BEA WebLogic Server

Using WAP with WebLogic Server

WebLogic Server 6.0
Document Edition 1.0
December 2000

Using WAP with WebLogic Server

Document Edition	Date	Software Version
6.0	December 2000	BEA WebLogic Server 6.0 Beta

Contents

Preface

This document explains how to use the wireless application protocol in the BEA WebLogic Server™ environment.

This document discusses the following topics:

- Chapter 1, "Using Wireless Application Protocol (WAP) with WebLogic Server," discusses how to provide content that is suitable for WAE and how to configure and use WebLogic Server with a WAP Gateway.
- Chapter 2, "Programming WAP Applications," discusses how to generate WML, how to specify MIME types, and provides application design considerations.

What You Need to Know

This document is intended primarily for application developers who are interested in building transactional Java applications that run in the WebLogic Server environment. It assumes a familiarity with the WebLogic Server platform and J2EE (Java™ 2, Enterprise Edition) programming, and wireless application protocol concepts.

e-docs Web Site

The BEA WebLogic Server product documentation is available on the BEA Systems, Inc. corporate Web site. From the BEA Home page, click the Product Documentation button or go directly to the "e-docs" Product Documentation page at http://e-docs.bea.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Server documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Server documentation Home page, click the PDF Files button, and select the document you want to print.

If you do not have the Adobe Acrobat Reader installed, you can download it for free from the Adobe Web site at http://www.adobe.com/.

Contact Us!

Your feedback on the BEA WebLogic Server documentation is important to us. Send us e-mail at **docsupport@bea.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Server documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Server and include the release number.

If you have any questions about this version of BEA WebLogic Server, or if you have problems installing and running BEA WebLogic Server, contact BEA Customer Support through BEA WebSUPPORT at www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes

- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and filenames and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`import java.io.Serializable;`
	`public String getName();`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*

Convention	Item
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.
`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
`...`	Indicates one of the following in a command line: ■ That an argument can be repeated several times in a command line ■ That the statement omits additional optional arguments ■ That you can enter additional parameters, values, or other information The ellipsis itself should never be typed. *Example*: `buildobjclient [-v] [-o name ] [-f file-list]... [-l file-list]...`
. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

CHAPTER

1 Using Wireless Application Protocol (WAP) with WebLogic Server

This section includes the following topics:

- Overview
- Wireless Application Environment (WAE)
- The WAP Gateway
- Additional Resources

Overview

Wireless Application Protocol (WAP) is a set of protocols that allow for the development of Internet and Web-based services for mobile phones and other mobile devices. The WAP standard was developed by the WAP Forum whose founding members include Ericsson, Motorola, Nokia, and Phone.com and addresses the limitations of mobile networks (low bandwidth, high latency, and unpredictable availability and stability) and mobile devices (limited CPU, memory, and battery life, and a simple user interface). The WAP Forum has developed their standards in such a way that they leverage and compliment existing industry standards as much as possible. The WAP standard specifies two essential elements of wireless communication: an end-to-end application protocol and an application environment, the Wireless Application Environment (WAE), based on a browser.

There are a number of products currently available that implement the end-to-end application protocol for WAP. These products, called WAP Gateways, form the connection between clients on the mobile network and applications hosted on application servers on the Internet. The WAP Gateway builds a bridge between the telecommunication and computer networks by routing requests from mobile clients to the application servers. It can be physically located in either network, though it is needed in only one of them.

This document discusses how to provide content that is suitable for WAE and how to configure and use WebLogic Server with a WAP Gateway. For general information on WAP technologies, see the Additional Resources section.

Wireless Application Environment (WAE)

WAE defines the framework for network-neutral, wireless applications for narrow-band devices. Two of the main components of WAE are Wireless Markup Language (WML) and WMLScript (WMLS).

Wireless Markup Language (WML)

WML is analogous to HTML for HTTP applications. It is an XML-based language that is specifically designed to interface with the micro-browsers that exist in WAP-enabled devices. The Wireless Markup Language Specification defines the tags and structure of a WML document.

A WML document is a collection of one or more *cards*. Each card is considered a well defined unit of interaction. The general rule of thumb is that a card carries enough information to fit in one screen of a mobile device. One or more cards can be logically grouped into a *deck* of cards. See The WAP Gateway section for information on ways to serve WML documents to mobile clients. For general information on WML, see the Additional Resources section.

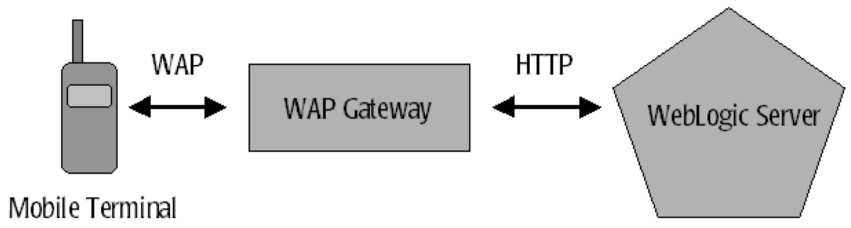
WMLScript (WMLS)

WMLScript provides general scripting capability to the WAP architecture. It is designed to overcome the limitations of narrowband communication and mobile clients. While many of the services that can be used with small mobile clients can be implemented with WML, the human behavioral compatibility of scripting improves the standard browsing and presentation facilities of WML. WMLScript resides in `.wmls` files that can be made available to mobile clients by placing them into the document root. The document root is the root directory for files that are publicly available on WebLogic Server. For more information, see the information on directory structures in Deploying and Configuring Web Applications. For general information on WMLScript, refer to Additional Resources.

The WAP Gateway

The WAP Gateway acts as the bridge between the mobile network containing mobile clients and the computer network containing application servers as shown below.

Figure 1-1 WAP Application Architecture



A WAP Gateway typically includes the following functionality:

- Protocol Gateway—the protocol gateway translates requests from the WAP protocol stack to the WWW protocol stack (HTTP and TCP/IP).
- Content Encoders and Decoders—the content encoders translate Web content into compact encoded formats to reduce the number of packets traveling over the wireless data network.

When a mobile client sends a request to your WAP application running on WebLogic Server, the request is first routed through the WAP Gateway where it is decoded, translated to HTTP, then forwarded to the appropriate URL. The response is then re-routed back through the gateway, translated to WAP, encoded, and forward to the mobile client. This proxy architecture allows application developers to build services that are network and terminal independent.

There is a growing number of vendors that provide WAP Gateways. WebLogic Server should work with any WAP-compliant Gateway. For a current list of WAP-compliant Gateways and other WAP products, refer to the WAP Deployment Fact Sheet compiled by the WAP Forum.

Additional Resources

Related WebLogic technologies

- Programming WebLogic JSP
- Programming WebLogic HTTP Servlets
- Programming WebLogic XML
- Deploying and Configuring Web Applications

General WAP information

- Ericsson: WAP Developers’ Zone
- MobilServer.com
- Motorola
- Nokia: WAP Solutions for Mobile Business
- Phone.com
- WAP Forum
- WAP Hole Sun

Wireless Application Protocol Specifications

White papers

WAP Toolkits
- Nokia WAP Toolkit
- Visual Pulp - WAP content creator
- WAPtop - WML editing tool
- WMLTools

WAP message boards
- WAP Freaks

CHAPTER

2 Programming WAP Applications

This section includes the following topics:

- Generating WML
- Specifying MIME Types
- Application Design Considerations

Generating WML

Requests from mobile clients are routed through the WAP Gateway to WebLogic Server in the form of HTTP requests. WebLogic server can respond to HTTP request by serving static files or HTTP Servlets written as Java Servlets or JavaServer Pages (JSP). For WAP applications, static files will typically be WML files while servlets and JSPs will be used to generate WML dynamically.

The phonebook example in the `samples/examples/wap` subdirectory of your WebLogic distribution demonstrates serving up a static WML file by placing the file into the document root and requesting the file’s URL from a mobile client. The WML file presents options to the user for looking up phone numbers via the `select` element as shown below.

Listing 2-1 phone.wml from samples/examples/wap/phoneBook

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

<!-- Copyright (c) 2000 by BEA Systems, Inc.
     All Rights Reserved. -->

<wml>
  <card id="card1" title="Phone Book" newcontext="true">
    <p>
      Name:
      <select name="name" value="" title="Name">
        <option value="">All</option>

        <option value="John">John</option>
        <option value="Paul">Paul</option>
        <option value="George">George</option>
        <option value="Ringo">Ringo</option>
      </select>
    </p>
    <do type="accept" label="Get number">

      <!-- Edit the URL below to point to the appropriate
           hostname and listenport of your WebLogic Server -->
      <go href="http://localhost:7001/phone?name=$(name:escape)"/>
    </do>
  </card>
</wml>
```

Based on the user's input, an HTTP request is made to `PhoneServlet` and a query parameter (`name`) is added to the servlet's URL. In this example, `PhoneServlet` is an existing servlet example that generates an HTML response. The HTML response is then converted to WML by the WAP Gateway before forwarding the response to the mobile client. Using the WAP Gateway to automatically translate HTML to WML is fine for demonstration purposes, however it is strongly encouraged to generate WML directly since WML is designed to address the display limitations of most WAP devices. See the Application Design Considerations section for additional information.

The “date” example in the `samples/examples/wap` subdirectory of your WebLogic distribution demonstrates generating a WML document from a JSP. When `Date.jsp` is accessed by the mobile client, the page determines the current date and time and returns the results in a WML document as shown below.

Listing 2-2 Date.jsp from samples/examples/wap/date

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
  "http://www.wapforum.org/DTD/wml_1.1.xml">
<!-- Copyright (c) 2000 by BEA Systems, Inc.
     All Rights Reserved. -->

<!-- set document type to WML -->
<%@ page contentType="text/vnd.wap.wml" %>

<wml>
  <template>
    <do type="prev" label="back">
      <prev/>
    </do>
  </template>

  <card title="WML DATE EXAMPLE" id="frstcard">
    <p>
      <small>The current date is:
      <br/>
      <%= new Date() %>
      <br/>
      Copyright &#xA9; 1999-2000 by BEA Systems, Inc.
        All Rights Reserved.</small>
    </p>
  </card>
</wml>
```

As shown in the code above, the line that sets the page’s `contentType` to `text/vnd.wap.wml` is required whenever you are generating WML directly from a JSP or servlet. This line sets the MIME type of the generated document to the WML MIME type. Without this line, the MIME type will default to the HTML MIME type and the WAP Gateway will attempt to translate the document into WML with unfavorable results.

WML is based on XML. Refer to Programming WebLogic XML for additional examples of generating XML from within WebLogic Server.

Specifying MIME Types

To run a WAP application on WebLogic Server, you must specify the MIME types associated with WAP in the `web.xml` file of the web application. The MIME type is defined by the `mime-mapping` deployment descriptor element. For information on creating and editing a web.xml file, see the Writing Web Application Deployment Descriptors section. (In earlier versions of WebLogic Server, MIME types were defined on each server, as server properties. The new method of defining MIME types within the web application is consistent with the Java Servlet Specification 2.2 published by Sun Microsystems, Inc.)

Table 2-1 shows the MIME types required for WAP applications

Table 2-1 MIME Type Definitions for WAP Applications

Extension	Mime Type	Description
`.wml`	`text/vnd.wap.wml`	WML source files
`.wmlc`	`application/vnd.wap.wmlc`	WML compiled files
`.wmls`	`text/vnd.wap.wmlscript`	WMLScript source files
`.wmlsc`	`application/vnd.wap.wmlscriptc`	WMLScript compiled files
`.wbmp`	`image/vnd.wap.wbmp`	Wireless bitmaps

Application Design Considerations

When developing a WAP application, you must consider the limitations of mobile devices and determine the most efficient and flexible way to provide suitable content. This section discusses some of these considerations.

Simple User Interfaces

Most mobile devices have extremely simple user interfaces. WML and WMLScript were specifically designed to address these limitations. While some WAP Gateways have the ability to automatically translate HTML to WML, in practice, it is encouraged to generate WML directly and tailor the interface to the specific needs of the wireless user. Developing a corresponding WML front-end leverages the previous engineering effort to develop the business logic and content of your application, while providing significant user interface benefits.

Because of the limited real estate of graphical displays on most WAP-enabled devices, it is often desirable to allow users to customize the application offering to allow them to see only those services that they are interested in. Tools such as Personalization are well suited for providing this sort of flexibility in your application.

Limited Memory

Most WAP devices have little memory. When grouping WML cards into WML decks, you should be aware that a deck is the smallest download unit. In other words, information is downloaded to a mobile client in decks, not cards. Because of the memory limitations, it is highly recommended to avoid decks with large amounts of cards.

Supporting Multiple Client Types

Typically, a WAP application is an extension of an existing HTML browser-based application. The back-end functionality should not require modification in offering the same services to mobile clients. Instead, a corresponding WML front-end can be developed to leverage the same back-end functionality.

There are two strategies for handling both HTML and WML client types. There can be separate URLs for HTML and WML entry points, or a single URL can be used which will generate content according to the browser type of the requestor. The browser type can be determined by examining the `User-Agent` request header of an HTTP request.

See the SnoopServlet example included in the `examples/servlets` subdirectory of your WebLogic distribution for an example of accessing this type of header information.

A similar strategy can be used, if the developer whishes to take advantage of the different features and display sizes of the different WAP-enabled devices available on the market. The display sizes of WAP-enabled devices currently ranges from four lines of text to about eight lines of text (although this is likely to change dramatically in the near future). By examining the browser type of the client, an application can use the extra graphical real estate only when it is available. Obviously, the simplest method is to create content suitable for the lowest common denominator (four lines).

The future direction is the usage of XSL (eXtensible Style Language). An application can have JSPs and servlets generate XML. An XSL style sheet can then transform the content to into HTML or WML depending on the browser. See Programming WebLogic XML for additional information on XSL.

Session Tracking

Session tracking is useful to keep track of a user's progress over multiple servlets or pages. As described in Programming WebLogic HTTP Servlets, tracking is accomplished by storing session data in a `javax.servlet.http.HttpSession` object that can be retrieved given the session ID. The session ID is typically stored in a cookie that is set in the client. However, WAP does not support cookies.

One alternative is to use URL rewriting which causes the session ID to be encoded into hyperlinks on the page that your servlet sends back to the browser. The session ID is then retrieved from the URL parameters when the link is activated. However, the length of the session ID (to ensure secure sessions with a uniformly random distribution, it is necessary for session IDs to contain a certain number of characters) can cause problems for WAP-enabled devices because many devices limit URLs to 128 characters.

There are two ways to limit the length of the session ID:

- You can limit the length of the portion of the URL that contains the session ID by setting the `IDLength` attribute in the Web Application deployment descriptor `weblogic.xml`, in the `<session-descriptor>` element at http://e-docs.bea.com/wls/docs60/programming/weblogic_xml.html#session-descriptor.

- If your application is not deployed on a cluster of WebLogic Servers, you can specify that information on the primary and secondary servers not be included in the session ID by setting the `WAPEnabled` attribute to `true`. Set the `WAPEnabled` attribute in the `<WebServer>` element in the `config.xml` file for your domain. For more information, see config.xml Elements and Attributes at http://e-docs.bea.com/wls/docs60/config_xml/mbeans.html.