizontal)) return;
125.     boolean found = false;
126.     for (int k = 0; !found && k < sizes.length; k++) {
127.         if (sizes[k] == size) {
128.             found = true;
129.             sizes[k] = 0;
130.         }
131.     }
132.     if (!found) return;
133.
134.     for (int k = 0; k < size; k++)
135.         positions[i + (horizontal ? 0 : k)]
136.             [j + (horizontal ? k : 0)] = OCCUPIED;
137. }
138.
139. private boolean boatFits(int size, int i, int j,
140.     boolean horizontal) {
141.     boolean found = false;
142.     for (int k = 0; !found && k < sizes.length; k++) {
143.         if (sizes[k] == size) found = true;
144.     }
145.     if (!found) return false;
146.     if (horizontal && j + size > columns
147.         || !horizontal && i + size > rows)
148.         return false;
149.     for (int k = 0; k < size; k++)
150.         if (positions[i + (horizontal ? 0 : k)]
151.             [j + (horizontal ? k : 0)] != 0)
152.             return false;
153.     return true;
154. }
155.
156. private void addRandomBoat(int size) {
157.     if (rows < size || columns < size) return;
158.     int i;
159.     int j;
160.     boolean horizontal;
161.     boolean fits;
162.     do {
163.         horizontal = generator.nextBoolean();
164.         i = generator.nextInt(rows - (horizontal ? 0 : size));
165.         j = generator.nextInt(columns - (horizontal ? size : 0));
166.     } while (!boatFits(size, i, j, horizontal));
167.     addBoat(size, i, j, horizontal);
168. }
169. }
```



**Listing 11-15** phonebattle/WEB-INF/classes/com/corejsf/GameBean.java

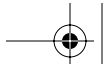
```
1. package com.corejsf;
2.
3. public class GameBean {
4.     private BattleGround own;
5.     private BattleGround opponent;
6.
7.     public GameBean() { initialize(); }
8.
9.     // PROPERTY: own
10.    public BattleGround getOwn() { return own; }
11.    public void setOwn(BattleGround newValue) { own = newValue; }
12.
13.    // PROPERTY: opponent
14.    public BattleGround getOpponent() { return opponent; }
15.    public void setOpponent(BattleGround newValue) { opponent = newValue; }
16.
17.    public String initialize() {
18.        own = new BattleGround();
19.        own.setRows(10);
20.        own.setColumns(10);
21.        own.initialize();
22.        opponent = new BattleGround();
23.        opponent.setRows(10);
24.        opponent.setColumns(10);
25.        opponent.initializeRandomly();
26.        return "setup";
27.    }
28.
29.    public String move() {
30.        opponent.fire();
31.        if (opponent.isGameOver()) return "win";
32.        own.makeRandomMove();
33.        if (own.isGameOver()) return "lose";
34.        return "turn";
35.    }
36. }
```



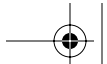
**Listing 11-16** phonebattle/WEB-INF/classes/com/corejsf/SetupForm.java

```
1. package com.corejsf;
2.
3. import javax.faces.application.FacesMessage;
4. import javax.faces.component.UIInput;
5. import javax.faces.component.UISelectOne;
6. import javax.faces.context.FacesContext;
7. import javax.faces.model.SelectItem;
8.
9. public class SetupForm {
10.     private boolean horizontal = true;
11.     private String size = "2";
12.     private String position = "";
13.     private UISelectOne directionComponent;
14.     private UISelectOne sizeComponent;
15.     private BattleGround battleGround;
16.
17.     // PROPERTY: size
18.     public String getSize() {
19.         if (battleGround.getAvailableSizes().size() > 0)
20.             size = ((SelectItem)
21.                 battleGround.getAvailableSizes().get(0)).getLabel();
22.         return size;
23.     }
24.     public void setSize(String newSize) { this.size = newSize; }
25.
26.     // PROPERTY: horizontal
27.     public String getHorizontal() { return "" + horizontal; }
28.     public void setHorizontal(String newHorizontal) {
29.         this.horizontal = Boolean.valueOf(newHorizontal).booleanValue();
30.     }
31.
32.     // PROPERTY: position
33.     public String getPosition() { return position; }
34.     public void setPosition(String newPosition) { this.position = newPosition; }
35.
36.     // PROPERTY: directionComponent
37.     public UISelectOne getDirectionComponent() { return directionComponent; }
38.     public void setDirectionComponent(UISelectOne newValue) {
39.         directionComponent = newValue;
40.     }
41.
42.     // PROPERTY: sizeComponent
43.     public UISelectOne getSizeComponent() { return sizeComponent; }
```



**Listing 11-16** phonebattle/WEB-INF/classes/com/corejsf/SetupForm.java (cont.)

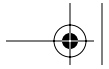
```
44. public void setSizeComponent(UISelectOne newValue) {
45.     sizeComponent = newValue;
46. }
47.
48. // PROPERTY: battleGround
49. public void setBattleGround(BattleGround newBattleGround) {
50.     this.battleGround = newBattleGround;
51. }
52.
53. public void validate(FacesContext context, UIInput input) {
54.     String dval = (String) directionComponent.getValue();
55.     boolean horiz = Boolean.valueOf(dval).booleanValue();
56.
57.     String sval = (String) sizeComponent.getValue();
58.     int sz = Integer.parseInt(sval);
59.     if(!battleGround.boatFits(sz, horiz)) {
60.         input.setValid(false);
61.         context.addMessage(input.getId(),
62.             new FacesMessage(FacesMessage.SEVERITY_ERROR,
63.                 "Boat doesn't fit",
64.                 "The boat that you specified doesn't fit"));
65.     }
66. }
67.
68. public String submitAction() {
69.     battleGround.addBoat(Integer.parseInt(size), horizontal);
70.     if (battleGround.getAvailableSizes().size() == 0)
71.         return "turn";
72.     SelectItem item
73.         = (SelectItem) battleGround.getAvailableSizes().get(0);
74.     size = item.getLabel();
75.     return "setup";
76. }
77. }
1.
```



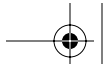
Listing 11-17 phonebattle/WEB-INF/classes/com/corejsf/BattleMapTag.java

```
2. package com.corejsf;
3.
4. import javax.faces.component.UIComponent;
5. import javax.faces.webapp.UIComponentTag;
6.
7. public class BattleMapTag extends UIComponentTag {
8.     private String own;
9.     private String value;
10.    private String validator;
11.
12.    // PROPERTY: own
13.    public void setOwn(String newValue) { own = newValue; }
14.
15.    // PROPERTY: value
16.    public void setValue(String newValue) { value = newValue; }
17.
18.    // PROPERTY: validator
19.    public void setValidator(String newValue) { validator = newValue; }
20.
21.    public void setProperties(UIComponent component) {
22.        super.setProperties(component);
23.
24.        com.corejsf.util.Tags.setString(component, "value", value);
25.        com.corejsf.util.Tags.setBoolean(component, "own", own);
26.        com.corejsf.util.Tags.setValidator(component, validator);
27.    }
28.
29.    public void release() {
30.        own = null;
31.        value = null;
32.        validator = null;
33.    }
34.
35.    public String getRendererType() { return "com.corejsf.BattleMap"; }
36.    public String getComponentType() { return "com.corejsf.BattleMap"; }
37. }
```



**Listing 11-18** phonebattle/WEB-INF/classes/com/corejsf/BattleMapRendererer.java

```
38. package com.corejsf;
39.
40. import java.io.IOException;
41. import java.util.Map;
42. import javax.faces.application.FacesMessage;
43. import javax.faces.component.UIComponent;
44. import javax.faces.component.UIInput;
45. import javax.faces.component.ValueHolder;
46. import javax.faces.context.FacesContext;
47. import javax.faces.context.ResponseWriter;
48. import javax.faces.event.ActionEvent;
49. import javax.faces.render.Renderer;
50.
51. public class BattleMapRendererer extends Renderer {
52.     public void encodeBegin(FacesContext context, UIComponent component)
53.         throws IOException {
54.         ResponseWriter writer = context.getResponseWriter();
55.         String id = component.getId();
56.         Object value = ((ValueHolder) component).getValue();
57.         BattleGround ground = (BattleGround) value;
58.         writer.write(id + "=");
59.
60.         boolean own = ((Boolean)
61.             component.getAttributes().get("own")).booleanValue();
62.         /*
63.          0 = water, not hit, or unknown if not owner
64.          1 = ship, not hit
65.          2 = water, hit
66.          3 = ship, hit
67.         */
68.         for (int i = 0; i < ground.getRows(); i++) {
69.             if (i > 0) writer.write("+");
70.             for (int j = 0; j < ground.getColumns(); j++) {
71.                 int v = ground.getValueAt(i, j);
72.                 boolean hit = (v & BattleGround.HIT) != 0;
73.                 if (own || hit) {
74.                     writer.write('0' + v);
75.                 } else
76.                     writer.write('0');
77.             }
78.         }
79.     }
80. }
```

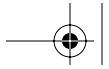
**Listing 11-18** phonebattle/WEB-INF/classes/com/corejsf/BattleMapRenderer.java (cont.)

```
81. public void decode(FacesContext context, UIComponent component) {
82.     if (context == null || component == null) return;
83.
84.     UIInput input = (UIInput) component;
85.     String id = input.getId();
86.     Object value = input.getValue();
87.     BattleGround ground = (BattleGround) value;
88.
89.     // if we don't do the following, then the local value is null
90.     input.setValue(value);
91.
92.     Map parameters
93.     = context.getExternalContext().getRequestParameterMap();
94.     String coords = (String) parameters.get(id);
95.     if (coords == null) return;
96.
97.     try {
98.         ground.setCurrent(coords);
99.         input.queueEvent(new ActionEvent(input));
100.    } catch (Exception ex) {
101.        input.setValid(false);
102.        context.addMessage(id,
103.            new FacesMessage(FacesMessage.SEVERITY_ERROR,
104.                "Invalid position",
105.                "The boat position that you specified is invalid"));
106.    }
107. }
108. }
```

Listing 11-19 phonebattle/misc/BattleshipMIDlet.java

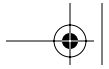
```
1. import java.io.IOException;
2. import java.io.InputStream;
3. import java.io.OutputStream;
4. import java.io.UnsupportedEncodingException;
5. import java.util.Enumeration;
6. import java.util.Hashtable;
7. import javax.microedition.io.Connector;
8. import javax.microedition.io.HttpConnection;
9. import javax.microedition.lcdui.Canvas;
10. import javax.microedition.lcdui.Choice;
11. import javax.microedition.lcdui.ChoiceGroup;
12. import javax.microedition.lcdui.Command;
```



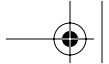
**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
13. import javax.microedition.lcdui.CommandListener;
14. import javax.microedition.lcdui.Display;
15. import javax.microedition.lcdui.Displayable;
16. import javax.microedition.lcdui.Form;
17. import javax.microedition.lcdui.Graphics;
18. import javax.microedition.lcdui.StringItem;
19. import javax.microedition.midlet.MIDlet;
20.
21. public class BattleshipMIDlet extends MIDlet implements CommandListener {
22.     private Display display;
23.     private Form addBoatForm;
24.     private StringItem position;
25.     private ChoiceGroup size;
26.     private ChoiceGroup direction;
27.     private StringItem message;
28.     private StringItem result;
29.     private Command exitCommand;
30.     private Command startCommand;
31.     private Command nextCommand;
32.     private Command addCommand;
33.     private Command opponentCommand;
34.     private Command fireCommand;
35.     private Command continueCommand;
36.     private Command newGameCommand;
37.     private BattleCanvas addBoatCanvas;
38.     private BattleCanvas own;
39.     private BattleCanvas opponent;
40.     private Form startForm;
41.     private Form messageForm;
42.     private Form waitForm;
43.     private Form gameOverForm;
44.     private String webform;
45.     private ConnectionWorker worker;
46.     private Thread workerThread;
47.
48.     // Required methods
49.
50.     public void startApp() {
51.         display = Display.getDisplay(this);
52.         exitCommand = new Command("Exit", Command.EXIT, 1);
53.         createStartForm();
54.         createAddBoatForms();
55.         createBattleCanvases();
56.         createMessageForm();
57.         createGameOverForm();
58.     }
```



**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
59.     worker = new ConnectionWorker();
60.     workerThread = new Thread(worker);
61.     workerThread.start();
62.     waitForm = new Form("Waiting...");
63.
64.     display.setCurrent(startForm);
65. }
66.
67. public void pauseApp() {}
68.
69. public void destroyApp(boolean unconditional) {}
70.
71. // Initialization
72.
73. public void createStartForm() {
74.     startForm = new Form("Start");
75.     startForm.setTitle("Welcome");
76.     startForm.append("Start the Battleship Game");
77.     startCommand = new Command("Start", Command.OK, 0);
78.     startForm.addCommand(startCommand);
79.     startForm.addCommand(exitCommand);
80.     startForm.setCommandListener(this);
81. }
82.
83. public void createAddBoatForms() {
84.     addBoatCanvas = new BattleCanvas();
85.     addBoatCanvas.setTitle("Select Position");
86.     nextCommand = new Command("Next", Command.OK, 0);
87.     addBoatCanvas.addCommand(nextCommand);
88.     addBoatCanvas.addCommand(exitCommand);
89.     addBoatCanvas.setCommandListener(this);
90.
91.     addBoatForm = new Form("Add Boat");
92.     direction = new ChoiceGroup("Direction", Choice.EXCLUSIVE);
93.     size = new ChoiceGroup("Size", Choice.EXCLUSIVE);
94.     position = new StringItem("", null);
95.     addBoatForm.append(direction);
96.     addBoatForm.append(size);
97.     addBoatForm.append(position);
98.     addCommand = new Command("Add", Command.OK, 0);
99.     addBoatForm.addCommand(addCommand);
100.    addBoatForm.addCommand(exitCommand);
101.    addBoatForm.setCommandListener(this);
102. }
103.
```

**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

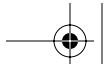
```
104. public void createBattleCanvases() {
105.     own = new BattleCanvas();
106.     own.setTitle("Your battleground");
107.
108.     opponent = new BattleCanvas();
109.     opponent.setTitle("Your opponent");
110.
111.     opponentCommand = new Command("Opponent", Command.OK, 0);
112.     own.addCommand(opponentCommand);
113.     own.addCommand(exitCommand);
114.     own.setCommandListener(this);
115.     fireCommand = new Command("Fire", Command.OK, 0);
116.     opponent.addCommand(fireCommand);
117.     opponent.addCommand(exitCommand);
118.     opponent.setCommandListener(this);
119. }
120.
121. public void createMessageForm() {
122.     messageForm = new Form("Message");
123.     message = new StringItem("", null);
124.     messageForm.append(message);
125.     continueCommand = new Command("Continue", Command.OK, 0);
126.     messageForm.addCommand(continueCommand);
127.     messageForm.addCommand(exitCommand);
128.     messageForm.setCommandListener(this);
129. }
130.
131. public void createGameOverForm() {
132.     gameOverForm = new Form("Game Over");
133.     result = new StringItem("", null);
134.     gameOverForm.append(result);
135.     newGameCommand = new Command("New Game", Command.OK, 0);
136.     gameOverForm.addCommand(newGameCommand);
137.     gameOverForm.addCommand(exitCommand);
138.     gameOverForm.setCommandListener(this);
139. }
140.
141. // Commands
142.
143. public void commandAction(Command c, Displayable s) {
144.     if (c == startCommand) doStart();
145.     else if (c == nextCommand) doNext();
146.     else if (c == addCommand) doAdd();
147.     else if (c == continueCommand) doContinue();
148.     else if (c == opponentCommand) doOpponent();
```



**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

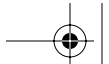
```
149.     else if (c == fireCommand) doFire();
150.     else if (c == newGameCommand) doNewGame();
151.     else if (c == exitCommand) notifyDestroyed();
152.   }
153.
154.   public void doStart() {
155.       connect("setup.faces", null);
156.   }
157.
158.   public void doNext() {
159.       position.setText("Position: " + addBoatCanvas.getString());
160.       display.setCurrent(addBoatForm);
161.   }
162.
163.   public void doAdd() {
164.       Hashtable request = new Hashtable();
165.       request.put("size", size.getString(size.getSelectedIndex()));
166.       request.put("direction",
167.           direction.getString(direction.getSelectedIndex()));
168.       request.put("own", addBoatCanvas.getString());
169.       request.put("form", "setup");
170.       request.put("submit", "");
171.       connect("setup.faces", request);
172.   }
173.
174.   public void doContinue() {
175.       display.setCurrent(addBoatCanvas);
176.   }
177.
178.   public void doOpponent() {
179.       display.setCurrent(opponent);
180.   }
181.
182.   public void doFire() {
183.       Hashtable request = new Hashtable();
184.       request.put("own", own.getString());
185.       request.put("opponent", opponent.getString());
186.       request.put("form", "turn");
187.       request.put("fire", "");
188.       connect("turn.faces", request);
189.   }
190.
191.   public void doNewGame() {
192.       Hashtable request = new Hashtable();
193.       request.put("form", webform);
```



**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

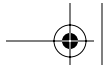
```
194.     request.put("newgame", "");
195.     connect(webform + ".faces", request);
196. }
197.
198. // Connection
199.
200. public void connect(String url, Hashtable request) {
201.     display.setCurrent(waitForm);
202.     worker.connect(url, request);
203. }
204.
205. public void connectionCompleted(Hashtable response) {
206.     webform = (String) response.get("form");
207.     if (webform.equals("setup")) showSetup(response);
208.     else if (webform.equals("turn")) showTurn(response);
209.     else if (webform.equals("win")) showGameOver(response);
210.     else if (webform.equals("lose")) showGameOver(response);
211. }
212.
213. // Navigation
214.
215. public void showSetup(Hashtable response) {
216.     select(size, response, "size");
217.     select(direction, response, "direction");
218.     addBoatCanvas.parse((String) response.get("own"));
219.     String msg = (String) response.get("messages.own");
220.     if (msg != null) {
221.         message.setText(msg);
222.         display.setCurrent(messageForm);
223.         return;
224.     }
225.     display.setCurrent(addBoatCanvas);
226. }
227.
228. public void showGameOver(Hashtable response) {
229.     result.setText((String) response.get("result"));
230.     display.setCurrent(gameOverForm);
231. }
232.
233. public void showTurn(Hashtable response) {
234.     own.parse((String) response.get("own"));
235.     opponent.parse((String) response.get("opponent"));
236.     display.setCurrent(own);
237. }
238.
```



**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
239. private static void select(ChoiceGroup group,
240.     Hashtable response, String name) {
241.     String value = (String) response.get(name);
242.     int i = 0;
243.     String label;
244.     group.deleteAll();
245.     while ((label = (String) response.get(name + ".label." + i))
246.         != null) {
247.         group.append(label, null);
248.         if (label.equals(value))
249.             group.setSelectedIndex(i, true);
250.         i++;
251.     }
252. }
253.
254. private class ConnectionWorker implements Runnable {
255.     private String url;
256.     private String urlPrefix = "http://localhost:8080/phonebattle/";
257.     private Hashtable request;
258.     private Hashtable response;
259.     private String sessionCookie;
260.     private boolean busy = false;
261.
262.     public synchronized void run() {
263.         try {
264.             for (;;) {
265.                 while (!busy)
266.                     wait();
267.                 try {
268.                     byte[] data = post();
269.                     response = decode(data);
270.
271.                 }
272.                 catch (IOException ex) {
273.                     ex.printStackTrace();
274.                 }
275.                 busy = false;
276.                 connectionCompleted(response);
277.             }
278.         }
279.         catch (InterruptedException ie) {}
280.     }
281.
282.     public synchronized void connect(String url, Hashtable request) {
283.         this.url = url;
```



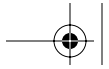
**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
284.         this.request = request;
285.         if (busy) return;
286.         busy = true;
287.         notify();
288.     }
289.
290.     private void urlEncode(String s, OutputStream out)
291.     throws IOException {
292.         byte[] bytes = s.getBytes("UTF8");
293.         for (int i = 0; i < bytes.length; i++) {
294.             byte b = bytes[i];
295.             if (b == ' ')
296.                 out.write('+');
297.             else if ('0' <= b && b <= '9'
298.                 || 'A' <= b && b <= 'Z'
299.                 || 'a' <= b && b <= 'z'
300.                 || "-_!.~*'",).indexOf(b) >= 0)
301.                 out.write(b);
302.             else {
303.                 out.write('%');
304.                 int b1 = (b & 0xF0) >> 4;
305.                 out.write((b1 < 10 ? '0' : 'a' - 10) + b1);
306.                 int b2 = b & 0xF;
307.                 out.write((b2 < 10 ? '0' : 'a' - 10) + b2);
308.             }
309.         }
310.     }
311.
312.     private boolean isspace(byte b) {
313.         return " \n\r\t".indexOf(b) >= 0;
314.     }
315.
316.     private Hashtable decode(byte[] data) {
317.         if (data == null) return null;
318.         Hashtable table = new Hashtable();
319.         try {
320.             int start = 0;
321.             for (;;) {
322.                 while (start < data.length && isspace(data[start]))
323.                     start++;
324.                 if (start >= data.length) return table;
325.                 int end = start + 1;
326.                 int count = 0;
327.                 while (end < data.length && data[end] != '=') end++;
328.                 String key =
```

**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

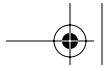
```
329.         new String(data, start, end - start, "ASCII");
330.         start = end + 1;
331.         end = start;
332.         while (end < data.length && !isspace(data[end])) {
333.             count++;
334.             if (data[end] == '%') end += 3; else end++;
335.         }
336.         byte[] b = new byte[count];
337.         int k = start;
338.         int c = 0;
339.         while (k < end) {
340.             if (data[k] == '%') {
341.                 int h = data[k + 1];
342.                 if (h >= 'a') h = h - 'a' + 10;
343.                 else if (h >= 'A') h = h - 'A' + 10;
344.                 else h = h - '0';
345.                 int l = data[k + 2];
346.                 if (l >= 'a') l = l - 'a' + 10;
347.                 else if (l >= 'A') l = l - 'A' + 10;
348.                 else l = l - '0';
349.                 b[c] = (byte) ((h << 4) + l);
350.                 k += 3;
351.             }
352.             else if (data[k] == '+') {
353.                 b[c] = ' ';
354.                 k++;
355.             }
356.             else {
357.                 b[c] = data[k];
358.                 k++;
359.             }
360.             c++;
361.         }
362.         String value = new String(b, "UTF8");
363.         table.put(key, value);
364.         start = end + 1;
365.     }
366. }
367. catch (UnsupportedEncodingException ex) {
368. }
369. return table;
370. }
371.
372. private byte[] post() throws IOException {
373.     HttpURLConnection conn = null;
```



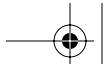
**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
374.     byte[] data = null;
375.
376.     try {
377.         conn = (HttpConnection) Connector.open(urlPrefix + url);
378.
379.         conn.setRequestMethod(HttpConnection.POST);
380.         conn.setRequestProperty("User-Agent",
381.             "Profile/MIDP-2.0 Configuration/CLDC-1.0");
382.         conn.setRequestProperty("Content-Language", "en-US");
383.         conn.setRequestProperty("Content-Type",
384.             "application/x-www-form-urlencoded");
385.         if (sessionCookie != null)
386.             conn.setRequestProperty("Cookie", sessionCookie);
387.
388.         OutputStream out = conn.openOutputStream();
389.         if (request != null) {
390.             Enumeration keys = request.keys();
391.             while (keys.hasMoreElements()) {
392.                 String key = (String) keys.nextElement();
393.                 String value = (String) request.get(key);
394.                 urlEncode(key, out);
395.                 out.write('=');
396.                 urlEncode(value, out);
397.                 if (keys.hasMoreElements()) out.write('&');
398.             }
399.         }
400.
401.         int rc = conn.getResponseCode();
402.         if (rc != HttpURLConnection.HTTP_OK)
403.             throw new IOException("HTTP response code: " + rc);
404.
405.         InputStream in = conn.openInputStream();
406.
407.         // Read the session ID--it's the first cookie
408.         String cookie = conn.getHeaderField("Set-cookie");
409.         if (cookie != null) {
410.             int semicolon = cookie.indexOf(';');
411.             sessionCookie = cookie.substring(0, semicolon);
412.         }
413.
414.         // Get the length and process the data
415.         int len = (int) conn.getLength();
416.         int actual = 0;
417.         int bytesread = 0 ;
418.         if (len > 0) {
```



**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

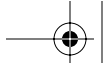
```
419.         data = new byte[len];
420.         while ((bytesread != len) && (actual != -1)) {
421.             actual = in.read(data, bytesread, len - bytesread);
422.             if (actual != -1) bytesread += actual;
423.         }
424.     } else {
425.         final int BLOCKSIZE = 1024;
426.         data = new byte[BLOCKSIZE];
427.         while (actual != -1) {
428.             if (bytesread == data.length) {
429.                 byte[] bigger = new byte[data.length + BLOCKSIZE];
430.                 System.arraycopy(data, 0, bigger, 0, data.length);
431.                 data = bigger;
432.             }
433.             actual = in.read(data, bytesread,
434.                 data.length - bytesread);
435.             if (actual != -1) bytesread += actual;
436.         }
437.         if (bytesread < data.length) {
438.             byte[] smaller = new byte[bytesread];
439.             System.arraycopy(data, 0, smaller, 0, bytesread);
440.             data = smaller;
441.         }
442.     }
443. } catch (ClassCastException e) {
444.     throw new IOException("Not an HTTP URL");
445. } finally {
446.     if (conn != null) conn.close();
447. }
448. return data;
449. }
450. }
451. }
452.
453. class BattleCanvas extends Canvas {
454.     public static final int ROWS = 10;
455.     public static final int COLUMNS = 10;
456.     public static final int OCCUPIED = 1;
457.     public static final int HIT = 2;
458.
459.     private int[][] positions = new int[ROWS][COLUMNS];
460.     private int currentRow = 0;
461.     private int currentColumn = 0;
462.
463.     public void parse(String state) {
```

**Listing 11-19** phonebattle/misc/BattleshipMIDlet.java (cont.)

```
464.     int n = 0;
465.     for (int i = 0; i < ROWS; i++) {
466.         for (int j = 0; j < COLUMNS; j++) {
467.             char c = state.charAt(n);
468.             n++;
469.             positions[i][j] = c - '0';
470.         }
471.         n++;
472.     }
473. }
474.
475. public String getString() {
476.     return "" + (char) ('A' + currentRow) + (1 + currentColumn);
477. }
478.
479. public void paint(Graphics g) {
480.     int width = getWidth();
481.     int height = getHeight();
482.     int oldColor = g.getColor();
483.     g.setColor(0xFFFFFFFF);
484.     g.fillRect(0, 0, width, height);
485.     g.setColor(oldColor);
486.     int cellWidth = width / (COLUMNS + 2);
487.     int cellHeight = height / (ROWS + 2);
488.     int cellSize = Math.min(cellWidth, cellHeight);
489.     for (int i = 0; i <= ROWS; i++) {
490.         int y = (i + 1) * cellSize;
491.         g.drawLine(cellSize, y, (COLUMNS + 1) * cellSize, y);
492.     }
493.     for (int j = 0; j <= COLUMNS; j++) {
494.         int x = (j + 1) * cellSize;
495.         g.drawLine(x, cellSize, x, (ROWS + 1) * cellSize);
496.     }
497.     for (int i = 0; i < ROWS; i++) {
498.         int y = (i + 1) * cellSize;
499.         for (int j = 0; j < COLUMNS; j++) {
500.             int x = (j + 1) * cellSize;
501.             int p = positions[i][j];
502.             if ((p & OCCUPIED) != 0)
503.                 g.fillRect(x, y, cellSize, cellSize);
504.             if ((p & HIT) != 0) {
505.                 if (p == (HIT | OCCUPIED)) {
506.                     oldColor = g.getColor();
507.                     g.setColor(0xFFFFFFFF);
508.                 }

```





Listing 11-19 phonebattle/misc/BattleshipMIDlet.java (cont.)

```
509.         g.drawLine(x, y, x + cellSize, y + cellSize);
510.         g.drawLine(x + cellSize, y, x, y + cellSize);
511.         if (p == (HIT | OCCUPIED)) g.setColor(oldColor);
512.     }
513. }
514. }
515. int x = (currentColumn + 1) * cellSize;
516. int y = (currentRow + 1) * cellSize;
517. g.drawRect(x - 1, y - 1, cellSize + 2, cellSize + 2);
518. }
519.
520. public void keyPressed(int keyCode) {
521.     int action = getGameAction(keyCode);
522.     if (action == LEFT)
523.         currentColumn = (currentColumn + COLUMNS - 1) % COLUMNS;
524.     else if (action == RIGHT)
525.         currentColumn = (currentColumn + 1) % COLUMNS;
526.     else if (action == UP)
527.         currentRow = (currentRow + ROWS - 1) % ROWS;
528.     else if (action == DOWN)
529.         currentRow = (currentRow + 1) % ROWS;
530.     repaint();
531. }
532. }
```

