BEA WebLogic Server™

and BEA WebLogic Express™

Programming WebLogic Server for Wireless Services

BEA WebLogic Server Version 6.1
Document Date: April 24, 2003

Programming WebLogic Server for Wireless Services

Part Number	Document Date	Software Version
N/A	September 19, 2001	BEA WebLogic Server Version 6.1

Contents

About This Document

This document explains how to use the BEA WebLogic Server™ platform to design and write Web applications that interface not only with the traditional desktop browser but also with the different types of wireless devices.

This document is organized as follows:

- Chapter 1, “Introduction,” provides the basic information you need to know before using WebLogic Server to extend Web applications to wireless subscribers.
- Chapter 2, “Using WAP with WebLogic Server,” discusses how to provide content suitable for the Wireless Application Protocol (WAP) application environment and how to configure and use WebLogic Server with a WAP Gateway.
- Chapter 3, “Using i-Mode with WebLogic Server,” discusses how to provide content suitable for the i-Mode application environment and how to configure and use WebLogic Server with an i-Mode Gateway.
- Chapter 4, “Writing Web Applications to Include Wireless Subscribers,” demonstrates through example how to use WebLogic Server to extend Web applications to wireless subscribers.

Audience

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

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation.

How to Print the Document

You can print a copy of this document from a Web browser, one main topic 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 Download Documentation, and select the document you want to print.

Adobe Acrobat Reader is available at no charge from the Adobe Web site at `http://www.adobe.com`.

Related Information

The BEA corporate Web site provides all documentation for WebLogic Server. Other WebLogic Server documents that you may find helpful when using WebLogic Server to write application services are:

- Introduction to BEA WebLogic Server at `http://e-docs.bea.com/wls/docs61/intro/index.html`
- Administration Guide at `http://e-docs.bea.com/wls/docs61/adminguide/index.html`
- Programming WebLogic XML at `http://e-docs.bea.com/wls/docs61/xml/index.html`

Contact Us!

Your feedback on BEA 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 documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and document date of your documentation. 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 http://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	Usage
Ctrl+Tab	Keys you press simultaneously.
italics	Emphasis and book titles.

`monospace text`	Code samples, commands and their options, Java classes, data types, directories, and file names and their extensions. Monospace text also indicates text that you enter from the keyboard.
	Examples:
	`import java.util.Enumeration;`
	`chmod u+w *`
	`config/examples/applications`
	`.java`
	`config.xml`
	`float`

monospace italic text	Variables in code. *Example*: `String` *`CustomerName`*`;`
UPPERCASE TEXT	Device names, environment variables, and logical operators. *Examples*: LPT1 BEA_HOME OR
`{ }`	A set of choices in a syntax line.
`[ ]`	Optional items in a syntax line. *Example*: `java utils.MulticastTest -n name -a address [-p portnumber] [-t timeout] [-s send]`

Convention	Usage

`|`	Separates mutually exclusive choices in a syntax line. *Example*:
	`java weblogic.deploy [list|deploy|undeploy|update]`
	`     password {application} {source}`

`...`	Indicates one of the following in a command line: ■ An argument can be repeated several times in the command line. ■ The statement omits additional optional arguments. ■ You can enter additional parameters, values, or other information
. . .	Indicates the omission of items from a code example or from a syntax line.

CHAPTER

1 Introduction

The following sections provide information that you need to know before using the BEA WebLogic Server™ platform to extend Web applications to wireless subscribers:

- Overview
- Wireless Data Protocols
- The WAP Protocol at a Glance
- The i-Mode Protocol at a Glance
- Other Wireless Data Protocols
- Other Wireless Markup Languages
- Evolution of Wireless Data Protocols and Wireless Markup Languages
- Additional Information

Overview

The wireless Internet-enabled subscriber base continues to grow. Telecommunications companies and wireless service providers are looking for ways to rapidly create and deploy new revenue-generating Internet services while providing their wireless subscribers with a more personalized experience.

Because wireless subscribers have a different set of essential desires and needs than desktop or even laptop Internet users, Internet services must be designed to present the optimum experience for subscribers using different types of wireless devices. Writing Internet services in this manner requires a deep understanding of the technical issues unique to the wireless environment.

Wireless Data Protocols

Using Internet technologies such as Hypertext Transfer Protocol (HTTP), Transport-Layer Security (TLS), Transmission Control Protocol (TCP), and Hypertext Markup Language (HTML) to set up and tear down connections and to transport Web content on a wireless network is inefficient for several reasons, including:

- HTTP and TCP are not optimized for the intermittent coverage, long latencies, and limited shared bandwidth associated with wireless networks.
- HTTP sends its headers and commands in a text format, which is not well suited for the low-bandwidth constraints of wireless networks, instead of the more efficient compressed binary format.
- HTTP session setup and teardown require many messages to be exchanged between the wireless device (client) and the server.
- HTML Web content intended for the desktop browser cannot be displayed in an effective way on the small-size screens of handheld mobile phones, personal digital assistants (PDAs), pagers, two-way radios, and smartphones.
- Navigation around and between screens is not easy due to no keyboard and no mouse; navigation is accomplished by pressing numeric keys or writing with a stylus.

Wireless services using these protocols are often slow, costly, and difficult to use, which is why special *wireless data protocols* were created to transport Web-based data on a wireless network. Wireless data protocols, such as Wireless Application Protocol (WAP), i-Mode, and others, are designed for the unique constraints of the wireless environment. They use binary transmission for greater compression of data and are optimized for long latency and low to medium bandwidth. They support protocol data unit (PDU) concatenation and delayed acknowledgement to help reduce the number of messages sent. Their associated markup languages make optimum use of small screens and allow easy navigation around and between screens.

To read the markup language associated with a wireless data protocol, a wireless device requires a *microbrowser*, which is client software specially designed to interpret the markup language. As examples, a WAP-enabled device has a Wireless Markup Language (WML) microbrowser, and an i-Mode-enabled device has a *compact* HTML (cHTML) microbrowser.

The WAP Protocol at a Glance

The Wireless Application Protocol (WAP) specification consists of a wireless data protocol—a standardized way that the microbrowser on a wireless device communicates with a WAP gateway installed in the wireless network—and a markup language, Wireless Markup Language (WML). WML is an Extensible Markup Language (XML) used to specify the content and user interface for WAP-enabled devices.

WAP sessions cope with intermittent coverage and can operate over a wide variety of wireless bearer networks including but not limited to Cellular Digital Packet Data (CDPD), Code Division Multiple Access (CDMA), Global System for Mobiles (GSM), Time Division Multiple Access (TDMA), and General Packet Radio Service (GPRS). Although most WAP services in Europe and the United states are circuit-switched (dial-up), WAP will also work on packet-switched networks.

The i-Mode Protocol at a Glance

The i-Mode specification consists of a wireless data protocol—a standardized way that the microbrowser on a wireless device communicates with an i-Mode gateway installed in the wireless network—and a markup language, *compact* HTML (cHTML). cHTML is a subset of HTML with extensions and is used to specify the content and user interface for i-Mode-enabled devices.

Note: In publications, i-Mode appears in many different forms including *i-Mode*, *I-Mode*, *I-mode*, *i-mode*, and *imode*. *i-Mode* is used throughout this discussion and the discussions that follow.

While WAP is an open, global specification, i-Mode is currently a proprietary, closed specification developed and deployed by NTT DoCoMo of Japan. The only i-Mode implementation is NTT DoCoMo’s mobile internet access system, although several telecommunication companies in Europe and the United States have expressed an interest in i-Mode.

Although i-Mode currently operates only on NTT-DoCoMo’s PDC-P mobile voice system, i-Mode will work equally well on any underlying wireless bearer network.

Other Wireless Data Protocols

Other wireless data protocols include:

- Handheld Device Transport Protocol (HDTP) developed by Phone.com (formerly Unwired Planet and now Openwave Systems Inc.)
- Mobitex developed by the Mobitex Operators Association

The wireless industry is also very interested in the Voice over IP (VoIP) technology, which moves voice and data traffic over a common IP infrastructure. The wireless industry is looking for ways to converge the VoIP and wireless data protocol technologies so that voice and Web data can be carried on the same over-the-air channel.

Other Wireless Markup Languages

Other wireless markup languages include:

- Handheld Device Markup Language (HDML) developed by Phone.com (formerly Unwired Planet and now Openwave Systems Inc.)
- Web Clipping developed by Palm, Inc.
- Extensible HTML (XHTML) (Basic) developed by the World Wide Web Consortium (W3C)
- VoiceXML developed by the VoiceXML Forum, an industry organization founded by AT&T, IBM, Lucent, and Motorola

Note: HDTP carries Web content tagged with HDML. HDTP and HDML heavily influenced the development of WAP and WML.

Evolution of Wireless Data Protocols and Wireless Markup Languages

The various wireless data protocols and wireless markup languages in existence today may very well evolve into a single wireless data protocol and wireless markup language tomorrow. As evidence, proponents of the WAP protocol are about to migrate to a new generation of the protocol known as WAP-NG, and both AT&T an NTT DoCoMo have made allegiances to WAP-NG. In addition, proponents of WAP and i-Mode are talking about migrating to the common markup language XHTML (Basic).

Additional Information

Here are some Web sites (organized by category) to visit for additional information about wireless data protocols, wireless markup languages, manufacturers of wireless devices, and wireless standards and specifications.

Related WebLogic Wireless Information

BEA-written white paper titled "Beyond the Wire—Developing Software for Many Devices" at `http://www.bea.com/products/weblogic/server/paper_pervasive.shtml`

BEA Wireless Newsgroup at `news://newsgroups.bea.com/weblogic.developer.interest.wap`

BEA Wireless FAQ at `http://e-docs.bea.com/wls/docs61/faq/wireless.html`

General Wireless Information

The Wireless FAQ at `http://allnetdevices.com/faq`

FierceWireless at `http://www.fiercewireless.com`

MBizCentral at `http://www.mbizcentral.com`

Unstrung at `http://www.unstrung.com`

WirelessDevNet at `http://www.wirelessdevnet.com`

Wireless-Device Manufacturers

Palm at `http://www.palm.com/wireless`

PocketPC at `http://www.microsoft.com/mobile/pocketpc/faq.asp`

RIM BlackBerry at `http://www.rim.net/products/handhelds/index.shtml`

Symbian Limited at `http://www.symbian.com`

Wireless Standards and Specifications

XHTML Basic W3C Recommendation at `http://www.w3.org/TR/xhtml-basic`

VoiceXML Forum Specifications at `http://www.voicexml.org/`

The Short Message Peer to Peer (SMPP) Forum at
`http://www.smpp.org/index.html`

SMS Protocol—SMPP Specification at
`http://www.smpp.org/doc/public/index.html`

SMS Protocol—CIMD2 Specification at
`http://www.forum.nokia.com/download/cimdspec2.pdf`

SNPP pagers at `http://www.faqs.org/rfcs/rfc1861.html`

CHAPTER

2 Using WAP with WebLogic Server

The following sections describe how to provide content suitable for the Wireless Application Protocol (WAP) application environment and how to configure and use WebLogic Server with a WAP gateway:

- Overview
- WAP Application Environment
- WAP Gateway
- Additional Resources

Overview

Wireless Application Protocol (WAP) is an open, global specification, developed and deployed by the WAP Forum, that allows for the development of Internet and Web-based services for mobile phones and other wireless digital devices. Its founder members include the major wireless vendors of Ericsson, Motorola, Nokia, and Phone.com (formerly Unwired Planet and now Openwave Systems Inc).

The WAP specification addresses the limitations of wireless networks (low bandwidth, high latency, and unpredictable availability and stability) and wireless devices (limited CPU, memory, and battery life, and a simple user interface). It specifies two essential elements of wireless communication: an over-the-air wireless protocol and an application environment.

WAP gateways form the connection between clients on the wireless 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 wireless clients to the application servers. It can be physically located in either network, though it is needed in only one of them.

WAP Application Environment

The WAP application environment defines the framework for network-neutral, wireless applications for narrowband devices. Two of the main components of the WAP application environment are Wireless Markup Language (WML) and WMLScript.

WML

WML for WAP applications is analogous to HTML for TCP/IP applications. It is an XML-based language that is specifically designed to interface with the microbrowsers that exist in WAP-enabled devices. The Wireless Markup Language Specification at `http://www.wapforum.org/what/technical.htm` defines the tags and structure of a WML document.

A WML document is a collection—referred to as a *deck*—of one or more *cards*. Each card in a deck of cards 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 wireless device. For information on ways to serve WML documents to wireless clients, see "WAP Gateway" on page 2-3. For general information on WML, see "Additional Resources" on page 2-6.

WMLScript

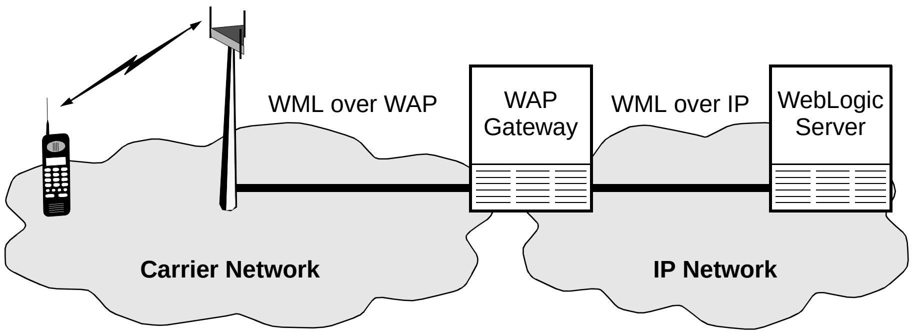
WMLScript provides general scripting capability to the WAP architecture. It is designed to overcome the limitations of narrowband communication and wireless clients. For example, WMLScript is a good way to validate form input without making a round trip to the server.

WMLScript resides in `.wmls` files that are made available to wireless 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 detail, see the information on directory structures in Web Applications Basics at `http://e-docs.bea.com/wls/docs61/webapp/basics.html`. For general information on WMLScript, see "Additional Resources" on page 2-6.

WAP Gateway

As shown in the following figure, the WAP gateway acts as the bridge between the wireless network containing wireless clients and the computer network containing application servers.

Figure 2-1 WAP Application Architecture



WAP Gateway Functionality

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 and size of packets traveling over the wireless data network.

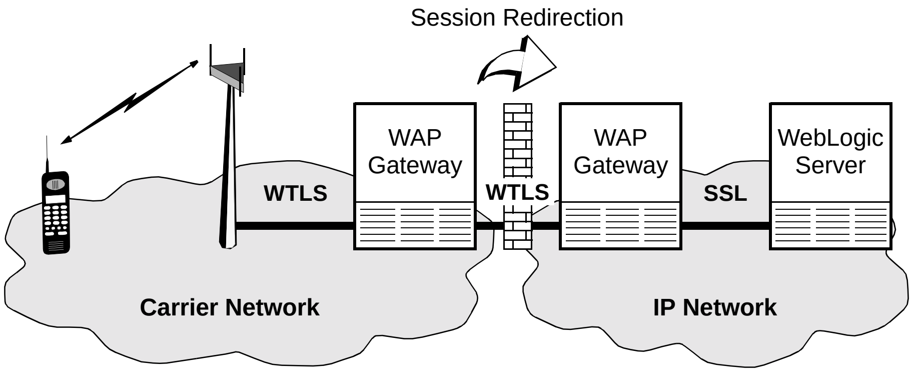
When a wireless client sends a request to a WAP application running on WebLogic Server, the request is first routed through the WAP gateway where it is decoded, translated to HTTP, then forwarded to the appropriate URL. The response is then re-routed back through the gateway, translated to WAP, encoded,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

WAP Gateway Security and Security Concerns

The security layer of the WAP protocol stack is called Wireless Transport Layer Security (WTLS). WTLS is based upon the established Transport-Layer Security (TLS) protocol standard.

For a secure connection employing the WAP protocol, a very small security risk exists at the WAP gateway during the switching of WTLS (WAP side) to SSL (IP side) and SSL to WTLS. Since the WAP protocol allows a session to be redirected from the carrier’s gateway to the enterprise’s gateway, an enterprise may want to control this minimal risk by including a WAP gateway behind its firewall. As shown in the following figure, the enterprise secures the server running the WAP gateway in a controlled environment to eliminate any exposure to the security risk.

Figure 2-2 WAP Session Redirection



Since the carrier is a trusted entity and is continuously responsible for protecting voice, fax, computer and other types of data, enterprises probably do not need to host their own WAP gateway.

WAP Gateway Vendors

There are a growing number of vendors that provide WAP gateways. WebLogic Server should work with any WAP-compliant gateway. For a list of WAP-compliant gateways and other WAP products, refer to the WAP Deployment Fact Sheet at `http://www.wapforum.org/new/index.htm` compiled by the WAP Forum.

Additional Resources

Here are some Web sites (organized by category) to visit for additional information about WebLogic Server programming, WAP, and WML.

Related WebLogic Technologies

Programming WebLogic JSP at `http://e-docs.bea.com/wls/docs61/jsp/index.html`

Programming WebLogic HTTP Servlets at `http://e-docs.bea.com/wls/docs61/servlet/index.html`

Programming WebLogic XML at `http://e-docs.bea.com/wls/docs61/xml/index.html`

Web Applications Basics at `http://e-docs.bea.com/wls/docs61/webapp/basics.html`

General WAP Information

WAP Forum at `http://www.wapforum.org`

Ericsson: Developers' Zone at `http://www.ericsson.com/developerszone`

Motorola at `http://developers.motorola.com/developers/wireless`

Nokia: WAP Solutions for Mobile Business at `http://www.nokia.com/corporate/wap/index.html`

Openwave Developer Resources at `http://developer.openwave.com/resources/index.html`

WAP Specifications and White Papers

WAP specifications at `http://www.wapforum.org/what/technical.htm`

WAP white papers at `http://www.wapforum.org/what/whitepapers.htm`

WAP Toolkits

Nokia WAP Toolkit at `http://www.nokia.com/corporate/wap/sdk.html`

Motorola Tools and Downloads at
`http://developers.motorola.com/developers/wireless/tools`

Openwave Software Development Kit at
`http://developer.openwave.com/download/index.html`

Ericsson Developers’ Zone at `http://www.ericsson.com/developerszone`

3 Using i-Mode with WebLogic Server

The following sections describe how to provide content suitable for the i-Mode application environment and how to configure and use WebLogic Server with an i-Mode gateway:

- Overview
- i-Mode Markup Language
- i-Mode Gateway
- Additional Resources

Overview

i-Mode is currently a proprietary, closed specification, developed and deployed by NTT DoCoMo of Japan, that allows for the development of Internet and Web-based services for mobile phones and other wireless digital devices. The only i-Mode implementation is DoCoMo's mobile internet access system, although several telecommunication companies in Europe and the United States have expressed an interest in i-Mode.

Note: i-Mode is a trademark and/or service mark owned by NTT DoCoMo.

The i-Mode specification addresses the limitations of wireless networks (low bandwidth, high latency, and unpredictable availability and stability) and wireless devices (limited CPU, memory, and battery life, and a simple user interface). It specifies two essential elements of wireless communication: an over-the-air wireless protocol and a markup language.

i-Mode gateways form the connection between clients on the wireless network and applications hosted on application servers on the Internet. The i-Mode gateway builds a bridge between the telecommunication and computer networks by routing requests from wireless clients to the application servers.

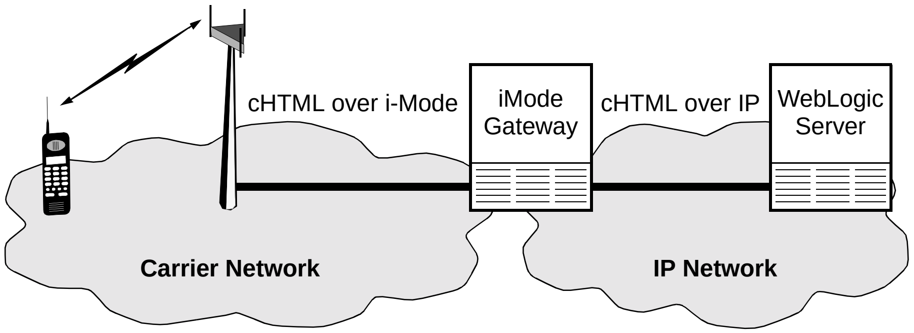
i-Mode Markup Language

i-Mode applications use the *compact* Hypertext Markup Language (cHTML), which is specifically designed to interface with the microbrowsers in i-Mode-enabled devices. cHTML is a subset of HTML, with a number of extensions for the mobile phone environment. The NTT DoCoMo (English) Supported Tags and Specs at `http://www.nttdocomo.com/` defines the tags and structure of a cHTML document.

i-Mode Gateway

As shown in the following figure, the i-Mode gateway acts as the bridge between the wireless network containing wireless clients and the computer network containing application servers.

Figure 3-1 i-Mode Application Architecture



An i-Mode gateway typically includes a protocol gateway, which translates requests from the i-Mode protocol stack to the WWW protocol stack (HTTP and TCP/IP).

When a wireless client sends a request to an i-Mode application running on WebLogic Server, the request is first routed through the i-Mode gateway and then forwarded to the appropriate URL. The response is then re-routed back through the gateway, translated to i-Mode, and forward to the wireless client. This proxy architecture allows application developers to build services that are network and terminal independent.

Additional Resources

Here are some Web sites (organized by category) to visit for additional information about WebLogic Server programming, i-Mode, and cHTML.

Related WebLogic Technologies

Programming WebLogic JSP at
`http://e-docs.bea.com/wls/docs61/jsp/index.html`

Programming WebLogic HTTP Servlets at
`http://e-docs.bea.com/wls/docs61/servlet/index.html`

Programming WebLogic XML at
`http://e-docs.bea.com/wls/docs61/xml/index.html`

Web Applications Basics at
`http://e-docs.bea.com/wls/docs61/webapp/basics.html`

General i-Mode Information

All about i-mode at http://www.nttdocomo.com/corebiz/imode/index.html

CHAPTER

4 Writing Web Applications to Include Wireless Subscribers

The following sections demonstrate through example how to use WebLogic Server to extend Web applications to wireless subscribers:

- Overview
- Locating the Wireless Example Applications
- Understanding Basic Web Application Design Concepts
- Writing WAP-Only Web Applications
- Writing Multi-Target Web Applications

Overview

With the growing popularity of Internet-enabled wireless devices, application developers need to design their Web applications to interface with more than just the traditional desktop browser. In addition, because wireless devices vary greatly in their form factors, their user input interfaces, and their use of over-the-air (OTA) protocols and markup languages, developers need to consider the different types of wireless devices when designing Web applications.

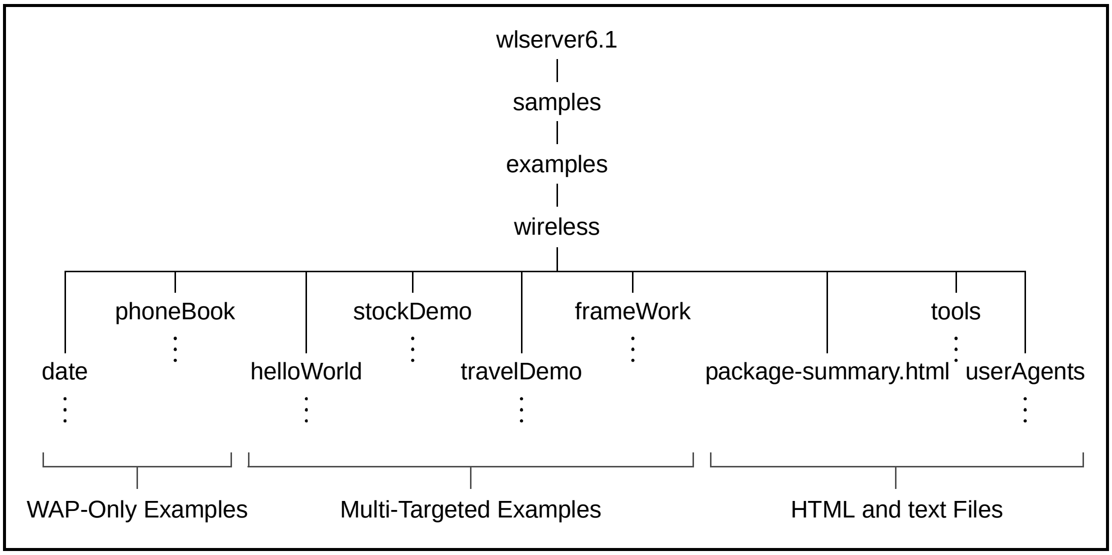
The good news is that any existing application built on WebLogic Server is ready for the wireless Web. In fact, for a well written application, an application's entire back-end is ready for use by the new Internet-enabled wireless devices without any significant code modifications. In the simplest case, an application developer need only write new front-end software to enable an application to interface with Internet-enabled wireless devices.

The sections that follow explore different ways of extending Web application services to wireless devices. These methods are by no means the only methods for extending services to wireless devices, but are intended to help you use the WebLogic Server J2EE features to develop your own methods. Five simple example applications are included to clarify the procedures.

Locating the Wireless Example Applications

During the WebLogic Server software installation, the installer program places the directories and files for the five wireless example applications in the `wireless` directory, as shown in the following figure.

Figure 4-1 WebLogic Server Wireless File Tree



The product directory shown here, `wlserver6.1`, is the default for WebLogic Server 6.1. The default name can be changed during installation.

The top-level directories and files of the WebLogic Server wireless directory structure are briefly described in the following table. The five examples demonstrate how to access JavaServer Pages (JSPs) and Java Servlets from a wireless device or desktop browser.

Directory Name	**Description**
`date`	Contains sample code and resources for the `date` example, including a `package-summary.html` file describing the building, configuring, and running of the example in detail. The `date` example demonstrates (1) connectivity between a WML client device and WebLogic Server through a WAP gateway and (2) generation of dynamic WML documents (content tagged with WML).
`phoneBook`	Contains sample code and resources for the `phoneBook` example, including a `package-summary.html` file describing the building, configuring, and running of the example in detail. The `phoneBook` example demonstrates (1) connectivity between a WML client device and WebLogic Server through a WAP gateway, (2) serving up a static WML document, and (3) making an interactive request to a servlet received from the WML client.
`helloWorld`	Contains sample code and resources for the `helloWorld` example, including a `package-summary.html` file describing the building, configuring, and running of the example in detail. The `helloWorld` example demonstrates the use of the underlying examples framework to serve up a static markup-language document specific to the requesting wireless or desktop device. The document may be WML, HDML, cHTML, or HTML.

Directory Name	Description
`stockDemo`	Contains sample code and resources for the `stockDemo` example, including a `package-summary.html` file describing the building, configuring, and running of the example in detail. The `stockDemo` example demonstrates the use of the underlying examples framework to serve up a simple dynamic markup-language document specific to the requesting wireless or desktop device. The document may be WML, HDML, cHTML, or HTML.
`travelDemo`	Contains sample code and resources for the `travelDemo` example, including a `package-summary.html` file describing the building, configuring, and running of the example in detail. The `travelDemo` example, which has multiple levels of user interaction, demonstrates the use of the underlying examples framework to serve up a more complex markup-language document specific to the requesting wireless or desktop device. The document may be WML, HDML, cHTML, or HTML.
`frameWork`	Contains sample code and resources for a simple, prototypical framework, including a `package-summary.html` file describing the framework in detail. The `frameWork` directory, which contains the examples framework used in the `helloWorld`, `stockDemo`, and `travelDemo` examples, generates markup language documents from within a JSP specific to the requesting wireless or desktop device. The document may be WML, HDML, cHTML, or HTML.
`package-summary.html` (file)	Contains a summary of the five wireless example applications provided by WebLogic Server, and provides a list of pointers to useful related information.

Directory Name	Description
`tools`	Contains two files listing pointers to the following types of tools: tools that convert images to the WML bitmap (WBMP) format for WAP devices, and tools that emulate wireless client devices.
`userAgents`	Contains a file listing sample userAgents from real devices; userAgent information appears in HTTP `User-Agent` headers.

Understanding Basic Web Application Design Concepts

You, as an application developer, can use a Web application's data and business logic for both wireline desktop devices and wireless devices, but you must tailor the layout and navigation for the different classes of devices. *The application flow should often be much different for a desktop browser than for a wireless device microbrowser.*

For example, consider the following application flow for presenting an e-commerce site on a desktop browser:

The e-commerce site home page presents many links to different sections of the e-store, a search form, and several advertisements for current specials. After selecting a part of the store, a product category, and a specific product, the consumer is presented with a full description of the product and an option to add the product to the shopping cart. After selecting the add option, the consumer is presented with options to continue shopping or to proceed to checkout.

This application flow encourages consumers to browse through more of the e-store and potentially make more purchases than they originally intended. With a desktop browser and a mouse, this flow works very well. However, using this flow to present the e-commerce site on a data-enabled wireless device may cause a consumer to leave the e-store in frustration; wireless devices simply are not well suited for browsing. As an alternative, consider the following application flow for presenting the same e-commerce site on a cell phone:

The home screen allows the consumer to enter a UPC bar code or some other unique identification for the product. The follow-up screen contains a link to add the item to the cart, proceed to checkout, or get a full description of the product. A link to the "general e-store" appears on each page so that consumers who want to browse can do so.

Note that this application flow is the inversion of the desktop-browser application flow. The key to creating an application flow for a wireless device is to limit the amount of user interaction with the device's input and selection mechanisms.

So when developing a Web application, consider the limitations of wireless devices and determine the most efficient and flexible way to provide suitable content. The following sections describe some of these considerations:

- "Form Factors" on page 4-6
- "Physical User Interfaces" on page 4-7
- "Limited Memory" on page 4-7
- "Support for Multiple Client Types" on page 4-8
- "Personalization" on page 4-9
- "Session Tracking" on page 4-9

Form Factors

Most wireless devices have different *form factors* due to the different sizes and shapes of their screen displays. Wireless markup languages, such as WML, HDML, cHTML, Web Clipping, and others, are specifically designed to accommodate these form factors. While some wireless gateways (WAP gateways, i-Mode gateways, ...) have the ability to automatically translate HTML to a different markup language, in practice the best applications use the markup language directly to tailor the interface to the specific needs of the wireless user. Doing so allows for the best possible use of a wireless device's form factor and provides the best user experience.

Since many wireless devices can only display four or five lines of text with ten to twenty characters per line, the challenge is to present information on small screens in such a way that users will be pleased with the display.

Physical User Interfaces

Most wireless devices have extremely simple user interfaces due to the lack of a keyboard and a mouse. Entering input or navigating between screens is accomplished by pressing numeric keys or buttons, or writing with a stylus.

You need to consider the data entry mechanism as well as the amount of data to be entered when designing and writing applications. If possible, you should separate an application's required data entry points from its optional data entry points, and only present the required data fields to wireless devices.

Using markup languages such as WML, HDML, cHTML, Web Clipping, and others, you can write application front-ends that assign operations to the keys and buttons on wireless devices. For a cell phone, as an example, you can program an application front-end to assign specific operations to the unlabeled *power* keys (operations such as <PREV>, <BACK>, <OK>, <SUBMIT>, <SELECT>, <DONE>, <NEXT>, <ABORT>, and <HOME>) and to reassign new operations to the labeled *soft* keys. For a PDA device, you can program an application front-end to reassign new operations to the soft buttons (the buttons imprinted on silk) and the hard buttons. Being able to assign keys or buttons on a wireless-device version of an application *to replace* the buttons and links in the desktop-browser version of an application is a very useful data entry and navigation capability.

Limited Memory

Most wireless devices have little memory. When grouping WML cards into decks, you should be aware that a deck is the smallest download unit. In other words, information is downloaded to a wireless client in decks, not cards. Because of the memory limitations, we at BEA highly recommend that you avoid decks with a large number of cards or single cards that are large.

Note: It is not always lack of memory that limits the number or size of cards in a deck. Some wireless gateways put limits on the size of a single payload delivered to the wireless clients as well.

Support for Multiple Client Types

The most valuable parts of a Web application are its unique content and its back-end database interaction, not the particular type of markup language (traditionally HTML) written to interact with the user. A well designed back-end functionality should not require modification to be able to offer the same services to wireless clients. You, as an application developer, simply need to develop markup-language specific front-ends (WML, HDML, cHTML, ...) to extend the same back-end functionality to the wireless devices. In addition, you need to define customized presentations for the different classes of devices (cell phones, PDAs, traditional desktop browsers, ...) and add features that are unique to specific devices, such as personalization for cell phones based on the geographic location of the requesting device.

Two basic strategies exist for handling user access to HTML-based browser and wireless client types: programming a Web application to use separate URLs for browser-based HTML and microbrowser-based WML/HDML/cHTML/... entry points (such as `www.foo.com`, `www.wap.foo.com`, and `www.imode.foo.com`), or programming the application to use a single URL (for example, `www.foo.com`) to generate content according to the browser type of the requestor. For the latter strategy, the Web application at the URL determines the browser type by examining the HTTP `User-Agent` header of the request and/or soliciting information about the requestor from the serving wireless gateway. For an example of accessing HTTP `User-Agent` header information, see the `SnoopServlet` example included in the `samples/examples/servlets` directory of your WebLogic Server installation.

You can use the same two strategies if you wish to take advantage of the different features and display sizes of the different wireless devices available on the market. The display sizes of wireless devices currently range 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 type of the client, an application can use the extra graphical real estate when it is available. Obviously, the simplest method is to create content suitable for the lowest common denominator (four lines), although this method does not lend itself to the best user experience on most devices.

Generally speaking, we at BEA strongly encourage you to use the same URL for all devices, including the traditional desktop browser, to make user access to an application as easy as possible.

Personalization

You should consider using a personalization mechanism that allows users to customize an application offering based on device class. For example, the services that a user wants to see on one device (say a desktop browser) may be very different from the services that the user wants to see on another device (say a cell phone). Tools such as BEA WebLogic Personalization Servers are well suited for providing this sort of flexibility in your application.

The personalization mechanism should also allow users to create portals based on device class and location. For example, a user may want a certain portal for a device (say a cell phone) when visiting on the east coast and a different portal for that same device when visiting on the west coast. Additionally, a user may want to personalize a portal based on the location of the device, such as:

When the device is more than twenty miles from home, local restaurant names appear on the home page. When the device is within twenty miles from home, no restaurant names appear.

The personalization mechanism should also allow for the sending of alerts to different devices based on the time of day or the day of the week. For example, if a user does not carry his or her pager on the weekends, a voice alert could be generated and left on an answering machine.

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 at `http://e-docs.bea.com/wls/docs61/servlet/index.html`, the server assigns each client session a session ID and an associated `javax.servlet.http.HttpSession` object that lives on the WebLogic Server for the lifetime of the session. The client provides the session ID with each request in order for the server to find session data in the `HttpSession` object.

The session ID for an application *not* deployed on a cluster of WebLogic Servers has the following format:

$$\underbrace{\textit{Rand\_Sess\_ID!}}_{\substack{\text{52 bytes}\\ \text{(default)}}}\underbrace{\textit{Primary\_JVMID\_DIFFERENTIATOR!HOST!PORT!SSLPORT}}_{\text{JVMID for Primary WebLogic Server} = \sim 60 \text{ bytes}}$$

A session ID for an application deployed on a cluster of WebLogic Servers has the following format consisting of an additional 60 bytes:

```
Rand_Sess_ID!Primary_JVMID_DIFFERENTIATOR!HOST!PORT!SSLPORT!
     SECONDARY_JVMID_DIFFERENTIATOR!HOST!PORT!SSLPORT
```

The server attempts to store the session ID by setting a *cookie* on the client. (Cookies are used on the fixed Internet to identify the Web browser and thereby assist in providing customized and streamlined services.) Once the cookie is set, the client includes the cookie with each user request sent to the server. The server automatically parses the session ID from the cookie to retrieve the session data in the appropriate `HttpSession` object.

However, since most wireless devices do not support cookies, and some wireless gateways do not handle cookies for their clients, you may have to choose an alternate session-tracking method.

An Alternative Session-Tracking Method—URL Rewriting

URL rewriting involves encoding the session ID into the hyperlinks on the page that your servlet sends back to the client. When the user subsequently clicks those links, WebLogic Server extracts the session ID from the URL and finds the appropriate `HttpSession` object.

To ensure secure sessions with a uniformly random distribution, session IDs must contain a certain number of characters. The length of the session ID, however, can cause problems for many wireless devices because many devices limit URLs to 128 or fewer characters (bytes).

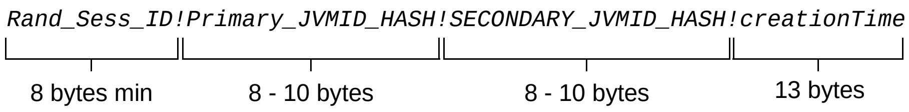
There are two ways to limit the length of the session ID:

- You can limit the length of the randomly generated portion of 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/docs61/webapp/weblogic_xml.html#session-descriptor`. The minimum value is 8 bytes.

- You can limit the length of the remaining portion (JMVID) of the session ID by setting the `WAPEnabled` attribute to `true` in the `<WebServer>` element in the `config.xml` file for your domain.

  For WebLogic Server 6.1 with no Service Packs applied, setting `WAPEnabled` to `true` shortens the session ID to just the randomly generated portion of the session ID, that is, no information about the primary and secondary servers is included in the session ID.

  For WebLogic Server 6.1 with Service Pack 1 (or greater) applied, setting `WAPEnabled` to `true` changes the session ID for an application deployed on a cluster of WebLogic Servers to the following format:


Rand_Sess_ID!Primary_JVMID_HASH!SECONDARY_JVMID_HASH!creationTime
8 bytes min
8 - 10 bytes
8 - 10 bytes
13 bytes

  The server maps the *`Primary_JVMID_HASH`* and *`SECONDARY_JVMID_HASH`* values to the primary and secondary server information stored on the server to retrieve the session data in the appropriate `HttpSession` object.

  Note that for both the long session ID format (`WAPEnabled` set to `false`) and the limited session ID format (`WAPEnabled` set to `true`), in-memory replication of state is used: session persistence is not a requirement in a WebLogic Cluster. If the serving WebLogic Server is not using in-memory replication, the JVMID will not be encoded in the URL. For more information about the `WAPEnabled` attribute, see config.xml Elements and Attributes at `http://e-docs.bea.com/wls/docs61/config_xml/mbeans.html`.

  The `creationTime` value refers to the current time when the session ID was generated. It is the return value of the `System.currentTimeMillis()` method.

A Better Alternative Session-Tracking Method

A better alternative is to include a WML postfield element containing the session ID along with any WML `go` element. In WML, the `go` element indicates navigation to a URL. The `go` element may contain one or more postfield elements. These elements specify information to be submitted to the origin server during the request. In the example WML code (with scriptlet) that follows, the session ID is obtained from the session and used to set the value of the `JSESSIONID` postfield.

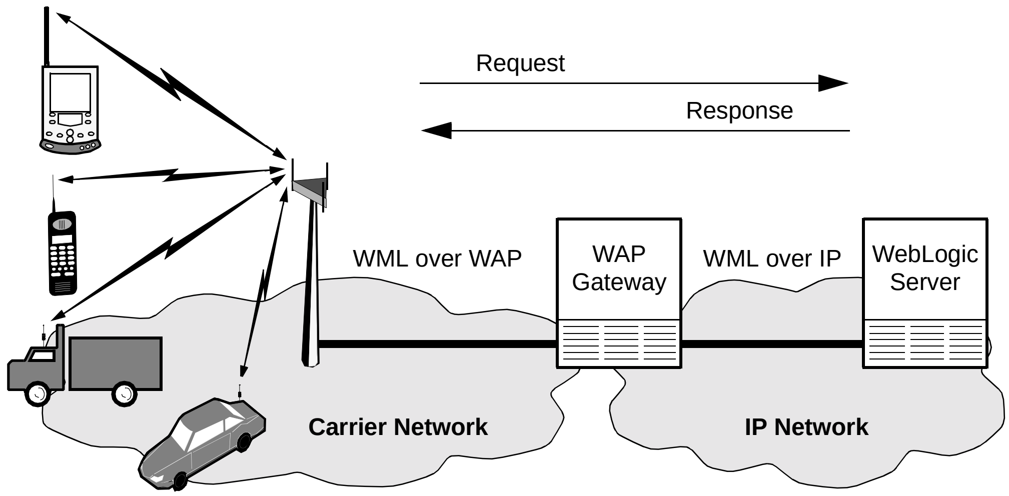
```
<go href="index.jsp" method="post"> <postfield name="JSESSIONID"
     value="<%= session.getId() %>"/></go>
```

The preceding code will cause an HTTP POST to the URL `index.jsp` with a message entity containing `JSESSIONID=sessionID`, where `sessionID` is the ID obtained from the `session.getId()` call. From within `index.jsp`, the session ID can then be obtained by getting the parameter from the HTTP request with the call `request.getParameter("JSESSIONID")`.

Writing WAP-Only Web Applications

The `date` and `phoneBook` applications are examples of WAP-only Web applications. These example applications, as clarified in the following figure, demonstrate how to access WebLogic Server from WML clients.

Figure 4-2 Data Flow for the WAP-Only Example Applications



Requests from WML 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 JSP or Java Servlets. For WAP applications, static files will be WML files, while JSPs and servlets will be used to generate WML dynamically.

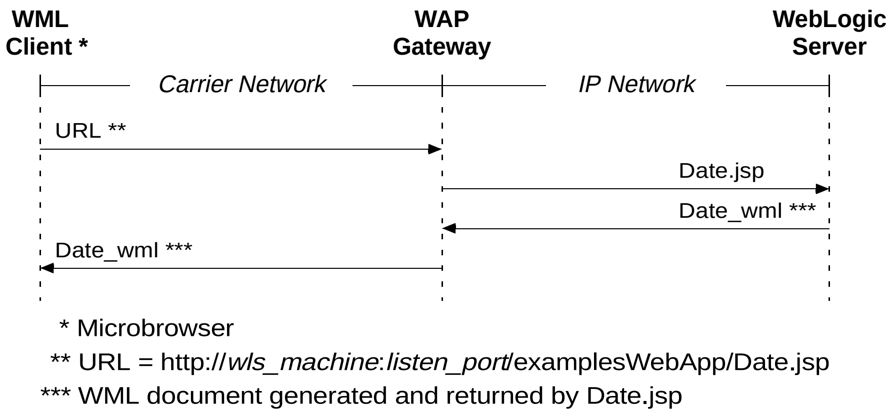
To create a WAP application, you write a WML front-end to support a single class of application clients: wireless devices having WML-enabled microbrowsers. Since WML is based on XML, you may want to see Programming WebLogic XML at `http://e-docs.bea.com/wls/docs61/xml/index.html` for additional examples of generating XML from within WebLogic Server.

The `date` and `phoneBook` example applications are relatively simple and are a good place to begin learning about how to develop WAP applications using the J2EE and WebLogic Server APIs. However, keep in mind that the examples demonstrate just one of several ways to put the WAP application model into practice and therefore should be taken as illustrative only.

Working with the date Example

The `date` example demonstrates generating a WML document from a JSP and serving it to a WML client. The following figure summarizes the data flow.

Figure 4-3 Accessing the date Application


WML Client *
WAP Gateway
WebLogic Server
Carrier Network
IP Network
URL **
Date.jsp
Date_wml ***
Date_wml ***
* Microbrowser
** URL = http://*wls_machine*:*listen_port*/examplesWebApp/Date.jsp
*** WML document generated and returned by Date.jsp

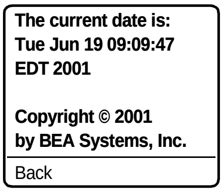
The WAP gateway converts WML requests received from the client to HTTP Servlet requests and then forwards the converted requests to `Date.jsp` running on WebLogic Server. `Date.jsp` responds by determining the current date and time and returning the results in the following WML document.

Listing 4-1 Date.jsp from date Directory

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

<!-- Copyright (c) 2001 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; 2001 by BEA Systems, Inc.
           All Rights Reserved.</small>
    </p>
  </card>
</wml>
```

The requesting WML client accesses the markup-language document and prints the current date and time, as illustrated in the following example display.

Figure 4-4 date Example Display



The current date is:
Tue Jun 19 09:09:47
EDT 2001

Copyright © 2001
by BEA Systems, Inc.

Back

The line `<%@ page contentType="text/vnd.wap.wml" %>` in `Date.jsp` sets the MIME types of the generated document to the WML MIME type; it is required whenever you are generating WML directly from a JSP or servlet. Without this line, the MIME type will default to the HTML MIME type, and the WAP gateway may attempt to translate the document into WML with unfavorable results.

Note: MIME, for Multipurpose Internet Mail Extensions, is a specification for enhancing the capabilities of standard Internet electronic mail. It offers a simple standardized way to represent and encode a wide variety of media types for transmission via Internet mail.

The MIME types required for the `date` example, identified in the following table, are already registered by WebLogic Server.

Table 4-1 MIME Type Definitions for the WAP-Only Example Applications

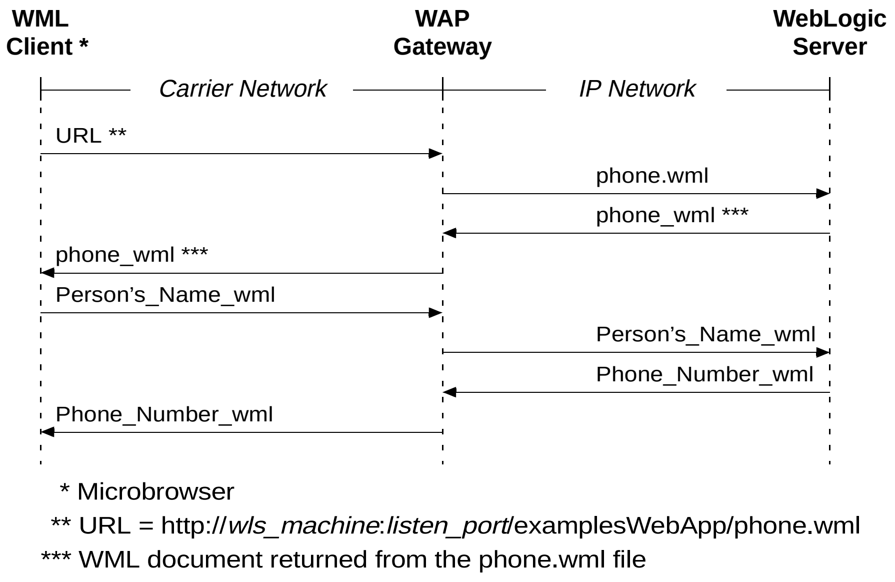
Extension	Mime Type	Description
`.wml`	`text/vnd.wap.wml`	WML source files
`.wmlc`	`application/vnd.wap.wmlc`	WML compiled files
`.wmls`	`text/vnd.wap.wmlscript`	WMLScript source files
`.wmlsc`	`application/vnd.wap.wmlscriptc`	WMLScript compiled files
`.wbmp`	`image/vnd.wap.wbmp`	WML bitmaps

You can view these MIME types in the `config/examples/applications/examplesWebApp/WEB-INF/web.xml` file of your WebLogic Server installation. The MIME types in this file are informational only: adding or deleting MIME types in this file will not change the MIME types registered with WebLogic Server.

Working with the phoneBook Example

The `phoneBook` example demonstrates processing data from a form received from a WML client. The following figure summarizes the data flow.

Figure 4-5 Accessing the phoneBook Application



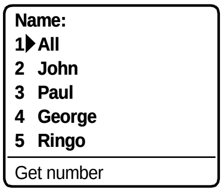
The `phone.wml` file, shown in the following listing, presents options to the user for looking up phone numbers via the `select` element.

Listing 4-2 phone.wml from phoneBook Directory

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

<!-- Copyright (c) 2000-2001 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/examplesWebApp/
           PhoneServlet?name=$(name:escape"/>
    </do>
  </card>
</wml>
```

The requesting WML client accesses the markup-language document and prints the options for looking up phone numbers, as illustrated in the following example display.

Figure 4-6 phoneBook Example Display



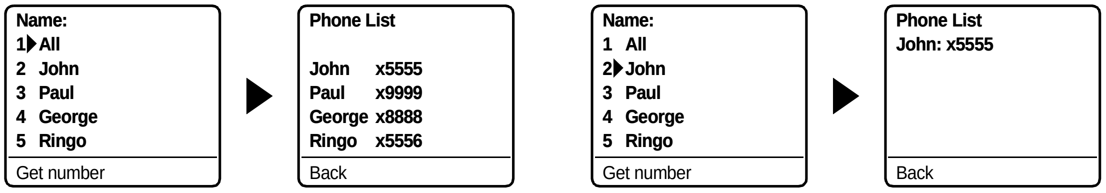
Name:
1▶All
2 John
3 Paul
4 George
5 Ringo
Get number

Based on the user's input (`All` or a name), the client initiates the following HTTP request through the WAP gateway to `PhoneServlet`—an existing servlet example located in the `samples/examples/servlets` directory of your WebLogic Server installation:

```
http://wls_machine:listen_port/examplesWebApp/phoneServlet?name=
    $(name:escape)
```

The query parameter `name` is added to the servlet's URL. In this example, `PhoneServlet` generates a WML response from a text file named `phonelist`, located in the `samples/examples/servlets` directory of your WebLogic Server installation, and the WAP gateway forwards the response to the WML client. The following figure helps clarify the `phoneBook` application flow and user input.

Figure 4-7 Using the phoneBook Application—Example



```
Name:
1▶All
2  John
3  Paul
4  George
5  Ringo
Get number
```

```
Phone List

John    x5555
Paul    x9999
George  x8888
Ringo   x5556
Back
```

```
Name:
1  All
2▶John
3  Paul
4  George
5  Ringo
Get number
```

```
Phone List
John: x5555
Back
```

Note: Although some WAP gateways can be configured to automatically translate HTML to WML, we at BEA strongly encourage you to generate WML directly because WML is designed to address the display limitations of most WML client devices.

Installing Additional Software for the WAP-Only Example Applications

You need to install the following additional software to run the `date` and `phoneBook` example applications:

- WML client device or suitable emulation software
- WAP gateway or suitable emulation software

For pointers to vendors who distribute WML client emulation software, see the `samples/examples/wireless/tools/emulators.txt` file in your WebLogic Server installation.

Running the WAP-Only Example Applications

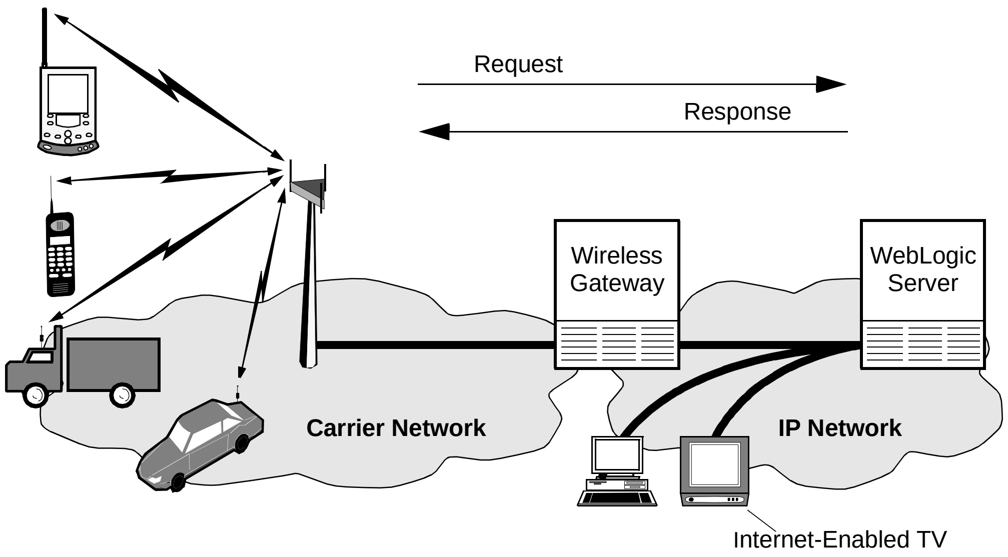
The examples Web application directory hierarchy is in the `config/examples/applications` directory of your WebLogic Server installation. When you build the `date` and `phoneBook` examples, the `config/examples/applications/examplesWebApp` directory will store the application archives created for the example applications.

For step-by-step instructions for building, configuring, and running the `date` or `phoneBook` example application, see the `package-summary.html` file in the target example directory (`date`, `phoneBook`) containing the source files for the example.

Writing Multi-Target Web Applications

The `helloWorld`, `stockDemo`, and `travelDemo` applications are examples of writing multi-target applications. These example applications, as clarified in the following figure, demonstrate how to access WebLogic Server from WML, HDML, cHTML, and HTML clients.

Figure 4-8 Data Flow for Multi-Target Example Applications



Each of the example applications (`helloWorld`, `stockDemo`, and `travelDemo`) has four front-ends supporting the following classes of application clients: devices having WML-enabled microbrowsers, devices having HDML-enabled microbrowsers, devices having cHTML-enabled microbrowsers (see note), and devices having

HTML-enabled microbrowsers or browsers. Writing multi-targeted applications allows an application developer to gain access to wireless subscribers while preserving previous engineering effort in standard Web technology.

Note: The examples are sufficiently simple that the HTML JSP pages for the examples also serve as cHTML JSP pages for the examples.

The `helloWorld`, `stockDemo`, and `travelDemo` applications are relatively simple and are a good place to begin learning about how to develop multi-target Web applications using the J2EE and WebLogic Server APIs. However, keep in mind that the examples demonstrate just one of several ways to put the multi-target application model into practice and therefore should be taken as illustrative only.

Understanding the Examples Framework

The `helloWorld`, `stockDemo`, and `travelDemo` examples use a simple, prototypical framework to select device-independent pages. When the examples framework is called to handle a device request, it determines the type of the requesting device by examining the HTTP `User-Agent` header of the request and then returns an object specifying which JSP page to use and the values to pass to the page. The examples framework is designed to use a single URL to generate content (WML, HDML, cHTML, HTML) according to the type of the requesting device.

Registering MIME Types

The MIME types required for the `helloWorld`, `stockDemo`, and `travelDemo` examples, identified in the following table, are already registered by WebLogic Server.

Table 4-2 MIME Type Definitions for the Multi-Target Example Applications

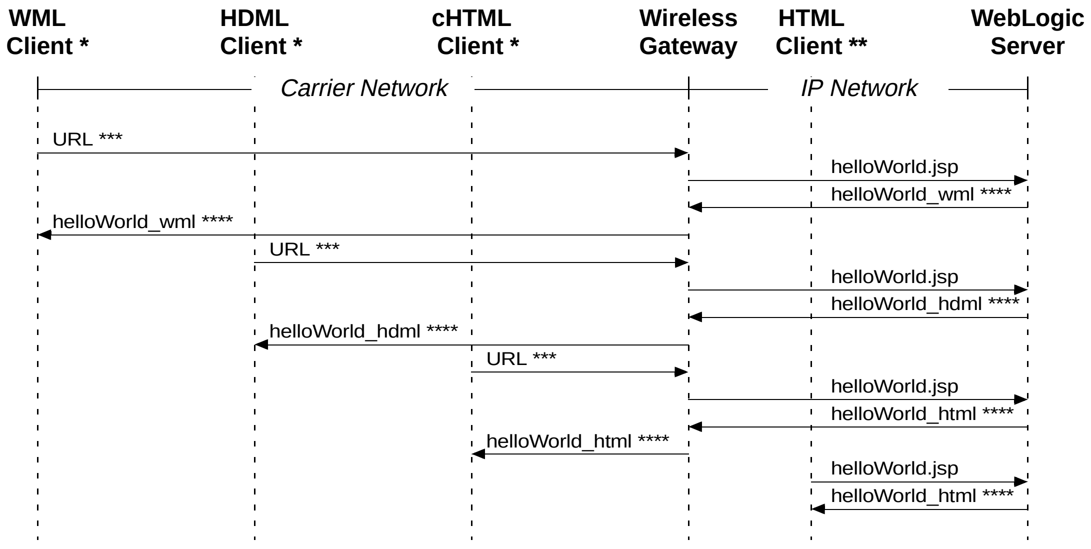
Extension	Mime Type	Description
`.wml`	`text/vnd.wap.wml`	WML source files
`.wmlc`	`application/vnd.wap.wmlc`	WML compiled files
`.wmls`	`text/vnd.wap.wmlscript`	WMLScript source files
`.wmlsc`	`application/vnd.wap.wmlscriptc`	WMLScript compiled files
`.wbmp`	`image/vnd.wap.wbmp`	WML bitmaps
`.hdml`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

You can view these MIME types in the `config/examples/applications/examplesWebApp/WEB-INF/web.xml` file of your WebLogic Server installation. The MIME types in this file are informational only: adding or deleting MIME types in this file will not change the MIME types registered with WebLogic Server.

Working with the helloWorld Example

The `helloWorld` example demonstrates (1) generating a simple markup-language document specific to the requesting device and (2) serving the generated document to the requesting device. The following figure summarizes the data flow.

Figure 4-9 Accessing the helloWorld Application



* Microbrowser ** Browser
*** URL = http://*wls_machine*:*listen_port*/helloWorld/Hello
**** Document generated and returned by markup-language specific helloWorld.jsp

For a request from a wireless client, the client's wireless gateway converts the markup-language request received from the client to an HTTP Servlet request and then forwards the converted request to `helloWorld`'s HTTP Servlet running on WebLogic Server. For a request from a desktop client, the desktop client sends an HTTP Servlet request directly to `helloWorld`'s HTTP Servlet running on WebLogic Server.

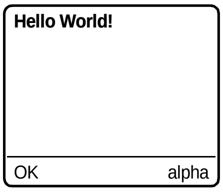
The HTTP Servlet passes the request to the examples framework for device-type and JSP-page resolution, and the examples framework returns the location of the appropriate `helloWorld.jsp` page in the `wml`, `hdml`, or `html` subdirectory of the `helloWorld` example directory. The selected `helloWorld.jsp` responds by generating and returning a markup-language specific document to the requesting device. The document, of which the WML-version is shown in the following listing, presents the user with the `Hello World!` string.

Listing 4-3 helloWorld.jsp from wml Subdirectory in helloWorld Directory

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

<!-- Copyright (c) 2001 by BEA Systems, Inc.
     All Rights Reserved. -->

<%@ page contentType=”text/vnd.wap.wml”%>

<wml>
  <template> <do type=”prev” label=”back”> <prev/></do> </template>
  <card id=”firstcard”>
    <p> Hello World!
    <br/> <a title=”next” href=”#secondcard”>Next</a>
    </p>
  </card>
  <card id=”secondcard”>
    <p> Goodbye from WML.</p>
  </card>
</wml>
```

The requesting device accesses the markup-language document and prints `Hello World!`, as shown in the following example display.

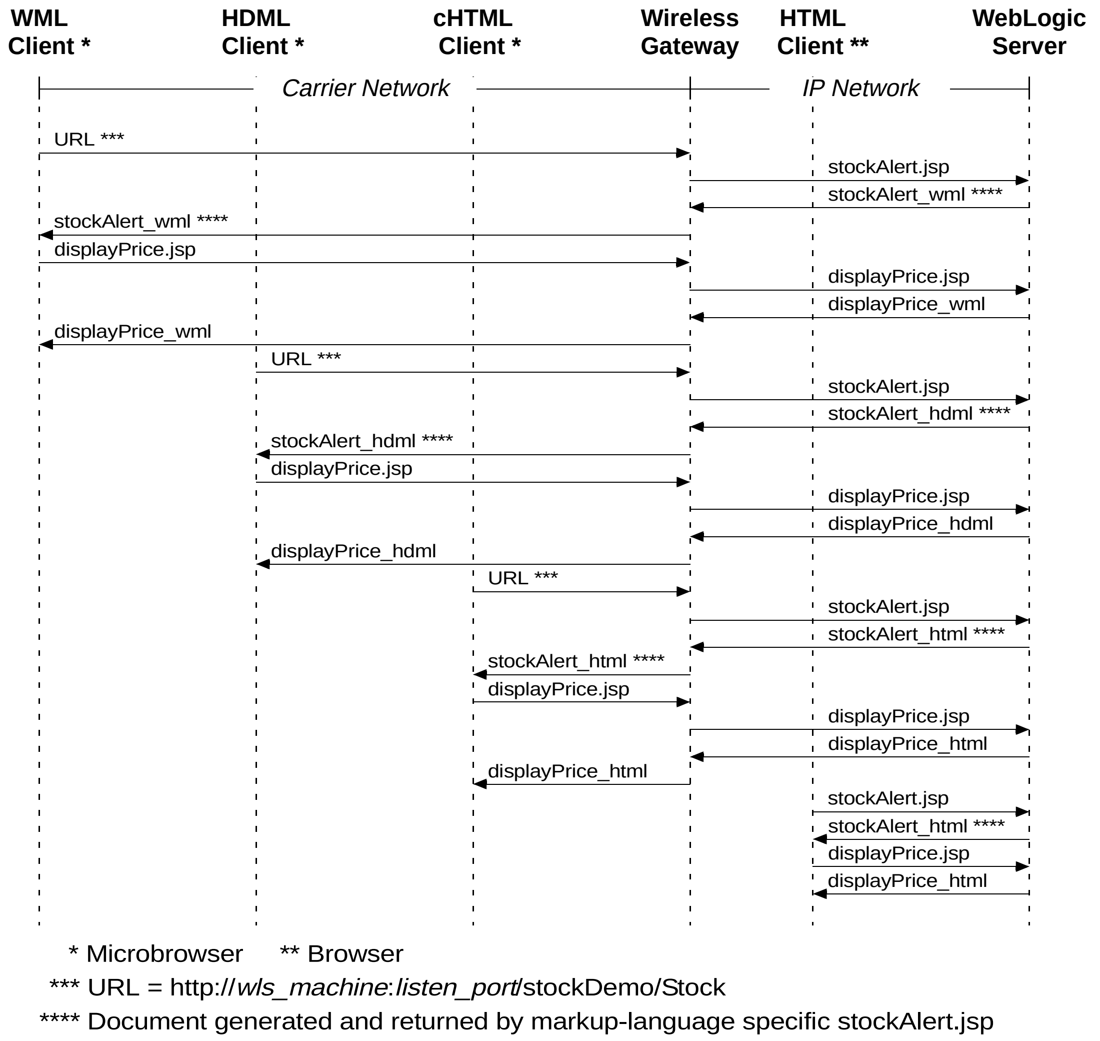
Figure 4-10 helloWorld Example Display



Working with the stockDemo Example

The `stockDemo` example application demonstrates processing data from a simple form received from a client. The following figure summarizes the data flow.

Figure 4-11 Accessing the stockDemo Application



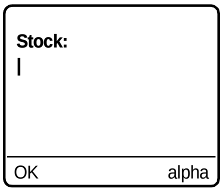
When `stockDemo`'s HTTP Servlet receives an HTTP Servlet request, it passes the request to the examples framework for device-type and JSP-page resolution, and the examples framework returns the location of the appropriate `stockAlert.jsp` page in the `wml`, `hdml`, or `html` subdirectory of the `stockDemo` example directory. The selected `stockAlert.jsp` responds by generating and returning a markup-language specific document to the requesting device. The document, of which the HDML-version is shown in the following listing, presents the user with a form containing a `Stock:` prompt for entering stock queries.

Listing 4-4 stockAlert.jsp from hdml Subdirectory in stockDemo Directory

```
<%@ page contentType="text/x-hdml"%>

<!-- Copyright (c) 2001 by BEA Systems, Inc.
     All Rights Reserved. -->

<HDML VERSION="3.0" TTL="0" PUBLIC="TRUE">

<ENTRY name="first" KEY="stock">
<ACTION TYPE="ACCEPT" TASK="GO"
     DEST="/stockDemo/StockAlert?stock=$(stock)">
Stock:
</ENTRY>

</HDML>
```

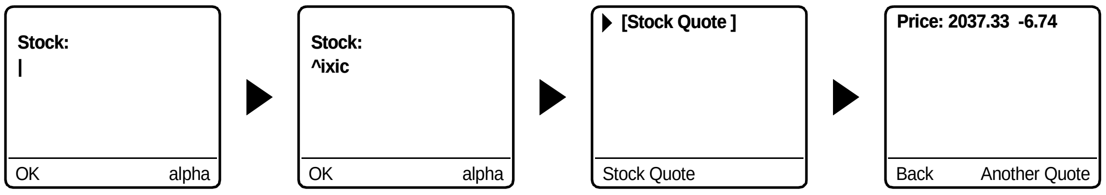
The requesting device accesses the markup-language document and prints the `Stock:` prompt and an empty entry field, as shown in the following example display.

Figure 4-12 stockDemo Example Display


Stock:
|
OK alpha

Based on the user's input (a stock name), an HTTP request is made to `stockDemo`'s HTTP Servlet, and a query parameter (`stock`) is added to the servlet's URL. The examples framework returns the location of the appropriate `displayPrice.jsp` page (in the `wml`, `hdml`, or `html` subdirectory of the `stockDemo` example directory) and the value of the requested stock (obtained from `YaHoo!`) to pass to the page. The selected `stockAlert.jsp` generates and returns a markup-language specific document to the requesting device, and the requesting device accesses the markup-language document and prints the document on its screen. The following figure helps clarify the `stockDemo` application flow and user input.

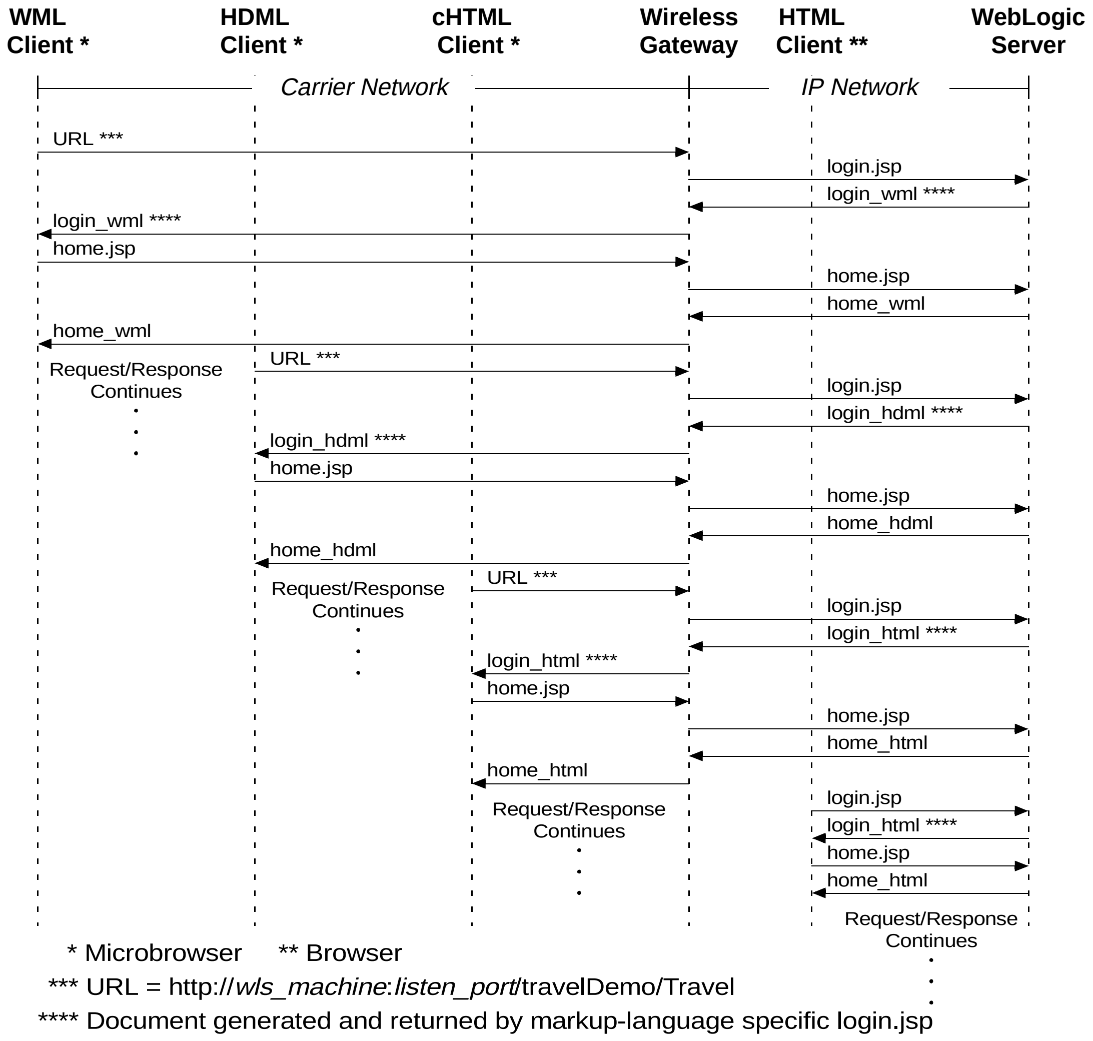
Figure 4-13 Checking a Stock Quote from a Cell Phone—Example



Working with the travelDemo Example

The `travelDemo` example application demonstrates processing data from a more complex form received from a client. The following figure summarizes the data flow.

Figure 4-14 Accessing the travelDemo Application



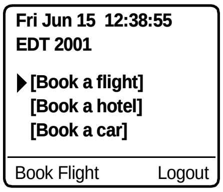
The `travelDemo` example application sends a series of form documents to a requesting device, starting with a login form. After the user successfully logs in using `x` as the username and `y` as the password, `travelDemo`'s `home.jsp` generates and returns a device-specific travel form document to the requesting device, of which the HTML-version is shown in the following listing.

Listing 4-5 home.jsp from html Subdirectory in travelDemo Directory

```
<html>

<!-- Copyright (c) 2001 by BEA Systems, Inc.
     All Rights Reserved. -->

<HEAD>
<TITLE>Home</TITLE>
<meta name=”palmcomputingplatform” content=”true”>
</HEAD

<body>
<h1>TravelDemo Homepage</h1>
<ul>
<li><a href=”<%=
     response.encodeURL(“/travelDemo/FindAFlight”)%>”>
          Book a flight</a>
<li><a href=”<%=
     response.encodeURL(“/travelDemo/NotImplemented”)%>”>
          Book a hotel</a>
<li><a href=”<%=
     response.encodeURL(“/travelDemo/NotImplemented”)%>”>
          Book a car</a>
<li><a href="<%=
     response.encodeURL("/travelDemo/NotImplemented")%>">
          View reservations</a>
<li><a href=”<%=
     response.encodeURL(“/travelDemo/NotImplemented”)%>”>
          View account</a>
<li><a href=”<%=
     response.encodeURL(“/travelDemo/Logout”)%>”>Logout</a>
</ul>
</body>
</html>
```

The requesting device accesses and prints the travel form document, as shown in the following example display.

Figure 4-15 travelDemo Example Display

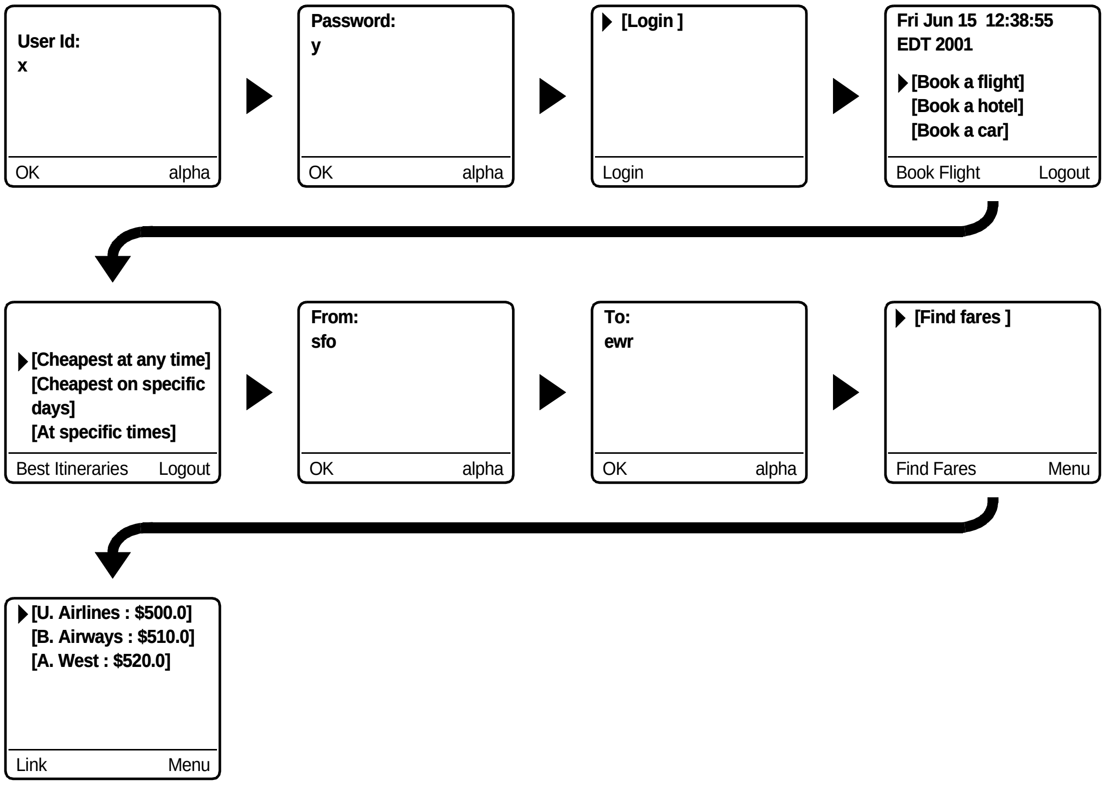


The document presents the user with choices for booking a flight, a hotel, or a car, or viewing reservations or an account, or logging out. The actual choices that display depend upon the type of requesting device.

Note: `Book a hotel`, `Book a car`, `View reservations`, and `View account` are not implemented for the `travelDemo` example application.

When a user chooses the `Book a flight` selection, a new screen appears showing the following selections: `Cheapest at any time` and `Cheapest on specific days`. When a user chooses one of these selections, the only valid user input for the ensuing `From:` prompt is `sfo` (for San Francisco International Airport) and the ensuing `To:` prompt is `ewr` (for Newark International Airport). The following figure helps clarify the `travelDemo` application flow and user input.

Figure 4-16 Booking a Flight from a Cell Phone—Example



Based on the user's input, `travelDemo` generates a response and forwards the response to the requesting device. The response is a "canned" response, meaning that a user cannot actually book a flight, a hotel, or a car, or view reservations or an account by accessing the `travelDemo` example application.

Installing Additional Software for the Multi-Target Example Applications

You need to install the additional following software to run the `helloWorld`, `stockDemo`, and `travelDemo` example applications:

- Wireless client devices (WML, HDML, cHTML, HTML) or suitable emulation software
- Wireless gateways or suitable emulation software

For pointers to vendors who distribute wireless client emulation software, see the `samples/examples/wireless/tools/emulators.txt` file in your WebLogic Server installation.

Running the Multi-Target Example Applications

The examples Web application directory hierarchy is in the `config/examples/applications` directory of your WebLogic Server installation. When you build the `helloWorld`, `stockDemo`, and `travelDemo` examples, the `config/examples` directory will store the exploded `war` files created for the example applications, and the `config/examples/applications` directory will store the `jar` file created for the examples framework.

For step-by-step instructions for building, configuring, and running the `helloWorld`, `stockDemo`, or `travelDemo` example application, see the `package-summary.html` file in the target example directory (`helloWorld`, `stockDemo`, `travelDemo`) containing the source files for the example. Building any of the examples also builds the examples framework.