Lutris®

Enhydra™

Wireless Application Developer's Guide

Copyright © 2000, 2001 by Lutris Technologies, Inc. All rights reserved.

No part of this book shall be reproduced, stored in a retrieval system, or transmitted by any means, electronic, mechanical, photocopying, recording, or otherwise, without written permission from Lutris Technologies, Inc. No patent liability is assumed with respect to the use of the information contained herein. Although every precaution has been taken in the preparation of this book, the author assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

The Lutris and Enhydra logos, Enhydra XMLC, Enhydra Enterprise, and InstantDB are trademarks or registered trademarks of Lutris Technologies, Inc. All other trademarks, trade names or company names referenced herein are used for identification only and are the property of their respective owners.

Sun, Sun Microsystems, the Sun logo, Solaris, Forte, Java, JavaScript, Java 2, JDBC, J2EE, iPlanet, and all Sun, Java, and iPlanet based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. UNIX® is a registered trademark in the United States and other countries, exclusively licensed through X/Open Company, Ltd. Windows, WinNT, Win32, and Access are registered trademarks of Microsoft Corp. InstallShield is a trademark of InstallShield Software Corp. Cygwin is a trademark of Cygnus Solutions Corp. Oracle is a trademark or registered trademark of Oracle Corp. Sybase is a trademark of Sybase Corp. Informix is a trademark of Informix Corp. Red Hat Linux is a trademark of Red Hat Corp. Linux is a registered trademark of Linus Torvalds. Netscape is a registered trademark of America Online, Inc. PostgreSQL is Copyright © 1996-2000 by PostgreSQL Inc. JBuilder™ and InterBase® are trademarks of Borland/Inprise. The Bluetooth trademarks are owned by Telefonaktiebolaget L M Ericsson, Sweden. All other product names mentioned herein are trademarks of their respective owners.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed trademarks. Where those designations appear in this book, and Lutris Technologies, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

Acknowledgements

Lutris Enhydra Development: Jason Abbott, Kyle Clark, Mark Diekhans, Larry Deran, Michael Gardner, Dick Gemoets, Scott Harrison, Craig Heath, Peter Hearty, Aidan Hosler, Wes Isberg, Andy John, Peter Johnson, Matthew Kalastro, Ray Kiuchi, Paul Mahar, John Marco, Shawn McMurdo, Ioan Mitrea, Paul Morgan, Christophe Ney, Robert Pirani, Scott Pirie, Joseph Shoop, Wayne Stidolph, Josh Sugnet, Simon Tuffs, Mike Ward.

Lutris Customer Services: Andy Ames, Debbie Brackeen, Peter Darrah, Jim Dumont, Jason Dunton, Anne Hopkins, Andrew Longsworth, Lindsey Lonne, Livia Peras, John Powell, Christopher Reed, Jane Richter, Katrina Seitz, Steve Slany, Daniel Thomas.

Lutris Consulting: Ashley Baumann, David Black, Dennis Chatham, Jon Coyle, Tola Dalton, Jay Gunter, John Hellier, Donna Karolchik, Bill Karwin, Alyssa Lalanne, Graham Moore, Jim Murphy, Thom Nelson, Natasha Perry, Kristen Pol, Lisa Reese, Matt Schwartz, Harvey Thompson, Robert Trama, Shiming Shi, David Simons, Jonathan Webb.

Lutris Marketing: Keith Bigelow, Scott Campbell, Lynda Hall, Holly Hamner, Klaus Krull, Helen Meservey, Lynn Renshaw, Greg Schwarzer, Gillian Webster, David Young.

Lutris Enhydra Documentation: Teresa Andrews, Ian Evans, Curtis Gavin, Laurel Kline, Michael Maceri, C. Rand McKinney.

Thanks also to the following Lutris departments: customer service, consulting, finance, legal, IS, marketing, quality assurance, research and development, sales, technical support, training, and the executive staff.

Printed in the U.S.A.
ENW-US042-35 1E0R1200
0102030405-9 8 7 6 5 4 3 2 1

1200 Pacific Avenue, Suite 300, Santa Cruz, CA 95060 ✦ Phone 831.471.9753 ✦ Fax 831.471.9754 ✦ http://www.lutris.com

Contents

Chapter 1
Introduction 1

What you should already know 1
Conventions used in this book 1
Lutris Enhydra document set. 3
Getting Started 3
Developer's Guide 4
Wireless Application Developer's Guide . . . 4
Lutris documentation updates available online . 4
Contacting Lutris Technical Publications 4
Where to find support and training for Lutris Enhydra 5
Lutris support. 5
Registering your product online 5
Contacting Lutris Technical Support 5
Submitting bug reports to Lutris Technical Support 5
Lutris training 6
Available training courses 6
Contacting Lutris Education Services . . . 6
Additional Enhydra information available on Enhydra.org. 6
Enhydra.org mailing lists 7
Mailing list archives 7
Enhydra.org working groups 7
Documentation working group 8
Enhydra.org community documentation . . . 8
Open-source software downloads 8
Acknowledgments. 8

Chapter 2
Overview 9

The wireless world. 9
Major technologies 10
Major markets 10
Wireless devices 11
Personal computers 11
PDAs and handheld computers 12
Palm OS 12
EPOC. 13
Pocket PC 13
Web-enabled phones 13
Microbrowsers 14
Java phones 15
Pagers 15
Standard telephones 15
Wireless technologies 16
Bearer network technologies 16
Analog technologies 17
Digital technologies 17
GSM. 18
PCS 19
Wireless Application Protocol 19
Controversy over WAP 21
i-mode 22
i-mode vs. WAP 22
Location-based services. 23
SMS 24
Third-generation (3G) technologies 25
Client presentation technologies. 26
HTML 26
WML 26
cHTML 27
XHTML 27
VoiceXML 27
J2ME and the MIDP 28
Where to find more information 29
Books 29
Websites 30
Periodicals 31

Chapter 3
Getting started with wireless application development 33

Enhydra for wireless applications. 33
Enhydra advantages. 33
Application development tools 34
Core development tools. 34
Wireless development tools. 34
First steps: Installing Lutris Enhydra and tools 35
Using the AirSent sample application 35
Step 1: Building AirSent 36
Option 1: Building from the command line 36
Option 2: Building using JBuilder and Kelp 37
Step 2: Running AirSent 39
Option 1: Running AirSent from the command line 39

Option 2: Running AirSent from JBuilder . . . 39
Step 3: Using AirSent . . . 39
Using AirSent Web pages . . . 40
Viewing AirSent wireless content . . . 41
Running and using the AirSent J2ME client application . . . 44
Step 4: Modifying AirSent . . . 46
Rebuilding from the command line . . . 47
Rebuilding using JBuilder . . . 47
Running the application . . . 47
Creating a new wireless application . . . 49
Using the Application Wizard with JBuilder and Kelp . . . 49
Building and running the application . . . 52
Using the Application Wizard independently . . . 53
Building and running the application from the command line . . . 55

Chapter 4
Designing wireless applications 57

General design considerations . . . 57
Key features . . . 57
Web profile . . . 57
Personalization . . . 58
Location-based features . . . 58
Meaningfully partitioned content . . . 59
Use patterns . . . 60
Personas and profiles . . . 60
Device constraints . . . 60
Interface guidelines . . . 61
Layout and design . . . 62
Designing a wireless storyboard . . . 62
Navigation . . . 62
Applications with multiple profiles . . . 62
Wireless technology limitations . . . 63
WAP . . . 63
File and display size . . . 63
Images . . . 63
i-mode . . . 63
General specifications . . . 63
File and display size . . . 63
Images . . . 64
Other limitations . . . 64

Chapter 5
Survey of wireless presentation languages 65

Extensible Markup Language (XML) . . . 65
Concepts and terminology . . . 66
Elements . . . 66
Attributes . . . 66
Entities . . . 66
Document Type Definitions (DTDs) . . . 66
Wireless Markup Language (WML) . . . 68
Document structure . . . 68
First WML document . . . 68
Meta elements . . . 69
Card element . . . 69
Events . . . 70
Example . . . 70
Tasks . . . 70
Navigation elements . . . 71
Using the Wireless Telephony Application Interface (WTAI) . . . 72
Formatting elements . . . 72
Variables . . . 73
Reserved characters . . . 73
Compact HTML (cHTML) . . . 74
Differences from HTML . . . 74
accesskey attribute . . . 74
Icon symbols . . . 74
HTML 1.0 elements . . . 75
General elements . . . 75
Layout . . . 75
Text formatting . . . 76
Form elements . . . 76
Link-related elements . . . 77
Lists . . . 77
HTML 2.0 elements . . . 77
Voice Extensible Markup Language (VoiceXML) . . . 78
Fundamental concepts and terminology . . . 78
Applications . . . 78
Documents . . . 78
Dialogs . . . 79
A first VoiceXML document . . . 79
Further concepts . . . 79
Subdialogs . . . 80

Sessions 80
Grammars 80
Events 80
Links 81
Example 81
VoiceXML Elements 81
Document elements 82
Form elements. 82
Menu elements 82
Event handlers 83
Directives 83
Extensible HyperText Markup Language (XHTML) 83
Differences between XHTML and HTML . . . 84
Lowercase elements and attributes 84
Proper nesting. 84
Element closure 84
Attributes 85
id attribute. 85
Script element 85
Mandatory elements 86
XHTML Basic elements 86
Structural and general elements. 86
Text-formatting elements 87
Hypertext and related elements. 87
List elements 87
Form elements. 88
Table elements. 88
Image and other elements 88

Chapter 6
Using wireless development tools 89
Enhydra tools. 89
JBuilder and Kelp. 89
Kelp 90
Enhydra application wizard 91
Forte for Java Community Edition 91
XMLC 92
What XMLC does 92
How to use XMLC 93
DODS 93
Wireless tools and device emulators 93
YoSpace SmartPhone emulator 93
Nokia WAP toolkit 95
WAP server simulator 95
Motorola iDEN SDK 96
Pixo Internet microbrowser 97
WAPtor 98

Chapter 7
Enhydra programming techniques 99
Using wireless DOMs 99
Specifying DOMs 99
Manipulating DOMs 100
Multi-profile presentation logic 100
Using writeDOM(). 100
Using the HTML DOM for XHTML documents 101
Presentation techniques 101
Session maintenance 101
Detecting device types 101
Disabling caching 102
Document technique 102
Programmatic technique 103
Character encoding for i-mode 103
Setting MIME types 103
J2ME/MIDP programming 104
Using XML with J2ME 104

Chapter 8
Understanding the AirSent sample application 105
The scenario 105
User roles. 105
The application 106
Application layers. 107
Storyboard 107
Data layer 108
Database schema. 108
Data objects. 109
Business layer 110
Sending email 110
Presentation layer. 110
Utility classes 110
Base classes. 111
Document object instantiation 111
Event handling. 112
Detecting device types 112
Displaying dynamic content 113
WAP profile 113
i-mode profile 113
Authentication. 113
J2ME profile 114
AirSent J2ME client 115

Appendix A
Using XSLT with Enhydra **117**
Overview 117
When to use XSLT 117
References 118
Example: Using XSLT with XMLC 118
Building and running the application 118
Document type definition 119
XML document 120
XSLT stylesheet 120
HTML page 121
Presentation object 122

Appendix B
Using the DiscRack wireless profiles **125**
Building and running wireless DiscRack 125
Using the YoSpace SmartPhone Emulator 126
Troubleshooting 127
Overview of the WAP profile 127

Appendix C
WML reference **129**
Document structure 129
Element reference 129
<a> 130
<access> 131
<anchor> 131
<b> 132
<big> 132

 132
<card> 132
<do> 133
 134
<fieldset> 134
<go> 135
<head> 136
<i> 136
<img> 136
<input> 137
<meta> 138
<noop> 138
<onevent> 139
<optgroup> 139
<option> 140
<p> 140
<postfield> 141
<prev> 141
<refresh> 141
<select> 142
<setvar> 143
<small> 143
 143
<table> 143
<td> 144
<template> 144
<timer> 145
<tr> 145
<u> 146
<wml> 146

Appendix D
cHTML reference **147**
Element reference 147
<!-- --> (comment) 147
<!DOCTYPE> 147
&xx; 148
<A> 148
<BASE> 148
<BODY> 149

 149
<CENTER> 149
<DD> 150
<DIR> 150
<DIV> 151
<DL> 151
<DT> 151
<FORM> 152
<HEAD> 152
<Hn> 153
<HR> 153
<HTML> 154
<IMG> 154
<INPUT> 155
<LI> 157
<MENU> 157
<META> 157
<OL> 158
<OPTION> 158
<P> 159
<PLAINTEXT> 159

<PRE> 159
<SELECT>. 160
<TEXTAREA> 160
<TITLE>. 161
<UL>. 161

Appendix E
VoiceXML reference 163

Element reference 163
<assign>. 164
<audio> 164
<block> 165
<break> 165
<catch> 166
<choice>. 167
<clear>. 168
<disconnect> 168
 168
<dtmf>. 169
<else> 170
<elseif> 170
<emp>. 170
<enumerate> 171
<error>. 172
<exit> 172
<field>. 173
<filled> 174
<form>. 175
<goto>. 176
<grammar> 176
<help>. 177
<if>. 178
<initial> 178
<link> 179
<menu> 180
<meta>. 180
<noinput>. 181
<nomatch> 181
<object> 182
<option>. 183
<param>. 183
<prompt> 184
<property> 185
<pros>. 186
<record>. 186
<reprompt>. 187
<return>. 187
<sayas>. 188
<script>. 189
<subdialog> 189
<submit>. 190
<throw> 191
<transfer> 191
<value>. 192
<var> 193
<vxml> 193

Appendix F
XHTML Basic reference 195

Element reference 195
<a> 195
<abbr> 197
<acronym>. 197
<address> 197
<base> 197
<blockquote> 198
<body>. 198

. 199
<caption>. 199
<cite> 199
<code> 199
<dfn> 199
<dd> 200
200
<dl>. 200
<dt>. 200
201
<form> 201
<head> 202
<hn> 202
<html> 202
<img>. 202
<input>. 203
<kbd>. 204
<label> 204
204
<link>. 205
<meta> 206
<object> 207
. 208
<option> 208
<p> 208
<param> 209
<pre> 209

<q> 209
<samp> 210
<select> 210
 210
 210
<table> 211
<td> 211
<textarea> 212
<th> 212
<title> 213
<tr> 214
 214
<var> 214

Glossary **215**

Index **223**

Chapter 1

Introduction

This book introduces wireless application development for the Lutris® Enhydra™ Application Server. It provides an introductory overview of wireless technology and devices, a step-by-step tutorial on building and running the AirSent sample application, along with an explanation of the AirSent application and some of the programming techniques it uses.

Additional chapters address issues in design of wireless applications, along with a survey of the most popular HTML- and XML-based languages for wireless applications. The book includes detailed language references for WML and cHTML. The online documentation includes additional reference material for VoiceXML and XHTML.

What you should already know

This book assumes you have the following background:

- You have read *Getting Started with Lutris Enhydra*, and are familiar with the fundamental aspects of Enhydra application development. Ideally, you will also have read the *Developer's Guide*.
- You are familiar with Java programming and Java servlets.
- You have some basic knowledge of UNIX commands and the UNIX `make` utility. This is not necessary if you are developing your application with Kelp in an IDE such as JBuilder.

Conventions used in this book

The typographical conventions used in this book are listed in Table 1.1.

Table 1.1 Typographical conventions

Convention	Description
Italics	Indicates variables, new terms and concepts, and book titles. For example, • A *servlet* is a Java class that dynamically extends the functionality of a Web server.

Table 1.1 Typographical conventions (continued)

Convention	Description
`Fixed-width`	Used to indicate several types of items. These include: • Commands that you enter directly, code examples, utility programs, and options. For example, ◦ `cd mydir` ◦ `System.out.println("Hello World");` ◦ `make` utility ◦ `-keep` option • Java packages, classes, methods, objects, and other identifiers. For example, ◦ `ErrorHandler` class ◦ `run()` method ◦ `Session` object **Note:** Method names are suffixed with empty parentheses, even if the method takes parameters. **Note:** Only specific references to object names are in fixed-width; generic references to objects are shown in plain text. • File and directory names. For example: ◦ `/usr/local/bin` **Note:** UNIX path names are used throughout and are indicated with a forward slash (`/`). If you are using the Windows platform, substitute backslashes (`\`) for the forward slashes (`/`).
`Fixed-width italic` and *`<Fixed-width italic>`*	Indicates variables in commands and code. For example, • `xmlc [`*`options`*`|`*`optfile`*`.xmlc ...] `*`docfile`* **Note:** Angle brackets (`< >`) are used to indicate variables in directory paths and command options. For example, • `-class <`*`class`*`>`
Boldface	Used for the words **Note**, **Tip**, **Important**, and **Warning** when they are used as headings that draw your eye to essential or useful information.
Keycaps	Used to indicate keys on the keyboard that you press to implement an action. If you must press two or more keys simultaneously, keycaps are joined with a hyphen. For example, • *Ctrl-C*.
`|` (pipe)	Used as a separator in menu commands that you select in a graphical user interface (GUI), and to separate choices in a syntax line. For example, • File|New • `{a|b|c}` • `[a|b|c]`
{ } (braces)	Indicates a set of required choices in a syntax line. For example, • `{a|b|c}` means you must choose `a`, `b`, or `c`.
[ ] (brackets)	Indicates optional items in a syntax line. For example, • `[a|b|c]` means you can choose `a`, `b`, `c`, or nothing.

Table 1.1 Typographical conventions (continued)

Convention	Description
. . . (horizontal ellipses)	Used to indicate that portions of a code example have been omitted to simplify the discussion, and to indicate that an argument can be repeated several times in a command line. For example, • `xmlc [options|optfile.xmlc ...] docfile`
plain text	Used for URLs and generic references to objects. For example, • http://www.lutris.com/documentation/index.html • The presentation object is in the presentation layer.
ALL CAPS	Indicates SQL statements. For example: • CREATE statement

Table 1.2 lists additional conventions used in this book, including the convention used to describe the Enhydra root directory, platform-related conventions, and so on.

Table 1.2 Additional conventions

Convention	Description
Enhydra root directory	When you install Enhydra, you install the Enhydra executables and libraries in a directory of your choosing. This directory is referred to as the Enhydra root directory or `<enhydra_root>`.
Paths	UNIX path names are used throughout and are indicated with a forward slash (/). If you are using the Windows platform, substitute backslashes (\) for the forward slashes (/). For example, • `/usr/local/bin`
URLs	URLs are indicated in plain text and are generally fully qualified. For example, • http://www.lutris.com/documentation/index.html
Screen shots	Most screen shots reflect the Microsoft Windows look and feel.

Lutris Enhydra document set

The Lutris Enhydra documentation set is an excellent resource for information about Enhydra. The documentation set includes the following printed guides.

Note Online versions of these books in both PDF and HTML formats are provided with the purchase of Lutris Enhydra. These online books, along with additional Enhydra online documentation, are located in the `doc` subdirectory of the directory in which you installed Lutris Enhydra. You can also view the online books and installation instructions directly from the product CD.

Getting Started

Getting Started with Lutris Enhydra introduces the fundamentals of Enhydra. The purpose of this book is to introduce Lutris Enhydra and provide a groundwork for

understanding and working with Enhydra and its associated tools. It includes a detailed tutorial and an explanation of the Enhydra DiscRack sample application.

Note As part of our commitment to support the Enhydra and open-source communities, Lutris Technologies has made the latest online version of *Getting Started with Lutris Enhydra* available for free viewing and download from the Lutris Documentation home page at http://www.lutris.com/documentation/index.html.

Developer's Guide

The Lutris Enhydra *Developer's Guide* introduces advanced topics and explores key features of Enhydra in detail. The purpose of the *Developer's Guide* is to provide developers with the information they need to create and debug sophisticated Enhydra applications. This guide provides in-depth information on the Lutris Enhydra development tools:

- Application Wizard
- Multiserver Administration Console
- Kelp tools
- Enhydra™ XMLC
- Data Object Design Studio (DODS)
- InstantDB
- Enhydra Director

Note The *Developer's Guide* is available only with the purchase of Lutris Enhydra.

Wireless Application Developer's Guide

The Lutris Enhydra *Wireless Application Developer's Guide* presents information on wireless technologies and describes how to develop wireless applications with Enhydra. It includes a detailed tutorial and an explanation of the Enhydra AirSent wireless sample application.

Note The *Wireless Application Developer's Guide* is available only with the purchase of Lutris Enhydra.

Lutris documentation updates available online

The latest product documentation updates and release notes are available to registered users from the Lutris Documentation home page at http://www.lutris.com/documentation/index.html.

Contacting Lutris Technical Publications

We strongly encourage you to send us your feedback because it helps us understand your needs and makes our documentation even better. You can submit feedback from the Lutris website at

http://www.lutris.com/documentation/feedback/index.html. You can also submit feedback by sending email to documentation@lutris.com.

Where to find support and training for Lutris Enhydra

Lutris Enhydra includes a package of products for developing Enhydra applications, including open-source products. Lutris Technologies, Inc. provides support and services for Lutris Enhydra.

Note Open-source communities or commercial entities support the other products. For detailed information on the available support options for those products, please refer to the appropriate group or company website.

Lutris support

Lutris offers a variety of support programs designed to assist you with your technical support needs. We can help with installing and using your Lutris product, developing and debugging your code, maintaining your deployed applications, providing onsite consulting services, and delivering enterprise-level support. For more information about any of the Lutris technical support programs, see the Lutris Support home page at http://www.lutris.com/support/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590).

Registering your product online

Lutris strongly encourages you to register your product online. Registering your product entitles you to 15 days of free installation support and provides you with the option of purchasing Lutris Support Services.

To register online, browse to the product registration form that is available at http://www.lutris.com/register.html.

Contacting Lutris Technical Support

For more information about any of Lutris' technical support programs, see the Lutris Support home page at http://www.lutris.com/support/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590). You can also send email to support@lutris.com.

Submitting bug reports to Lutris Technical Support

To report suspected Lutris Enhydra bugs, fill out the Bug Report form available at http://lutrisbugs.custhelp.com/cgi-bin/lutrisbugs/people. We recommend that you choose the Search Bugs link before submitting a bug report so that you can see if your bug has already been reported. Be sure to include steps-to-reproduce, exact error messages, and code snippets, if applicable, to help us better evaluate your report.

Lutris training

Lutris wants to ensure your success. Our expert trainers guide participants through hands-on labs designed to provide an intensive learning environment where participants quickly learn how to maximize Lutris Enhydra in development and deployment environments.

Available training courses

The following courses are currently offered by Lutris Technologies. For more information about training offerings, see the Lutris Training home page at http://www.lutris.com/training/index.html.

Lutris Enhydra Fundamentals

Lutris Technologies currently offers a five-day, instructor-led course titled *Lutris Enhydra Fundamentals*. This course is intended primarily for developers, architects, project managers, IT staff, and consultants who will be using Lutris Enhydra or are evaluating it for future projects.

Building Wireless Applications

Lutris Technologies currently offers a two-day, instructor-led course titled *Building Wireless Applications with Lutris Enhydra*. This course is intended primarily for Enhydra developers who want to create applications that serve content to cellphones or other wireless devices.

Lutris Enhydra Database Techniques

Lutris Technologies currently offers a two-day, instructor-led course titled *Database Techniques with Lutris Enhydra*. This course is intended primarily for Enhydra developers working on applications that require existing database platform support, Java database specialists and DBAs responsible for maintaining the data layer of an Enhydra application, and evaluators interested in seeing the database capability available through Enhydra.

Contacting Lutris Education Services

For more information about training offerings, see the Lutris Training home page at http://www.lutris.com/training/index.html or call Lutris Customer Service toll-free at 1-877-688-3724, Monday–Friday, 8 a.m.–6 p.m. Pacific Time (outside of North America, please call 1-831-460-7590). You can also send email to training@lutris.com.

Additional Enhydra information available on Enhydra.org

You can find a variety of information about open-source Enhydra at the Enhydra website: http://www.enhydra.org. The Enhydra website is the home of the Enhydra open-source community, one of Enhydra's greatest assets. The Enhydra community

consists of numerous entities, including community sponsors, technology providers, users, and of course developers.

Enhydra.org mailing lists

The Enhydra.org website includes archives of the various electronic mailing lists that serve as the backbone of the Enhydra community, as well as instructions on how to subscribe to the mailing lists.

Lutris encourages you to join one or more of the following Enhydra email lists:

- **Enhydra@enhydra.org**

  The Enhydra mailing list for developer interaction. The Enhydra project team monitors this list. It is the ideal place to get answers to your questions from fellow Enhydra developers.

- **Enhydra-daily@enhydra.org**

  A daily collection of all mail sent to enhydra@enhydra.org is sent to subscribers of this list.

- **Enhydra-digest@enhydra.org**

  A weekly digest of all mail sent to enhydra@enhydra.org.

- **EnhydraEnterprise@enhydra.org**

  The Enhydra Enterprise mailing list is tailored for those who are developing and deploying Enhydra applications on a large scale. Here you can find answers to the more detailed Enhydra questions, such as those on Enterprise Java Beans (EJB) and the Common Object Request Broker Architecture (CORBA).

- **EnhydraEnterprise-digest@enhydra.org**

  A weekly digest of all mail sent to EnhydraEnterprise@enhydra.org.

- **Enhydra-announce@enhydra.org**

  The mailing list for receiving Enhydra announcements.

For information and instructions on joining one or more of these lists, go to http://www.enhydra.org/community/mailingLists/index.html.

Mailing list archives

You can search the combined Enhydra mailing list archives at http://www.enhydra.org/community/mailingLists/index.html.

Enhydra.org working groups

Enhydra *working groups* bring together developers interested in creating new Enhydra applications and contributing new technologies or bug fixes for Enhydra.

Each working group provides access to the current project source code and to the project email list. This lets you communicate with the project leaders and other developers.

For information and instructions on joining one or more of these groups, go to http://www.enhydra.org/project/workingGroups/index.html.

Documentation working group

The documentation working group is focused on facilitating developer-created documentation for open-source Enhydra and related technologies. The working group also provides a central point for documentation discussions and proposals.

Community members are encouraged to submit and collaborate on articles of any length, on topics of general interest to all Enhydra developers—from beginning to advanced.

For information and instructions on joining this groups, go to http://www.enhydra.org/project/workingGroups/index.html.

Enhydra.org community documentation

The Enhydra website also has documentation provided by members of the community. For more information on this documentation, see http://www.enhydra.org/software/documentation/enhydra/index.html.

Open-source software downloads

You can download the latest version of open-source Enhydra and other related software at: http://www.enhydra.org/software/downloads/index.html.

Acknowledgments

The following people provided valuable information and feedback on this book during its development: Keith Bigelow, Mark Beaulieu, John Hellier, Matthew Kalastro, Klaus Krull, Alyssa Lalanne, Denise Lee, Christopher Reed, Joseph Shoop.

As an open-source product, Enhydra benefits from the contributions of many developers around the world. Lutris would like to thank the members of the Enhydra community, particularly those who contributed information used in some form in this book.

Chapter 2

Overview

This chapter provides an overview of wireless networks, devices, and applications. For an introduction to developing wireless applications with Lutris Enhydra, see Chapter 3, “Getting started with wireless application development.”

The wireless world

In the 1990s, the Internet changed everything by providing instantaneous access to information across the globe via Web browsers, along with ubiquitous communication through email, chat, and similar technologies. At the same time, cellular telephones (“cellphones”) became increasingly popular and handheld computers began to proliferate. The collision of these two trends is creating a new world of wireless Internet applications, accessible from cellphones, handheld computers, pagers, and other mobile devices.

Figure 2.1 Wireless Internet technologies around the world

* Figures are for total celluar subscriptions as of December 2000

While wireless technology is rapidly evolving, a variety of technologies are in widespread use today. A wide array of standards and technologies—promoted by many different companies and alliances—are competing for a projected multibillion dollar market.

Major technologies

The section "Wireless technologies" on page 16 discusses wireless technologies in depth. In a nutshell, though, the major wireless Internet technologies in use today are:

- Wireless Application Protocol (WAP), which uses Wireless Markup Language (WML), a presentation language invented specifically for cellphones and other mobile devices. WAP is prevalent in Europe and the U.S. For more information, see "Wireless Application Protocol" on page 19.
- Handheld Device Markup Language (HDML), a precursor to WML developed by Phone.com, Inc. Although still popular in the U.S., HDML is rapidly being replaced by WML.
- i-mode, which uses compact HTML (cHTML), a subset of standard HTML for cellular phones. i-mode is prevalent in Japan. For more information, see "i-mode" on page 22.
- Short Message Service, a text messaging system for cellphones, pagers, and other mobile devices. SMS is very popular in Europe. For more information, see "SMS" on page 24.

Major markets

Although wireless networks are increasingly expanding worldwide, the major wireless markets—particularly for cutting-edge technologies—are Europe, Japan, and the U.S. Although the U.S. initially led innovation in analog cellular systems, it has since lagged behind Japan and Europe, in part due to multiple conflicting standards.

As illustrated in Figure 2.1, the most prevalent wireless Internet technologies are:

- United States: HDML and WAP
- Europe: WAP and SMS
- Japan: i-mode; WAP is also starting to make inroads

All these technologies are young, so it is too early to know how standards will evolve and which technologies and systems will dominate.

Lutris Enhydra provides sufficient flexibility to develop applications that are adaptable to multiple presentation technologies and wireless systems; for more information, see Chapter 4, "Designing wireless applications."

Wireless devices

Wireless technology is advancing rapidly. New devices and software are being developed and announced virtually every day. While there is a great variety of wireless devices, from the perspective of an application developer, they all fall under the broad rubric of *end clients* in the wireless network. That is, they provide the end-user interface, much as the PC Web browser does in a traditional Web application.

For the purpose of this discussion, a wireless device is any device that can connect to the Internet without a direct "wireline" connection, including:

- Personal computers (PCs)
- Handheld computers
- Web-enabled cellphones
- Pagers
- Standard telephones

Personal computers

You probably don't think of a PC as a wireless device, but increasing hardware miniaturization and the advent of wireless modems have brought PCs into the wireless world. Any notebook or subnotebook computer equipped with a wireless Internet connection becomes a mobile wireless device by definition. Even a desktop PC can be a wireless device (though immobile), if it has a wireless connection to the Internet.

Figure 2.2 A typical notebook PC

The emerging *Bluetooth*™ standard defines a standard for short-range wireless systems operating within a ten-meter range. It enables a PC to connect wirelessly to a phone or other wireless device, increasing interoperability. For example, Bluetooth will make it easier to upload your email address book from your PC to your cellphone or handheld computer.

For more information on Bluetooth, see http://www.bluetooth.com.

PDAs and handheld computers

Personal digital assistants (PDAs), sometimes referred to as palm-top computers, are increasingly popular as personal organizers and calendars. Many new models also have the ability to connect wirelessly to the Internet. They use handwriting recognition for input, though some models have tiny keyboards, and most can connect to a full-size keyboard for text entry.

As PDAs become more capable and less expensive, they are rapidly gaining popularity. Worldwide sales of PDAs are projected to reach 6 million units in 2000, and more than 15 million in 2003.

Figure 2.3 Personal Digital Assistant (PDA)

PDAs running the Palm OS® are by far the most popular. Palm Computing makes devices that run Palm OS, and many companies licence the Palm OS, such as Symbol, Sony, Handspring, and IBM. Other PDA operating systems include Microsoft Pocket PC and BlackBerry™ from Research in Motion (RIM).

Handheld computers are lightweight portable computers, similar to laptop PCs, but smaller. Handheld computers have small keyboards for input, and generally have more PC-like functionality. Larger models have close to full-size keyboards, and compete with subnotebook PCs. The two major handheld computer operating systems are EPOC and Pocket PC.

Palm OS

The Palm VII (released in 1999) was the first mass-market handheld computer to include wireless communications capabilities. Currently, the Palm OS is the leading platform for handheld computers, commanding 70 to 80 percent of the worldwide market. One of the reasons for its popularity is simplicity: It does only a few specialized tasks, but it does them very well.

The Palm OS graphical user interface uses several common UI widgets such as menus, buttons, check boxes, and scroll bars. You select items on the screen using a pen (or *stylus*), and enter text using a graphical keyboard or through the built-in Graffiti® character recognition system.

The Palm platform includes several options for wireless Web applications, including:

- Web clipping applications, documents written in a subset of HTML 3.2 and served by the Palm.Net™ service
- WAP applications, which use a WAP browser from a company such as AvantGo (http://www.avantgo.com)

For more information on Palm OS, see http://www.palmos.com.

EPOC

EPOC is a 32-bit, multithreading operating system optimized for wireless devices. Originally developed by Psion, EPOC is now controlled and promoted by the Symbian consortium, consisting of Nokia and Ericsson, Motorola, Matsushita, and Psion.

The C language was originally the primary development tool for EPOC. In 1999, Symbian created a JDK for EPOC and indicated its intent for Java to become the primary EPOC development tool. EPOC also supports WAP for wireless Web access.

For more information on EPOC and Symbian, see http://www.symbian.com.

Pocket PC

Microsoft's Pocket PC operating system is a scaled-down version of Windows for non-PC devices, which Microsoft calls "information appliances." Originally called Windows CE (consumer electronics), it has been successful in some vertical industry applications, but far less popular than Palm OS in the PDA market. In an attempt to reposition the product, Microsoft renamed Windows CE version 3.0 to Pocket PC. Microsoft has also introduced HandheldPC, an operating system for mobile devices with larger displays and greater capability than PDAs (approaching subnotebook PCs).

The primary difference between Pocket PC and competitors such as the Palm OS is that Pocket PC is a 32-bit, multithreaded, multitasking operating system. However, Pocket PC has a number of drawbacks that have hindered its market acceptance, including its memory and power requirements.

Pocket PC's similarity to desktop versions of Windows is a two-edged sword. While it provides a degree of familiarity to new users, it may not always be the most suitable interface for mobile devices. However, many software developers have experience with the Windows API and tools such as Visual C++ and Visual Basic, which may encourage its acceptance among desktop software developers.

For more information on Microsoft's mobile products, see http://www.microsoft.com/MOBILE.

Web-enabled phones

The most well-known wireless device is the Web-enabled cellphone, sometimes referred to as a *Web phone*. This broad category of devices generally encompasses digital phones that can connect to the Internet through a wireless network.

Figure 2.4 Web-enabled cellphone

In addition to standard voice telephone capability, Web phones can do one or more of the following:

- Browse the Internet with a built-in microbrowser
- Send faxes
- Send and receive SMS messages and email

Web phones that have PDA capability built in are sometimes referred to as *smartphones*, though the distinction is not well defined. Major manufacturers of Web phones include Motorola, Ericsson, Nokia, Qualcomm, NeoPoint, and others.

Microbrowsers

The software that actually renders Web content on a Web phone is known as a *microbrowser*. It is analogous to a standard Web browser running on a PC. While there are many different microbrowsers on the market today, only a few dominate the market.

The UP.browser™ from Phone.com (now OpenWave System, Inc.) is by far the most popular microbrowser. It supports HDML or WAP, depending on the version. According to Phone.com:

> UP.Browser has been released in over 80 distinct phone models and over 200 new models are currently in development for all major digital standards, including CDMA, GSM, PDC, PHS, and TDMA. This footprint covers 95% of the global handset market.

WAP microbrowsers are also marketed by Nokia, Neomar, and others.

In late 1999, Microsoft launched Mobile Explorer,™ a dual-mode WML/HTML microbrowser for the Windows CE platform. It includes support for location-based services, WAP's WTLS security protocol, and a Java virtual machine. It also supports applications created with the SIM Toolkit, part of GSM.

An alternative to WAP microbrowsers, the Pixo 2.0 microbrowser can display both HTML and cHTML, and supports HTTP 1.0.

Java phones

Some new cellphones include a Java virtual machine that enables them to run small Java applications, using Java™ 2 Micro Edition (J2ME). One advantage of such devices is that they can theoretically run the same Java applications as other J2ME devices, thus saving development time. For more information on J2ME, see "J2ME and the MIDP" on page 28.

Pagers

Originally, pagers were known as "beepers," and did one thing: alerted you that somebody needed you. The industry has evolved considerably, with the proliferation of *two-way pagers*. Two-way pagers can send and receive email and connect to the Web. These sophisticated devices have small keypads, and are blurring the distinction between pagers and PDAs.

Figure 2.5 Advanced two-way pager (courtesy Research In Motion, Inc.)

Two-way pagers use paging networks such as Mobitex or Motorola's DataTAC network. Major manufacturers of advanced pagers include Motorola and Research in Motion (RIM). For more information, see

- http://www.blackberry.net
- http://www.motorola.com/developers/wireless

Standard telephones

Although you may not think of a standard telephone (a telephone with only voice capability) as a Web-access device, advances in speech technology are turning it into one. This new field is sometimes referred to as the "voice Web."

Two fundamental speech technologies enable the voice Web:

- Speech recognition, which provides user input capability
- Text-to-speech, which provides "presentation" capability

Together, these capabilities enable applications which the user navigates by speaking commands, rather than by typing or clicking a mouse; the application then uses computer-generated speech for its output.

Since speech is the most natural means of human communication, such applications could potentially open up vast new markets. Access to voice applications requires only plain old telephone service (POTS), which is close to universal in major markets. Pundits predict that eventually, most Web content will be accessible from any telephone.

Some of the major voice Web applications are:

- Voice portals, which enable access to Web content such as sports scores, stock quotes, news, weather, and so on
- Automated call centers, for customer-support applications
- Handsfree automobile applications, in which a driver can access directions and other information; new laws forbidding cellphone use in cars may popularize these applications

The primary standard for text-to-speech is VoiceXML, an XML application tailored for voice applications. For more information on VoiceXML, see "VoiceXML" on page 27.

Wireless technologies

An in-depth discussion of wireless systems and standards is beyond the scope of this book. However, it is useful to have a basic knowledge of the underlying technologies, to be able to discuss the medium meaningfully. This section introduces the current major wireless technologies and standards.

Bearer network technologies

Like other networking technology, wireless networks are divided into a hierarchy of protocols and standards. The lowest level in the hierarchy is known as the *bearer network*, which defines the underlying communication technology. Bearer networks can be either analog or digital. Both types of networks support voice communication, but digital networks support Internet communication.

Table 2.1 summarizes the most popular bearer network technologies, which can be broadly grouped into three generations.

Table 2.1 Bearer network technologies

Technology		Features
First-generation (1G)	AMPS	Analog voice
	TACS	No data services
Second-generation (2G)	CDMA	Digital voice, 9.6 Kilobits per second to 14.4 Kilobits per second
	TDMA	Dial-up for data connection (not always on)
	GSM	
	PCS	
	iDEN	
Third-generation (3G)	WCDMA	Digital voice
	EDGE	Up to 2 Megabits per second
	cdma2000	Data connection always on
		Broadband multimedia features
		Enhanced roaming

Analog technologies

First-generation analog cellular systems initially developed in the late 1970s were based on standards such as Advanced Mobile Phone Service (AMPS) and Total Access Communications System (TACS). These standards use Frequency Division Multiple Access (FDMA), in which each user conversation is allocated a pair of frequencies. FDMA systems use bandwidth inefficiently, and carriers are rapidly replacing them with digital systems.

Although many of the phones in use today are older analog phones, almost all new phones sold are digital; additionally, many analog cellular networks are upgrading to digital. During the transitional period, many phones are *dual mode*, that is, they can use both digital or analog networks.

Digital technologies

The second generation of wireless technology is digital. Digital systems convert voice signals to binary format for transmission. These systems also enable wireless Internet access. There are two kinds of digital bearer networks: *circuit-switched* and *packet-switched*.

A circuit-switched network dedicates a separate channel (circuit) for the duration of the communication. Most traditional public telephone networks and analog cellular networks are circuit-switched: A dedicated connection remains open for the entire phone call. For this reason, it is an inefficient technique.

In contrast, a packet-switched network divides digital data into a number of packets (known as datagrams) and sends them out over various network routes to their location. In addition to the actual message data, each packet also contains its destination address, and the order in which it is to be reassembled. Packet switching,

unlike circuit switching, does not require a dedicated line, so it uses bandwidth more efficiently. The Internet's TCP/IP is a packet-switched protocol.

TDMA

Time Division Multiple Access (TDMA) is a circuit-switched digital technology that allows multiple users to share the same voice channel by splitting each channel into multiple time slots, transmitting each conversation alternately in each slot, and then reconstructing them upon reception.

CDMA

Code Division Multiple Access (CDMA) is a form of *spread spectrum* digital technology, developed primarily by Qualcomm Corporation. Spread spectrum means that a signal is distributed among a range of frequencies, instead of being transmitted on a single frequency over an interval of time. A CDMA phone uses a pseudo-random code to spread the original signal over a range of frequencies simultaneously. The base station (cellular tower) uses the inverse code to reconstruct the original signal. This scheme provides a number of advantages over TDMA. CDMA is now the leading digital wireless bearer technology, and is the basis for third-generation wireless technologies.

CDMA can carry between two and three times as many calls as TDMA in the same bandwidth. CDMA has greater range, requires less power, and is more secure than other bearer network technologies.

GSM

The Global System for Mobile Communications (GSM) is a digital cellular radio network operating in over 200 countries worldwide. It has become standard in most of Europe, and is growing in the Americas and Asia.

GSM includes an intelligent network that supports Subscriber Identity Module (SIM) cards in phone handsets. The SIM card is actually an advanced "smart card" with an embedded microchip that identifies the handset on the network, enabling it to work in many different countries and networks. Because of the almost universal adoption of GSM standards in Europe, a GSM cellphone will work almost anywhere in Europe, as well as in parts of Asia.

The original European version of GSM is a TDMA technology. The version of GSM used in the U.S., however, uses a different frequency and the two versions are therefore incompatible. GSM has a maximum data transfer rate of 9.6 Kbps. In 1998, the European Telecom Standards Institute endorsed CDMA for third-generation GSM. Additionally, GSM providers are planning upgrades such as GPRS and High Speed Circuit Switched Data (HSCSD).

PCS

In the 1990s, the U.S. Federal Communications Commission (FCC) authorized a new frequency band for wireless communication: the 1.9 GHz band, higher than other U.S. cellular systems. The Personal Communication Services (PCS) is a digital wireless network that uses this frequency band. In North America, PCS is primarily CDMA-based. PCS spectrum is now deregulated and also includes networks based on GSM and TDMA.

Wireless Application Protocol

The Wireless Application Protocol (WAP) enables wireless devices to access the Internet. WAP is an open standard developed initially by OpenWave Systems, (formerly Phone.com, and before that, Unwired Planet), for the WAP Forum, an industry consortium including Motorola, Nokia, and Ericsson, and over 200 other companies. WAP works across a broad range of devices, and many manufacturers and software developers are making WAP-compliant products. Although it is a relatively new standard, WAP has received widespread attention. It is the leading cellular phone Internet standard in Europe and is gaining popularity in the U.S.

WAP defines the Wireless Markup Language (WML), which evolved from Handheld Device Markup Language (HDML) language, developed by Phone.com. Currently, the vast majority of existing Internet-capable cellphones support HDML. However, many WAP gateways will automatically convert WML content to HDML when necessary.

You can connect to the Web using a WAP-enabled cellphone or other WAP device. All WAP devices contain a microbrowser that can display WML. Another key part of the equation is a *WAP gateway*, a server that transforms WAP requests to HTTP requests; then transforms the HTTP response back to a WAP response.

As illustrated in Figure 2.6, the process is:

1. Using your WAP-enabled cellphone or other device, you enter the URL of a website or click a link from a preloaded page.
2. The microbrowser sends the request over the wireless network to a cellular base station, and then on to the WAP gateway.
3. The WAP gateway retrieves the requested document from the Internet.
4. The WAP gateway compiles the WML document to bytecode format, and sends it back through the cellular network to your cellphone. Some WAP gateways can also convert HDML (or HTML) to WML “on the fly.”
5. The microbrowser displays the WML page on your cellphone’s display.

Figure 2.6 How WAP works

The current version of the WAP standard (WAP 1.2) includes the Wireless Transport Layer Security (WTLS), which specifies encryption for data sent from a cellphone to the network operator. Once the data reaches the WAP gateway, it is decrypted from WTLS and re-encrypted to SSL (or to whatever Internet security protocol is being used). For a brief time, depending on network latency and speed, the user's data is unencrypted. Although this vulnerability occurs on the server, it is still a concern.

As illustrated in Figure 2.7, WAP specifies a set of protocols that are analogous to Web standards. At the top level is WML, which corresponds to HTML for displaying content to the end user, along with WMLScript, a scripting language analogous to JavaScript. These two presentation technologies are part of the Wireless Application Environment (WAE).

Below the WAE, the wireless session protocol (WSP) and the wireless transaction protocol (WTP) correspond to the Hypertext Transfer Protocol (HTTP). WAP security is handled by the Wireless Transport Layer Security (WTLS), which corresponds to Secure Sockets Layer (SSL), and other Internet security mechanisms. Finally, at the lowest level, corresponding to UDP and TCP/IP on the Web, are the Wireless Datagram Protocol (WDP) and bearer protocols such as CDMA.

For more information on WAP, see http://developer.phone.com and http://www.wapforum.com.

Figure 2.7 WAP compared to Web protocols

Controversy over WAP

Initially the subject of extensive excitement and media attention, WAP has subsequently been criticized by some as overly complex and insufficient for the rapidly evolving wireless market. Because of its slow performance, rudimentary graphics, and architectural complexity, some have labeled it a "transitional technology" that will be supplanted when 3G networks come on line.

Additionally, an ongoing patent lawsuit between Geoworks, Inc., and Phone.com over WAP software has cast some doubt on adoption of the technology. However, this situation is likely to be temporary.

Industry analysts believe three key issues must be resolved for WAP to succeed:

- **It needs a high-speed backbone:** Sophisticated wireless Web applications will require greater bandwidth than is available with the current architecture.
- **The protocol needs improved security:** The WAP 1.2 specification resolves some security problems, but this area continues to be a concern.
- **WAP devices must be standardized:** All WAP microbrowsers do not implement the complete WAP standard, making it difficult to design applications that work reliably across many devices.

The proposed WAP 2.0 standard, which the WAP Forum is expected to approve by mid-2001, will incorporate standard technologies such as Extensible Hypertext Markup Language (XHTML), Transmission Control Protocol (TCP), as well as dynamic proxy navigation (DPN) which provides end-to-end encryption. This new version of WAP may help to address many concerns with the current WAP standard.

i-mode

Japan's NTT DoCoMo launched the i-mode wireless service in early 1999. It is currently the most popular Internet phone technology in Japan, with more than 15 million subscribers as of late 2000. i-mode originally was deployed only in Japan and Hong Kong, but DoCoMo is expected to bring the service to the U.S. and Europe.

i-mode uses both Hypertext Transfer Protocol (HTTP)—the Web standard—and compact HTML (cHTML), a subset of HTML for small information appliances. cHTML omits some of the resource-intensive features of HTML, to reduce download time to mobile devices. It uses HTTPS for secure transmission.

Figure 2.8 An i-mode phone

i-mode uses a PDC-P (Personal Digital Cellular-Packet) method of data transmission over the existing PDC network used for ordinary voice traffic. PDC-P is a Japanese packet-switched network that allows users to stay online continuously, but not be charged for the time they are online. Rather, users pay only for the amount of data that they retrieve. This is in contrast to a circuit-switched network in which the communication path is dedicated to a single connection and users pay for connection time.

The overall architecture of an i-mode network is similar to WAP: When an i-mode device makes a request, the i-mode gateway translates this to the server and back from the server to the wireless device. i-mode uses PDC as its transmission protocol, which enables several subscribers to use the same line at once. i-mode's transmission speed is 9.6 Kbps, and NTT DoCoMo is aggressively moving to a 3G network.

i-mode vs. WAP

As the world moves to interoperable wireless systems, WAP and i-mode may continue to co-exist as competing standards, but it is more likely that they will evolve into a single standard. It appears that both are moving toward XHTML for presentation; for more information, see "XHTML" on page 27.

Table 2.2 summarizes the major differences between i-mode and WAP.

Table 2.2 i-mode vs. WAP

	WAP	i-mode
Markup language	WML	cHTML
Access	Dial-up	Always on
Billing	Based on connection time	Based on amount of data transmitted
Security	WTLS	SSL
Standards compliance	Proprietary	Open Internet standard

Location-based services

Wireless *location services* are emerging technologies that provide Automatic Location Identification (ALI) to cellphones and other wireless devices. In its essential form, a location is specified by latitude and longitude, although other conventions such as zip code and area code are sometimes used.

Location Based Services (LBS) are end-user applications and content based on location services. In addition to obvious applications for emergencies, one can imagine applications to:

- Find the nearest restaurants, gas stations, or other retail sites, based on your current location (for example, “there is a Starbucks 3 blocks north of you”).
- Provide ongoing directions to a preset destination as you walk or drive (for example, “turn left at the next intersection”).
- Remind you to get something you need when you pass near a retail outlet (for example, “don’t forget to pick up milk at the grocery store, which is now two blocks away”).

Cellular carriers are implementing various ways of pinpointing a person's location, using techniques such as Global Positioning System (GPS) receivers built into handsets and sophisticated triangulation from cellular base stations. A major impetus for these services are new regulations from the U.S. Federal Communications Commission (FCC), which require all carriers to have 911 location-tracking systems in operation by October, 2001. This FCC requirement is known as *enhanced 911* (E911), and will require wireless carriers to provide emergency dispatchers with a caller’s location.

E911 requires carriers to deliver latitude and longitude location information, known as ALI, to emergency dispatchers. The FCC has a schedule that mandates phase-in of handset support for ALI. By the end of 2002, all new digital handsets activated are to be ALI-capable. Proposals have been made for European networks to provide similar services.

Motorola, Nokia, and Ericsson formed the Location Inter-operability Forum (LIF) to develop and promote LBS technology through global standards bodies and specification organizations. This nascent consortium may help to unify global LBS standards. For more information on the LIF, see http://www.locationforum.org.

Specialized location service vendors include

- SignalSoft, http://www.signalsoftcorp.com
- Xypoint, http://www.xypoint.com
- Kivera, http://www.kivera.com
- Gravitate, http://web4.xor.com/grvt8

SMS

The Short Message Service (SMS) provides mobile devices the ability to send and receive brief text messages. SMS is part of the GSM standard, and is also available on personal communications service (PCS) networks based on GSM, as well as on some CDMA and TDMA networks. While SMS is not widely available in the U.S., Europeans send over ten billion SMS messages a month, according to the GSM association.

The length of SMS messages depends on the bearer network:

- GSM limits messages to 160 characters
- CDMA limits messages to 256 characters
- iDEN limits messages to 140 characters

Every GSM network has a message center, which manages SMS traffic. As illustrated in Figure 2.9, when someone sends an SMS message, it first goes to the SMS center (SMSC). The message center adds the date, time, and phone number of the sender to the message and then sends it to addressee. If the recipient's telephone is deactivated (or out of range), the message is stored and sent when the recipient next connects to the network.

Figure 2.9 Sending an SMS message over the Internet

An SMS-capable phone can send or receive a message at any time, even during a voice or data call. SMS guarantees delivery of the message by the network, using its

"store and forward" system. Thus, it is not an "instant messaging" system, but more akin to email.

SMS has two basic services:

- Mobile-terminated short message (MT–SM), the ability to receive SMS messages
- Mobile-originated short message (MO–SM), the ability to send SMS messages

Nearly all GSM mobile telephones support MT-SM. At the beginning of 1999, approximately 75 percent of worldwide GSM mobile phones supported MO-SM.

Third-generation (3G) technologies

The latest buzzword in the wireless industry is *third-generation* (3G) systems. 3G is an umbrella term for packet-switched wireless broadband technologies that are designed to carry data and multimedia as well as voice.

The telecommunications industry, consortia, and standards bodies are pursuing a veritable alphabet soup of 3G technologies. The International Telecommunication Union (ITU) is the primary international standards body that controls the 3G standard, known as International Mobile Telecommunication 2000 (IMT-2000) or Universal Mobile Telecommunications System (UMTS).

Figure 2.10 Vision of a future 3G phone (courtesy Nokia Corp.)

The General Packet Radio Service (GPRS) is a data service that allows information to be sent and received across a GSM telephone network. GPRS can theoretically achieve data rates of up to 171 Kbps. This is about three times as fast as fixed telecommunications networks and ten times as fast as current circuit-switched data services on GSM networks.

In Japan, NTT DoCoMo is slated to roll out its 3G service in May, 2001. In Europe, 3G is expected to become available by 2002, followed soon thereafter by the U.S. Regardless of which technologies are adopted, the upshot is that wireless bandwidth will increase substantially, enabling greater use of graphics and other multimedia content.

European providers are phasing in UMTS through a variety of intermediate technologies, including:

- HSCSD (High Speed Circuit Switched Data)
- GPRS (General Packet Radio Service)
- EDGE (Enhanced Data rates for GSM Evolution)

Client presentation technologies

Unlike the conventional Worldwide Web, which has HTML as a universal standard, the wireless Web has a number of competing markup languages. The primary markup languages are:

- HTML
- WML
- cHTML
- XHTML
- VoiceXML (Voice Extensible Markup Language)

In addition, on some platforms you can create a custom client application to access wireless content. The emerging standard for such clients is J2ME; for information, see "J2ME and the MIDP" on page 28.

Note This book does not discuss HDML, because it is considered a legacy technology, and most WAP gateways will automatically convert WML content to HDML as necessary.

For a more detailed description of these languages, see Chapter 5, "Survey of wireless presentation languages."

HTML

Web browsers running on subnotebook PCs can display standard HTML. In addition, some microbrowsers support HTML, and many PDAs include support for HTML (for example Palm's Web-clipping applications). Finally, many wireless applications may include personalization, customization, or administration functions that the user performs from a PC Web browser. For this reason, wireless developers should not ignore standard HTML.

While appropriate for high-end wireless devices running standard Web browsers, complex HTML is generally not suitable for display on other devices such as Web phones or handhelds. HTML has superior graphics and layout capability, but is simply not tailored for the small display size and bandwidth of such wireless devices.

The Worldwide Web Consortium (W3C) recommends moving HTML development to XHTML, since all future HTML standards will be based on XHTML.

WML

The Wireless Markup Language (WML) is a stripped-down markup language designed for devices that have small displays and low communications bandwidth.

WML is part of the WAP standard, and is the presentation language for WAP devices.

WAP also defines a scripting language called WMLScript, which is similar to JavaScript. Not all microbrowsers support WMLScript, however. And support for WML is inconsistent across different phones. The website AnywhereYouGo.com has a list of devices that support different WAP standards, markup languages, security levels, and screen sizes. For more information, see http://www.anywhereyougo.com/ayg/ayg/wap/devices/Index.po.

cHTML

Compact HTML is a subset of HTML for wireless devices, promulgated by NTT DoCoMo of Japan. It omits many complex HTML layout tags, while retaining the key formatting tags.

The version of cHTML described on NTT DoCoMo's website includes some additional features not in HTML; however, the version of cHTML submitted to the W3C is a proper subset of HTML.

XHTML

XHTML is a developing standard from the W3C, and is a reformulation of HTML as an XML application. It is "backwards compatible" with HTML; that is, it can be displayed by devices that display HTML, such as current Web browsers. However, it has the advantage of being XML, so that XHTML content is viewable in XML-enabled devices.

There are several versions of XHTML; the version suitable for wireless presentation is *XHTML Basic*, which contains a subset of tags in the complete language. While there are currently no wireless devices that support XHTML, both the WAP forum and NTT DoCoMo have stated that XHTML Basic will become the markup language for WAP and i-mode in the future.

VoiceXML

Voice eXtensible Markup Language (VoiceXML) is an emerging XML-based standard that intends to bring the advantages of Web content and development to interactive voice-response applications. VoiceXML enables you to create audio dialogs using synthesized speech and voice recognition. It is being developed and promoted by the VoiceXML Forum, an industry organization founded by AT&T, IBM, Lucent, and Motorola. In May, 2000, W3C approved the VoiceXML 1.0 standard.

As illustrated in Figure 2.11, the VoiceXML architecture is similar to WAP: A Voice server acts as an intermediary between the user on the telephone and the content on the Internet. The Voice server includes a VoiceXML interpreter with a voice recognition and synthesis engine used to automate a conversation between a machine and a human being on the telephone.

Figure 2.11 VoiceXML access to Web content

VoiceXML applications provide a formal "question and answer" user experience. Although this may be appropriate for certain applications such as customer-support call centers, it may not be appealing for all applications.

For more information on the VoiceXML Forum, see http://www.voicexml.org. Several voice platform vendors also provide good information:

- Tellme Networks developer site, http://studio.tellme.com
- VoiceGenie developer site, http://developer.voicegenie.com/
- BeVocal developer site: http://cafe.bevocal.com/

J2ME and the MIDP

Java™ 2 Micro Edition (J2ME) from Sun Microsystems targets a range of devices from pagers and cellphones to television set-top boxes. While this technology can be viewed in some ways as competing with WAP and other wireless-specific standards, according to Sun's website on J2ME:

> ...a browser interface is convenient to access static information (news, sports scores) or display a list of applications and services, but the use of Java technology is more compelling for interactive or transaction-oriented applications and services (m-commerce, games, etc.) that the user can continue to interact with even while disconnected from the network.

The J2ME includes:

- Scaled-down Java virtual machines for a variety of devices; the most well-known is the Kilobyte Virtual Machine (KVM)
- Specialized APIs for each type of device, along with a "profile" that specifies the minimum set of useful APIs and the Java virtual machine functions required to support those APIs
- Tools for deployment and device configuration

Figure 2.12 J2ME Mobile Information Device Profile (MIDP)

In the case of handheld wireless devices, the J2ME *Connected Limited Device Configuration* (CLDC) addresses characteristics such as battery operation, constrained memory, limited processing power, and low bandwidth, high-latency network connections.

The heart of the CLDC is the *Kilobyte Virtual Machine* (KVM). The KVM is designed specifically for small mobile devices with 16/32-bit RISC/CISC microprocessors, and with as little as 160 Kbytes of total memory available, with 128K for the virtual machine and libraries themselves.

The *Mobile Information Device Profile* (MIDP) is a set of Java APIs which, together with the CLDC, provides a complete J2ME application runtime environment for mobile devices. The MIDP specification addresses issues such as user interface, persistence storage, networking, and application model.

For more information on J2ME, see the following websites:

- Sun's J2ME site, http://java.sun.com/j2me/
- Bill Day's J2ME archive, http://www.billday.com/j2me/index.html

Where to find more information

Wireless application development is a new topic, but there is more information every day. This section provides a few references as a starting point for further research and study.

Books

Until recently there were only a handful of books on wireless application development. The latter part of 2000 saw an increasing avalanche of books becoming available, as publishers rush to catch up to the emerging trend. This section provides an incomplete list of books available at the end of 2000.

Professional WAP, by Charles Arehart, et. al.
Wrox Press; July 2000

A lengthy (over 800 pages) introduction and survey of WAP from over a dozen contributors. Includes sections on using Java, XML, XSLT, ASP, and security, plus reference material on WML and WMLScript.

WAP Servlets, by John Cook
John Wiley and Sons; November 2000

Provides an introduction to WAP and WML, with focus on servlet-powered applications.

WAP Development with WML and WMLScript, by Ben Forta (et. al.)
Sams, September 2000

A step-by-step guide on how to design wireless applications, focusing on redesigning existing websites. Contains instructions for generating WAP content in all major languages, including Cold Fusion, ASP, Perl, Java, and JSP.

Telecosm, George Gilder
Free Press; September 2000

An excellent history and overview of the telecommunications industry (including wireless) from the acclaimed author of *Microcosm*. Good background reading.

HTML & XHTML: The Definitive Guide, by Chuck Musciano and Bill Kennedy
O'Reilly and Associates; August 2000

Comprehensive reference from a reliable publisher. Oriented more toward web development, but useful nevertheless.

Wireless Web Development, by Ray Rischpater
Apress; May 2000

A good introduction to wireless applications, HDML, WML, and WMLScript.

WAP-The Wireless Application Protocol: Writing Applications for the Mobile Internet by Sandeep Singhal, et. al.
Addison Wesley Professional; October 2000

A clearly-written guide and reference for WAP. Provides a logical engineering view of WAP elements. Moves cohesively from introductory material into advanced topics.

Websites

Since wireless technology and markets are evolving so rapidly, the best place to find information is often the World Wide Web. In addition to vendors websites that focus on specific products or technologies, there are an increasing number of wireless developer sites. Some of the best include:

- http://www.AnywhereYouGo.com
- http://www.wirelessdevnet.com
- http://www.allnetdevices.com

- http://www.gelon.net
- http://www.thefeature.com

Periodicals

RCR Wireless News is one of the most popular periodicals covering the wireless industry. It offers a mix of breaking news and in-depth analysis into the issues that mold today's wireless telecommunications environment. It sister publication, *Global Wireless,* covers the wireless industry and technology from a worldwide perspective.
Website: http://www.rcrnews.com/

Wireless Week is a weekly newspaper covering all the business, technology, and regulatory news in the areas of cellular, personal communications services, paging, wireless Internet, wireless data, satellite, wireless local loop, and microwave technologies.
Website: http://www.wirelessweek.com/

Mobile Computing & Communications provides critical information that enables buyers of mobile technology to make sound purchasing decisions. It focuses on all aspects of mobile technology. Each issue brings its readers trends and developments, mobile outlooks, application solutions, and much more.
Website: http://www.mobilecomputing.com/

Chapter 3

Getting started with wireless application development

This chapter introduces wireless application development with Lutris Enhydra. It does so by introducing and describing how to use the AirSent wireless sample Enhydra application. It describes how to build, run, use, and modify AirSent using the application development tools provided with Lutris Enhydra. It concludes by showing how to use the Enhydra Application Wizard to quickly create a new wireless application.

Note This chapter assumes you have a basic understanding of Enhydra. For a general introduction to Enhydra, see *Getting Started with Lutris Enhydra*.

Enhydra for wireless applications

One of the keys to success with the wireless Web is having a compelling application. With Enhydra, you can quickly create wireless applications using Internet standards such as Java servlets, and markup languages such as WML, cHTML, and XHTML.

Enhydra advantages

Using Enhydra allows you to develop an application with multiple *personas* (also called *skins)*, or presentations for different devices and user profiles. As illustrated in Figure 3.1, "Enhydra in the wireless world," a single Enhydra application can serve WML to WAP clients, cHTML to i-mode clients, and HTML to PC Web browsers. It is able to do this through its flexible and dynamic presentation logic, which is separate from the application business and data logic.

Using Enhydra for your wireless applications also provides you with all the advantages of the Enhydra application server platform:

- Scalability
- Servlet support
- JDBC database access
- Session management
- Load balancing through Enhydra™ Director

Figure 3.1 Enhydra in the wireless world

Application development tools

Lutris Enhydra provides a suite of development tools for wireless application development.

Core development tools

You can use the core tools for creating Web or wireless applications:

- JBuilder™ Foundation and Forte™ for Java™
- Kelp tools for JBuilder and JDeveloper™
- Enhydra XMLC,™ the extensible markup language compiler
- DODS, the Data Object Design Studio

For more detailed information on Kelp, XMLC, and DODS, refer to the *Developer's Guide*.

Wireless development tools

These additional tools are specifically for wireless application development:

- Motorola J2ME™ SDK Components Developer Edition (Windows only)
- Nokia® WAP Toolkit

- Pixo™ Microbrowser
- WAPtor™ WML editor
- YoSpace™ SmartPhone Emulator (Fully functional evaluation version).

For overview information on using these tools for wireless application development, see Chapter 6, "Using wireless development tools."

First steps: Installing Lutris Enhydra and tools

Before beginning the hands-on guided tour of the AirSent wireless sample application, you will need to install the following software (listed in recommended order) included with Lutris Enhydra:

Note To install each of these products, follow the step-by-step installation instructions available from the Welcome page (`\\index.html`) on the Lutris Enhydra CD.

1. InstantDB™
2. Lutris Enhydra™
3. Nokia WAP Toolkit or the YoSpace SmartPhone Emulator
4. Motorola J2ME SDK (only for Windows systems)

If you also want to use the JBuilder IDE, install:

1. JBuilder Foundation 4.0
2. Enhydra Kelp Tools for JBuilder

Note You can use an IDE other than JBuilder if you prefer (you are not required to use an IDE at all), but this book uses JBuilder for illustration purposes.

Be sure to configure Enhydra as described in the CD instructions before you do anything else.

Using the AirSent sample application

The AirSent application is a sample wireless Enhydra application that creates output in HTML, WML, cHTML, XHTML, and XML. It is installed to `<enhydra_root>/examples/AirSent`.

Note For a detailed explanation of how AirSent works, see Chapter 8, "Understanding the AirSent sample application."

There are two parts to AirSent:

- An Enhydra server application, which runs on a PC or UNIX system on which you have installed the Enhydra application server. This is implemented as a super-servlet application using Enhydra presentation objects (POs).
- A J2ME client application, which runs on the Motorola iDEN phone emulator (in deployment, it would run on a Java phone).

Although designed to work closely together, these are two distinct applications, and you must build them separately.

Step 1: Building AirSent

You can build the AirSent applications from the command line as described in "Option 1: Building from the command line" or, if you are using JBuilder, you can follow the instructions outlined in "Option 2: Building using JBuilder and Kelp" on page 37.

Option 1: Building from the command line

Building AirSent involves building the server application and then the client application.

Building the server application

To build the AirSent application from the command line, follow these steps:

1. Open a shell window (a Cygwin shell on Windows), and change the working directory to the AirSent server directory:

   ```
   cd <enhydra_root>/examples/AirSent/po
   ```

2. Edit `config.mk`, and make the following changes:

   1. Set the ENHYDRA_DIR variable to the location of your Enhydra installation, for example:

      ```
      ENHYDRA_DIR = /usr/local/lutris-enhydra3.5
      ```

   2. Set the IDB_HOME variable to the `Classes` subdirectory of your InstantDB installation, for example:

      ```
      IDB_HOME = /usr/local/instantdb/Classes
      ```

      Make sure you have installed InstantDB to the location you specify.

3. In the shell window, enter the command

   ```
   make
   ```

   This builds the application. You will see a series of messages displayed in the shell window.

For instructions on how to run the application, see "Step 2: Running AirSent" on page 39.

Tip If the build fails with a "no permission" error, make sure you don't have a trailing space in your ENHYDRA_DIR setting.

Building the client application

The AirSent J2ME client application runs on a Java-enabled cellphone or on an emulator, such as the one in the Motorola iDEN™ SDK.

Note The Motorola SDK is for Windows systems only.

To build the AirSent J2ME client application from the command line, follow these steps:

1. Open a Cygwin shell window, and make the AirSent client directory the active directory:

   ```
   cd <enhydra_root>/examples/AirSent/client/kvm
   ```

2. Edit `config.mk`, and make the following changes:

   1. Set the ENHYDRA_DIR variable to the location of your Enhydra installation, for example:

      ```
      ENHYDRA_DIR = /usr/local/lutris-enhydra3.5
      ```

   2. Set the MOTO_CLASSPATH variable to the location of your Motorola J2ME SDK installation, for example:

      ```
      MOTO_CLASSPATH = /MotoSDK
      ```

      Make sure you have installed the Motorola SDK to the location you specified.

3. In the shell window, enter the command

   ```
   make
   ```

   This builds the client J2ME application, and copies files to the Motorola SDK directory. You will see a series of messages displayed in the shell window.

For instructions on how to run the application, see "Running and using the AirSent J2ME client application" on page 44.

Option 2: Building using JBuilder and Kelp

Before beginning, make sure you have correctly installed both JBuilder and Kelp. For more information, see "First steps: Installing Lutris Enhydra and tools" on page 35.

Building the server application

To build AirSent using JBuilder and Kelp, follow these steps:

1. Start JBuilder.
2. Open the AirSent server project file: `<enhydra_root>/examples/AirSent/po/airsent.jpx`.
3. Choose Wizards | XMLC Compiler to invoke the Enhydra XMLC Compiler wizard. Click the button labeled ">>" to select all the files for compiling, then click the Compile button. Kelp will call XMLC to compile all the HTML, WML, cHTML, and XHTML files in the project.
4. When you see the message "--- XMLC finished compiling ---" click Close to exit the wizard.
5. Make sure the project is using the right libraries: Choose Project | Project Properties, select the Paths tab, then click on "Required Libraries." There are three required libraries:
   - Lutris Enhydra, which should reference `<enhydra_root>/lib/enhydra.jar`
   - InstantDB, which should reference `<instantdb-root>/Classes/idb.jar`

- JTA (Java™ Transaction API), which should reference
  `<instantdb-root>/Classes/jta-spec1_0_1.jar`

Select each library, and then click the Edit button to confirm that the library references the correct JAR file in the Class tab in the Library Settings section of the Configure Libraries dialog box.

6. To build the application, choose Project | Make Project "AirSent.jpx" or click the Make Project button in the toolbar. When JBuilder is finished compiling, the message "Build succeeded" is displayed at the bottom of the JBuilder window.
7. Choose Wizards | Enhydra Deployment to invoke the Enhydra Deployment wizard.
8. When the message "---Enhydra Deployment Complete---" appears, click Close to exit the wizard.

For information on running AirSent in JBuilder, see "Option 2: Running AirSent from JBuilder" on page 39.

Building the client application

Although you can use JBuilder to browse, edit, and build the Java code for the J2ME client application, you must put the application code through a pre-verifier before you can run it with the Motorola iDEN phone emulator. The pre-verifier program is a Windows application that ensures the code does not make any calls outside the J2ME API. Additionally, the emulator requires class files to be copied into certain directories. For these reasons, you must build the client application from the command line before you can run it in the emulator.

To build the client application:

1. Start JBuilder.
2. Open the AirSent Java client project file:
   `<enhydra_root>/examples/AirSent/client/kvm/airsent-jclient.jpx`
3. Make sure the project is using the right library: Choose Project | Project Properties, and then in the Paths tab, click "Required Libraries." There is one required library: MotoSDK, which should reference the `/lib` sub-directory of the Motorola SDK directory.

   Select the library, and then click the Edit button to confirm that the library references the correct directory in the Class tab in the Library Settings section of the Configure Libraries dialog box.
4. To build the application, choose Project | Make Project "airsent-jclient.jpx" or click the Make Project button in the toolbar. When JBuilder is finished compiling, the message "Build succeeded" is displayed at the bottom of the JBuilder window.

To run the pre-verifier and copy the class files to their runtime location, follow all the steps outlined in "Building the client application" on page 36.

Step 2: Running AirSent

You can run the AirSent server application either from the command line ("Option 1: Running AirSent from the command line") or from JBuilder ("Option 2: Running AirSent from JBuilder").

Note "Step 3: Using AirSent" on page 39 provides instructions on running and using the AirSent J2ME client application (see "Running and using the AirSent J2ME client application" on page 44).

Option 1: Running AirSent from the command line

To run the AirSent application from the command line:

1. In a shell window (a Cygwin shell on Windows), change the working directory to the AirSent server output directory:

   ```
   cd <enhydra_root>/examples/AirSent/po/output
   ```

2. Enter this command:

   ```
   ./start
   ```

   You will see a series of messages starting with

   ```
   ***
   ***  Connect to PO based AirSent at http://myServer:9000/
   ***
   ```

   Then load the URL http://localhost:9000 in your Web browser to access the application. Where 9000 represents the port number (9000 by default).

Option 2: Running AirSent from JBuilder

You can also run the AirSent application from JBuilder:

1. Start JBuilder.
2. Choose Run | Run Project, or click the Run button in the toolbar.

   Then load the URL http://localhost:9000 in your Web browser to access the application. Where 9000 represents the port number (9000 by default).

Step 3: Using AirSent

This section guides you through using the AirSent Web pages (see "Using AirSent Web pages") and the AirSent WAP content that is displayed on either the YoSpace SmartPhone Emulator or the Nokia Toolkit (see "Viewing AirSent wireless content" on page 41). It also provides instructions for running the AirSent J2ME client application using the Motorola iDEN emulator (see "Running and using the AirSent J2ME client application" on page 44).

Using AirSent Web pages

Enter the URL http://localhost:9000 in your Web browser. You will see the AirSent Welcome page as shown in Figure 3.2, "AirSent Welcome page."

Figure 3.2 AirSent Welcome page

Click Admin App to log in as an administrator. Enter the user name `admin` and password `admin`. Browse around and perform various functions. Click Logout to log out as an administrator.

Click Customer App to log in as a customer. You can use the preconfigured account with the username `enhydra` and password `wireless`, or you can create your own customer account by clicking Register to register as a new customer.

After logging in, try entering an order.

Viewing AirSent wireless content

You can use either YoSpace SmartPhone Emulator or the Nokia Toolkit to view AirSent WAP content. The Nokia Toolkit runs on Windows only. This example uses YoSpace SmartPhone Emulator, which runs on Linux or Windows.

Note If you want to view AirSent's cHTML content, use the Pixo microbrowser emulator.

To login and begin using AirSent WAP content:

1. Start the YoSpace SmartPhone Emulator, check the Evaluation Use Only box, then click OK.

Note The fully functional evaluation version of YoSpace only runs for ten minutes at a time. After ten minutes, YoSpace will exit, and you have to restart it.

2. In the YoSpace window, choose Options | Open Location, then enter the URL for AirSent: http://localhost:9000. Click OK and you will see the AirSent Home Page for couriers:

3 Click and drag down on the Navi Roller to scroll down. You will see the Courier Login screen:

4 When the Badge field is highlighted, click the Navi Roller to display the Alphanumeric Entry screen:

5 Using the Navi Roller, scroll down to the letter “b,” then use the Select softkey to choose OK. Enter “b” for both the badge field and the password field. Then click the Select softkey to choose Options, scroll down to Login, and use the Select softkey again to choose Login. You will then see this screen:

6 Click the Navi Roller to choose Pick-ups. You will then see this list of deliveries:

7 Click the Navi Roller when a pickup location (for example, “Lutris”) is highlighted to display the details for the selected deliveryThe emulator displays the address for the delivery; the example below shows the address for Lutris:

You can also try entering a new order from the customer screen, using the customer login `enhydra` and password `wireless`. Then, you can follow the procedure outlined above to verify picking up and dropping off the item.

Running and using the AirSent J2ME client application

This section describes how to use the Motorola iDEN emulator to run the AirSent J2ME client application and take a tour of the AirSent sample application functionality (similar to the tour shown in “Viewing AirSent wireless content” on page 41).

Note The Motorola iDEN emulator only runs on Windows systems.

To run and use the AirSent J2ME client application:

1 First, make sure the AirSent application is running (see “Step 2: Running AirSent” on page 39). Then, in a shell window (a Cygwin shell on Windows), enter this command:

```
cd <moto-SDK>/scripts
```

where moto-SDK is the directory where you installed the Motorola SDK. Then enter the command:

```
./AirSent.bat
```

This executes a batch file that runs the Motorola iDEN phone emulator, and loads the AirSent URL. You will see the emulator, with the AirSent messenger login page:

Figure 3.3 Motorola iDEN phone emulator with AirSent login page

2 Login as a courier with user name “b” and password “b.” To enter the letter “b,” click the 2 key three times, then scroll down with the Rocker button to the Password field. Enter “b” for the password, then click the right softkey to login.

You will see the AirSent messenger home page that provides choices for pickups and dropoffs:

3 To confirm that you (as a courier) have dropped off a package, scroll down to Drop Offs, and click the right softkey (labeled SELECT). In the screen that appears, scroll down to DONE, and click the right softkey.

4 Click the right softkey (SELECT) to choose Pick Ups. The phone will display the list of pickups:

5 By default, there is only one pickup for this messenger, “lutris.” Click the left softkey (labeled MENU) to display the menu for that pickup:

6 The menu lists two choices: DETAILS and DONE. The first item, DETAILS, is initially highlighted. Click the right softkey to select it. The phone will display the details for that pickup:

7 Scroll down using the rocker key to display the address for that pickup.

Step 4: Modifying AirSent

Now that you have seen AirSent in action, it’s time for you to get your feet wet by making a simple change to it. This change won’t be very meaningful, but it will familiarize you with the process of editing, rebuilding, and running the application.

1 Stop the application by entering *Ctrl-C* in the shell window in which you started AirSent. If you ran the application from JBuilder, stop it by clicking the red Stop button in the Multiserver tab in the lower left corner of the window.

2 Now, edit the file `<airsent-root>/presentation/admin/AdminLogin.java`. Find the `handleLogin()` method. The following statement in this method checks the user name and password entered and returns an error message if either is invalid:

```
if (messenger =
   getApplication().getHomeFactory().getMessengerFactory().validatePassword(login,
         password)
   == null) {
return showPage("Invalid username or password.");
```

3 Change the error message (the string argument to the `showPage()` method) to something more emphatic, for example, "Hey, dummy, you entered the wrong username or password!"

Rebuilding from the command line

After you have made the change:

1 In a shell window (a Cygwin shell on Windows), change the working directory to the AirSent server directory:

```
cd <enhydra_root>/examples/AirSent/po
```

2 Enter this command:

```
make clean
```

This removes all the existing classes, so when you rebuild the application, it will include your change.

3 Enter this command to rebuild the application:

```
make
```

Rebuilding using JBuilder

If you are using JBuilder, choose Project | Rebuild Project “airsent.jpx,” or click the Rebuild button on the toolbar.

Running the application

Run the application as before, from the command line or from JBuilder. Then:

1 Click the “Admin App” link as shown in Figure 3.2, “AirSent Welcome page.”

Figure 3.4 AirSent Welcome page

2 Enter an invalid username or password.

Confirm that the application displays the new error message instead of the old one. It should appear as shown in Figure 3.5, “AirSent Error Message page.”

Figure 3.5 AirSent Error Message page

Creating a new wireless application

The quickest way to create a new wireless application is to use the Enhydra Application Wizard. If you have installed Kelp, you can use the Application Wizard from JBuilder (see "Using the Application Wizard with JBuilder and Kelp") or you can use it independently, without Kelp (see "Using the Application Wizard independently" on page 53). This section explains both ways to create a wireless application using the Enhydra Application Wizard.

Note For complete information on the Enhydra Application Wizard, see the *Developer's Guide*.

Using the Application Wizard with JBuilder and Kelp

If you have not already installed JBuilder and Kelp, follow the instructions provided in "First steps: Installing Lutris Enhydra and tools" on page 35.

To create a new application using JBuilder and Kelp,

1 Start JBuilder, close any open projects, and then choose File | New Project. You will see the first dialog box of the JBuilder Project wizard as shown in Figure 3.6, "JBuilder Project Wizard."

Figure 3.6 JBuilder Project Wizard

2 For the name of the project file, enter any name, for example "myWAPapp." Click Next to continue.

3 In the next panel of the Project Wizard, you can optionally enter the name of the application, your name, and your company; this information is for use in the project information HTML file.

4 Click Finish to create the new project.

5 Now, from the JBuilder menu, choose File | New. You will see the Object Gallery dialog box as shown in Figure 3.7, "Object Gallery dialog box."

Figure 3.7 Object Gallery dialog box

6 Select the Enhydra tab at the top of the dialog box. You will then have two choices: Web Application and Enhydra super-servlet. Select Enhydra super-servlet. You

will see the first dialog box in the Enhydra application wizard as shown in Figure 3.8, “Enhydra Application Wizard: Step 1.”

Figure 3.8 Enhydra Application Wizard: Step 1

7 In the Client Type pull-down box, choose WML. In the Root Path text field, enter the name of the directory in which you want to create the application, or choose Set to browse the directory structure. Then click Next to display the dialog box shown in Figure 3.9, “Enhydra Application Wizard: Step 2.”

Figure 3.9 Enhydra Application Wizard: Step 2

8 Select one of the radio buttons to specify copyright information for the application, then click Next to display the dialog box shown in Figure 3.10, “Enhydra Application Wizard: Step 3.”

Figure 3.10 Enhydra Application Wizard: Step 3

9 Check the appropriate check boxes to have Makefiles and Shell Scripts created. Click Finish to create the new application.

Building and running the application

The wizard creates a readme.html page containing instructions for building the application. Follow these steps to build and run it:

1 From the menu, select Wizards | XMLC Compiler. In the dialog box that appears, click Compile to compile the WML document (Welcome.wml) into a DOM class with XMLC. When you see the message “XMLC finished compiling,” click Close to close the dialog box.

2 Choose Project | Make to build the application. When the message “Build succeeded” appears at the bottom of the JBuilder window, the process is complete.

3 Deploy the project by selecting Wizards | Enhydra Deployment. In the dialog box that appears, click Deploy to process configuration files (called “templates” in this dialog box), deploy static content, and create a JAR file.

4 Choose Run | Run Project. This starts the Enhydra server and the application.

5 Start a wireless device emulator (YoSpace or the Nokia Toolkit) and load the URL:

```
http://localhost:9000/
```

For example, if you are using the Nokia Toolkit, you will see `mywapapp` running as shown in Figure 3.11, “mywapapp running in the Nokia Toolkit emulator.”

Figure 3.11 mywapapp running in the Nokia Toolkit emulator

Using the Application Wizard independently

You can run the Enhydra Application wizard without JBuilder.

1 In a shell window (a Cygwin window if you are using Windows), enter the following command:

```
appwizard
```

This launches the Enhydra Application wizard. You will see the following dialog box shown in Figure 3.12, “Enhydra Application Wizard Generator dialog box.”

Figure 3.12 Enhydra Application Wizard Generator dialog box

2 In the Generator pull-down box, choose Web Application or Enhydra SuperServlet, then click Next.

The dialog box shown in Figure 3.13, "Enhydra Application Wizard Client type and directory dialog box" will then appear.

Figure 3.13 Enhydra Application Wizard Client type and directory dialog box

3 In the Client Type pull-down box, choose WML to create a wireless application. In the Project Directory Name field, enter the name of the directory in which to create the application; in the Package field, enter the name of the application (if you

want, you can enter a full package name, for example, com.mycompany.foo.bar). Click Next to go to the next dialog box.

Two more dialog boxes complete the Application wizard:

4 In the first dialog box, you can choose to enter copyright information or specify a copyright file; make your choice and choose Next to continue.

5 In the final dialog box, you specify whether the wizard will create make files and shell scripts. Finally, click Finish to create the new application. The wizard will display in the shell window the names of all the files it is creating.

Building and running the application from the command line

To build and run the project from the command line:

1 Make the project directory the active directory:

```
cd mywapapp
```

2 Use the `make` command to build the application:

```
make
```

3 Change to the output directory and start the application:

```
cd output
./start
```

4 Start a wireless device emulator (YoSpace or the Nokia Toolkit) and load the URL:

http://localhost:9000/

See "Option 2: Building using JBuilder and Kelp" on page 37 for an illustration of the application running in the Nokia Toolkit emulator.

Chapter 4

Designing wireless applications

This chapter describes how to design wireless applications. Design is actually the third step in the simplified application development process described in *Getting Started with Lutris Enhydra*. To create an effective design, you must have completed the first two steps, requirements definition and functional specification.

Technically, design includes a description of an application's

- Database schema and corresponding data layer
- Business layer, including business logic
- Presentation layer, including the overall user interface

This chapter focuses on the unique requirements imposed on the presentation layer by wireless applications.

General design considerations

When you design a wireless application, consider the following:

- The key desirable features that will make an application compelling
- Use patterns: what the application actually does, and how people will use it
- Device constraints: capabilities of wireless devices and networks

Key features

A good wireless application provides time-sensitive information in a short, simple transaction. Some additional key features of a wireless application are:

- A Web profile for initialization, customization, and data entry
- Personalized features
- Location-based features
- Meaningfully partitioned content

Web profile

A Web profile provides an interface through which users can initially set preferences and customize the content they will see in a wireless profile. An application should also generally use a Web profile for extensive data entry.

For example, the AirSent application has a Web profile through which customers enter the pickup and delivery information, and administrators set the courier badge

and password strings. Once a courier logs in, a wireless profile is preconfigured with all the pertinent information for that courier. All the courier has to do is press one button to click on a link to indicate when he or she has picked up and delivered a package.

Personalization

Personalization means that the content a wireless user sees is pertinent and relevant, which is crucial for ensuring that a wireless application is useful. Since display space is at a premium, personalization eliminates irrelevant information and the need for text entry. This feature ties in with a Web profile, because the Web interface will be where the user enters the personalized information.

For example, a stock quote application might be personalized to show only the stocks in the user's portfolio. The Web profile allows the user to enter the stocks in his or her portfolio, and to indicate what information he or she wants to see in the wireless profile (bid price, ask price, last trade price, volume, and so on). When using the wireless device, the user logs on, clicks "stocks," and then sees exactly the information he or she is interested in—nothing more, and nothing less.

You can take several approaches to personalization:

- Personalize based on user input. This is the most straightforward, because the user specifies what content he or she wants to see; typically, the user would enter this information from the application's Web profile. For example, the user could enter the name of a city for which he or she wants to see weather information when using a wireless profile.
- Personalize based on user's demographic information. The Web profile allows the user to enter information (for example: address, age, occupation) that the application uses to personalize content. For example, the application could provide weather for the user's home city, once the user entered his or her address information.
- Personalize based on user behavior. The application tracks the user's actions and responses, and deduces personal preferences. For example, an application could "remember" the last city for which the user requested weather, and put that city at the top of the pick list.

Location-based features

As described in "Location-based services" on page 23, location-based services (LBS) provide content based on a mobile user's location. While the technology for this is still evolving, it is widely seen as a key feature for "killer" wireless applications. Because wireless users are inherently mobile, relevant information is often dependent on their location.

Most applications of LBS can be grouped into three broad categories:

- Finding the nearest desired thing or place. For example:
  - "Where is the nearest Chinese restaurant?"
  - "I'm lost; where is the nearest subway station?"
  - "Where is the nearest gas station?"

- Providing directions from current location to desired location. For example:
  - "How do I get back to the freeway from here?"
  - "How do I get to that gas station (or Chinese restaurant)?"
- Locating a wireless device. For example:
  - "I lost my PDA—where is it now?"
  - "I forgot where I parked, where is my car?"
  - "Where is my child?"

Of course, there are nearly infinite variations on these concepts, but these are the primary useful groupings.

LBS technology alone is just the first step in providing applications with such capabilities. They also require extensive detailed map databases, plus up-to-date information on business and residential locations. Various providers are already lining up to provide some of these services.

Meaningfully partitioned content

A well designed wireless application provides small chunks of information, with multiple hyperlinks for further detail. The most important and popular items should be at the top of the hierarchy, and more detailed, specific information available through successive links. This chunking makes information more accessible to mobile users, who are in a hurry and using devices with limited screen space and bandwidth.

To partition information meaningfully, you must make a tradeoff between the amount of data and the number of hyperlinks on each page (or card). More links per page exponentially increases the number of pages the user can reach with a set number of button clicks, but may also increase the amount of scrolling required.

For example, as illustrated in Figure 4.1, if you have three links per page (shown in the figure on the left), then two button clicks can bring the user to nine data pages; five links per page (shown in the figure on the right) means three clicks will lead the user to 25 data pages.

Figure 4.1 Number of links per page increases number of data pages exponentially

In general, the number of data pages, d, that a user can reach is

$d = l^n$

where l is the number of links per page and n is the number of clicks the user is required to make. In Figure 4.1, n is two, so three links per page ($l = 3$) results in nine data pages; five links per page ($l = 5$) results in 25 data pages.

Use patterns

Why use a wireless application, after all? To be successful, a wireless application must provide significant value—it must provide information or capabilities not available through other means (voice phone calls, television, radio, the wired Web, and so on). Wireless applications must meet the needs of wireless users, which are distinct from the needs of Web users, because wireless users are mobile, in a hurry, and are using a radically different interface from a Web PC.

Wireless users are by definition mobile users "on the go." They generally won't be sitting around their homes or offices, but will be traveling in a strange city, walking down the street, or riding on a train or in a car. You must design wireless applications with these use patterns in mind.

Wireless users are also generally in a hurry, so the application should not take up too much of their precious time. They want the information *now*, not after clicking through innumerable pages or entering copious data. The information they are seeking is timely and relevant to them.

Personas and profiles

A *persona* is a particular class of user, classified by device and by use pattern. For example, here are some hypothetical personas:

- Business travellers with WAP phones
- Package couriers with Java phones
- technical managers with PDAs
- customers with web PCs

An application's *profile* (sometimes called a *skin*) is a presentation tailored for a particular persona. So, an application could have a profile for each of the personas listed above. The device requirements, together with the way each persona will use the application will dictate design decisions.

Note Confusingly, profiles are sometimes referred to as personas, and profiles as skins. However, it is important to distinguish between the two concepts.

Device constraints

There are a wide variety of wireless devices, but most of them have these typical characteristics:

- Bandwidth is limited
- Graphics are generally monochrome; if available, colors are limited

- Text entry or input other than selecting is difficult
- Scrolling may be awkward

The specific constraints on an application will depend on the devices it is targeting. For example, devices such as cellphones are very limited in their display and text input capabilities. Entering text on a cellphone generally requires you to press a key multiple times to enter a single character, making text entry very tedious. PDAs and handhelds are less limited, but are still constrained compared to PCs with full-sized keyboards. For information on device constraints for WAP and i-mode devices, see "Wireless technology limitations" on page 63.

To make your application usable from a variety of devices, you can take two approaches:

- Design for the "lowest common denominator" device, ensuring that application will work reasonably on all required devices.
- Design for a "target device" that represents the largest (or most important) fraction of users; then create a "lowest common denominator" profile for the rest.

Sophisticated devices can display simple designs, but the reverse is generally not true: More complex designs do not display well on less capable devices. Therefore, the "lowest common denominator" profile will generally work on a wide range of devices; however it may not provide a compelling interface. For this reason, if you can identify a particularly significant persona, it makes sense to create a tailored profile for that persona. For example, if your application targets business managers at work, you may want to target Palm OS, but also provide an interface for WAP phone users.

If a wide variety of devices will access your application, it may make sense to identify several classes of devices with distinct constraints, and then design a profile for each class. For information on creating multiple application personas, see "Applications with multiple profiles" on page 62.

Interface guidelines

The following general guidelines for designing for wireless presentations are based on common use patterns and device constraints.

- Minimize text and use abbreviations where possible.
- Minimize use of graphics because they consume bandwidth and don't display consistently. Exception: cHTML includes a set of built-in icons that may be useful for some applications.
- Use only form elements available in the markup language you are targeting (WML, cHTML, and so on).
- Make sure each page displays without horizontal scrolling. Minimize vertical scrolling as much as possible.
- Avoid text entry, especially if the target device is a cellphone.

Some applications may require bending or breaking these guidelines, depending on the specific use patterns and target device constraints.

Layout and design

Layout and user interface design for a wireless application is significantly different than for a Web application. Because of the limited screen space and keypad keys, you must pay special attention to the overall user experience; otherwise users will become frustrated, and the application will not be useful.

Designing a wireless storyboard

Creatinga storyboard is particularly important for wireless applications, because there may be a variety of distinct user roles and use cases. For example, an administrator may access the application from a PC-based Web browser, while a regular user accesses it from a Web phone. The application should provide information-rich HTML pages to the administrator while showing the user simple, brief WML (or cHTML) pages, suitable for display on a small Web phone display.

In most cases, you can't use the same storyboard for wireless and Web applications because the user experience with wireless devices is very different from that with a Web browser. Also, microbrowsers have limited capabilities that change what you can display on a single screen.

Navigation

Wireless applications should enable users to quickly access the information or functionality in which they are interested. To facilitate easy information access:

- Instead of prompting for information, provide links. For example, instead of requiring a user to enter a search term, provide a series of links to allow the user to define the term.
- Minimize the number of button clicks the user must perform to complete a task or get the desired information: Make the site hierarchy flat.

Applications with multiple profiles

An application's *profile* is a presentation tailored for a particular persona. For example, an application could have an HTML profile, WML profile, and a cHTML profile. The first would be appropriate for someone viewing the application on a PC Web browser (Web personas); the second for someone using a WAP phone (WAP personas), and the third for someone using an i-mode phone (i-mode personas).

Some developers have to create multiple parallel applications to support different profiles. However, with Enhydra, you can easily create a single application that has multiple profiles. This reduces application development and maintenance costs.

Wireless technology limitations

Wireless technologies have different limitations. This section outlines the limitations for the most common technologies.

WAP

WAP has limits on file and image size, and WAP devices have display limitations.

File and display size

The maximum allowable file size varies from one phone to another. However, as a guideline, you should keep compiled pages under 1,400 bytes, and a maximum of 1258 bytes per card.

The lowest common denominator display limits for WAP cellphones are:

- 4 lines
- 12 characters per line

Images

The WAP standard specifies the wireless bitmap (WBMP) image format. WBMP Type 0 is a simple monochrome, uncompressed bitmap. Details of this format are provided in the WAE Specification at http://www.wapforum.org.

i-mode

The i-mode system has a number of specific limitations that you must consider when developing applications.

General specifications

Currently, i-mode is available in Japan, and applications targeting i-mode devices must use:

- Shift-JIS character encoding (Half-width kana characters can be used)
- cHTML (see the following section)
- Images in GIF format

File and display size

Technically, the absolute maximum allowable size of a cHTML page is 5 KB (5000 bytes). Depending on the tags being used, devices may not be able to display some pages even though they contain less than 5 KB. Therefore, the recommended maximum page size is 2 KB.

The lowest common denominator display limits for i-mode cellphones are:

- 6 lines
- 8 double-byte characters per line

Some i-mode phones have displays as large as 10 characters by 10 lines, but if you target your display for 6 x 8, it will work properly on all i-mode phones.

Note i-mode phones do not support horizontal scrolling.

For complete detailed display information on i-mode phones, see http://www.nttdocomo.com/i/tag/s5.html.

Images

i-mode supports 256-color GIF files, specifically GIF 87, 87a, and 89a formats. The maximum size of a GIF image is 94 x 72 dots.

Other limitations

An i-mode application must observe these maximum size limits:

- Maximum length of a URL is 100 bytes.
- Maximum buffer size of INPUT elements is 512 bytes.
- Maximum buffer size of SELECT elements is 4096 bytes.
- Maximum length of a character string is 200 bytes after URL encoding.
- Maximum length of the title of a page/bookmark is 24 bytes.

Chapter 5

Survey of wireless presentation languages

The most popular languages for wireless presentation are WML, cHTML, and voiceXML. XHTML is widely considered to be the future language of choice. This section provides an introduction to each of these languages, assuming you are already familiar with HTML.

There are two broad categories of wireless presentation languages:

- Languages derived from XML (technically known as XML applications), including WML, voiceXML, and custom XML derivatives
- HTML-like languages, including cHTML and XHTML. Although XHTML is technically an XML application, its elements are very similar to HTML

Both categories of languages share common concepts and terminology, discussed in this section.

Note The terms *elements* and *tags* are often used interchangeably. This chapter uses the term elements.

Extensible Markup Language (XML)

Extensible Markup Language (XML), a metalanguage created by the World Wide Web Consortium (W3C), is used to define markup languages such as WML, VoiceXML, and XHTML.

Though XML is similar in theory to HTML, its use is quite different in practice. HTML was designed to display data; XML was designed to describe data. For example, if you were creating an HTML document to display an address card, you would use elements to format the card. For example, you'd use an <h1> element for the name and body elements for the address and phone number.

With XML, you're more concerned with describing the data on the address card. To describe an address card, you might create elements such as <name>, <address>, and <phone>.

XML describes the content of a document; it does not describe what that content is.

Concepts and terminology

Although XML and HTML-like languages are fundamentally different in many ways, they are based on some similar concepts such as elements and attributes.

Elements

XML documents consist of a series of nested *elements*, also called *tags*. A starting element is simply the element name enclosed in angle brackets:

```
<element>
```

An ending element is the element name preceded by a slash, enclosed in angle brackets:

```
</element>
```

Anything between a start element and an end element is the *content* of the element. For example:

```
<element> content </element>
```

If an element has no content, you can abbreviate it as a single element with a trailing slash. For example:

```
<element/>
```

In HTML, element names are case insensitive; in contrast, XML element names are case sensitive.

Attributes

An element can have any number of attributes, which are name-value pairs that appear after the element name and before the closing angle bracket:

```
<element attribute1=value1 attribute2=value2 ... >
```

Some HTML elements can have an attribute without a value, though this is not legal in XML. In XML, element attributes must have a value (this is different from HTML, in which element attributes do not have to have a value).

Entities

Entities are storage units; you can think of them as constants. An XML entity can be a separate file, a string, or even a database record. When you use an entity name elsewhere in a DTD or XML document, the parser replaces the entity name with the corresponding value.

Document Type Definitions (DTDs)

A Document Type Definition (DTD) defines the legal entities and elements that can be used in a specific XML document. DTDs can be declared within an XML document or as an external reference.

An XML document is considered *well-formed* if its syntax is correct; it conforms to XML syntax rules. An XML document is considered *valid* when it has been checked

successfully against a DTD; the document conforms to the rules established in a DTD.

A DTD defines everything about an XML document, including:

- the elements allowed in a document
- the attributes that each element can have; optionally the allowable attribute values
- how elements can be nested

The AirSent application contains a simple example DTD in the directory `<enhydra_root>/examples/AirSent/po/src/com/lutris/airsent/resources/messenger/xml/dtd`.

In general, a DTD specifies elements with this notation:

```
<!ELEMENT element-name [(nested-element1, nested-element2, ...)]>
```

where *element-name* is the name of the element being defined, followed by a comma-delimited list of *nested-element* that enumerate all the elements that can be nested inside it. (The square brackets indicate an item that is optional.)

For example, the AirSent DTD defines a Details element that can contain ErrorText, Name, Address, and Directions elements:

```
<!ELEMENT Details ( ErrorText, Name, Address, Directions)>
```

To indicate that an element can contain only a text child, use the notation #PCDATA, which stands for "parsed character data." For example, the AirSent DTD specifies that the DisplayTemplateName element can only have text children like this:

```
<!ELEMENT  DisplayTemplateName  (#PCDATA)>
```

A DTD defines an element's attributes with this notation:

```
<!ATTLIST [enclosing-element] [attribute-name] [type] [#REQUIRED | #IMPLIED | #FIXED]>
```

where *enclosing-element* is the name of the element, *attribute-name* is the name of the attribute, *type* is its type, followed by a keyword that indicates whether the attribute is required.

The type can be one of:

- CDATA: Text
- enumerated: An exact list of options as (`option1 | option2 | option3` ...)
- ID: A unique name for the element
- IDREF: The value of an ID type attribute
- IDREFS: Multiple IDs, separated by whitespace
- ENTITY: The name of an entity declared in the DTD
- ENTITIES: Multiple entities, separated by whitespace
- NMTOKEN: An XML name
- NMTOKENS: Multiple XML names, separated by whitespace
- NOTATION: The name of a notation declared in the DTD

To indicate that an element must contain the attribute, use the keyword #REQUIRED. To indicate that the attribute is optional, use #IMPLIED. The #FIXED option is rarely used for applications.

For example, the AirSent DTD defines the attributes for the DisplayTemplateName element as follows:

```
<!ATTLIST  DisplayTemplateName
           id ID #IMPLIED
           href CDATA #IMPLIED>
```

Wireless Markup Language (WML)

WML uses the metaphor of a deck of cards to represent a document. A document (or *deck*) consists of a set of one or more *cards*. Each card represents a single (scrollable) screen of display text. This scheme enables a wireless device to download multiple screens with one retrieval, then navigate among them without accessing the network again, an advantage when bandwidth is at a premium.

For a comprehensive WML reference, see Appendix C, "WML reference."

Refer to these websites for additional information onWML:

- http://developer.phone.com (requires registration)
- http://www.AnywhereYouGo.com/ayg/ayg/Content.po?name=wap/Wmlidx
- http://www.wirelessdevnet.com/channels/refview.phtml?cat=wmltags

Document structure

Strictly speaking, a WML document consists of:

- A *prolog*, which specifies the version of XML and the Document Type Definition (DTD) being used.
- A single <WML> element that contains
  - The optional <HEAD> element document *heading*, containing meta-information that applies to all the cards in the deck.
  - One or more <CARD> elements, containing the remainder of the content.

Note The prolog contains a reference to a WML DTD, usually the standard DTD hosted on the WAP Forum website. To enable offline development, refer to a local copy of the DTD, rather than to a copy on the network.

First WML document

This is a very simple "Hello World" example. It contains one card with a brief message:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
         "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
   <card id="hello" title="Hello World">
      <p>
         <em id="greeting">Hello Wireless World!</em>
      </p>
   </card>
</wml>
```

This WML document does not have a heading, and contains a single card that displays the text, "Hello Wireless World!"

Meta elements

Like HTML, WML has a `<meta>` element that you can put in the document heading to specify "meta-information" for a deck. This element is basically a catch-all for functions not provided by other elements. There are innumerable attributes of the meta element for specific microbrowsers. However, some of the common attributes are:

- `forua` ("for user-agent"), which specifies whether the content of the meta element should be passed on to the microbrowser or stripped out by the WAP gateway. If `forua` is true, the meta element is passed on to the client; otherwise, the gateway is to follow any directives of the meta element, and then remove it from the document.
- `http-equiv` ("http equivalent"), which specifies a value that is equivalent to an HTTP header value.

One use for the meta element is to disable microbrowser caching of WML documents. As explained in "Disabling caching" on page 102, you should disable caching for pages that are dynamically generated by Enhydra. Use the following meta element:

```
<meta forua="true" http-equiv="Cache-Control" content="max-age=0"/>
```

Other uses for the `<meta>` element are generally microbrowser-specific. Refer to your microbrowser documentation.

Card element

The card element contains all the content actually displayed by a microbrowser. A card is intended to encapsulate information that can be displayed in a single screen, though this may not always be the case on all devices.

Table 5.1 lists the key attributes of the card element. Although the WAP standard specifies other attributes, they are not widely supported.

Table 5.1 Important attributes of the card element

Attribute	Value Type	Description
id	String	Sets the unique name for the element.
onenterbackward	URL	This event occurs when the card is entered from a `<prev>` task.
onenterforward	URL	This event occurs when the card is entered from a `<go>` task.
ontimer	URL	This event occurs when a `<timer>` expires.
ordered	True or False	Tells the user agent if the card content should be ordered in the display.
title	String	The card's title. It is used, for example, in bookmarks.

The URL of a specific card is constructed from the URL of the containing document, followed by the pound sign (#) and the card's id. For example, if the URL of the "Hello World" document above is http://www.mywap.com/helloWorld.wml, then the card's URL would be http://www.mywap.com/helloWorld.wml#hello.

Events

WML has four *events*:

- *onenterforward* occurs when you navigate to a card using a `<go>` element.
- *onenterbackward* occurs when you navigate to a card using a `<prev/>` element, or when you press the PREV key.
- *onpick* occurs when you select or deselect an item defined by an `<option>` element.
- *ontimer* occurs when a timer (defined by the `<timer>` element) expires.

To specify what happens when an event occurs, you bind an event to a *task*. You can bind the onenterforward, onenterbackward, and ontimer events to URLs in the attributes of the card element. You can also bind an event to any task in the `<onevent>` element. For information on tasks, see the next section, "Tasks" on page 70.

Example

Here is an example of a card that defines an onenterbackward event that immediately returns the browser to the card it was displaying. The result is that this card cannot be displayed as a result of pressing the PREV key or a `<prev/>` operation:

```
<card ...>
<onevent type="onenterbackward">
<prev/>
</onevent>
...
</card>
```

Tasks

Tasks are operations that the microbrowser performs in response to some user action. The following WML elements define *tasks*:

- `<go>` is a navigation task that loads the specified URL.
- `<prev>` is a navigation task that loads the previous card in the browser history list.
- `<noop>` is a "no-op" that does nothing.
- `<refresh>` redisplays the card and resets the variables, as specified by the contained `<setvar>` elements.

You can assign a task to an event using the `<onevent>` element, as described in the previous section. You can also assign tasks to particular user interface elements with the `<do>` element. The `<do>` element has the following syntax:

```
<do type="UI-element">
   task
</do>
```

where *UI-element* is the UI element that triggers the task, and *task* is a task element that defines the action that will occur when the user invokes the element. In most cases, the UI element is a softkey that the user can press to invoke the task. Table 5.2 describes the interface element types that can be bound to tasks.

Table 5.2 Interface element types that can be bound to tasks with <do>

UI element	Description
accept	ACCEPT mechanism
delete	DELETE mechanism
help	HELP mechanism
options	OPTIONS mechanism
prev	PREV mechanism from another card
reset	RESET mechanism (clears or resets current device state)
unknown	Invokes unknown mechanism (equivalent to `TYPE=""`)

For example, this code associates the <prev/> event with the PREV softkey:

```
<do type="prev">
   <prev/>
<prev/>
```

Navigation elements

Like HTML, navigation in WML is performed with links. However, you can also use `<select>` lists

Table 5.3 WML navigation elements

Element	Attributes	Description
<a>	href (required)	Equivalent to: `<anchor><go href="url"/></anchor>`
<anchor>	title (optional)	Anchors a task to the content text. Must contain both a task and link text.
<go>	accept-charset (optional) href (required) method (optional) sendreferer (optional)	Causes the browser to open the URL specified by the href attribute. If the URL specifies a card, the browser displays that card. If the URL specifies a deck, the device displays the first card in that deck.
<option>	onpick (optional) name (optional)	Specifies a particular choice within a <select> element, and labels the action key with the name attribute.
<select>	default (optional) idefault (optional) ikey (optional) key (optional) title (optional)	Specifies a list of options from which the user can choose. Use the <option> elements within the <select> element to create the options.

Using the Wireless Telephony Application Interface (WTAI)

Devices that support the WAP 1.2 specification implement the *wireless telephony application interface* (WTAI), which provides a way for WAP applications to access services either on a wireless device or on the network.

WTAI has functions that can:

- Dial a phone call
- Send DTMF tones
- Add a phone book entry

WTAI defines functions that are accessible from WMLScript and from WML code. In WML code, a URL that begins with "wtai://" executes a WTAI function. For example, the following code dials a phone number (instead of requesting a new WML page):

```
<A HREF="wtai://wp/mc;8315551212" id="phoneNumber">Dial Information</A>
```

When the user clicks this link, the phone calls the WTAI function to dial the phone number.

Formatting elements

WML provides a limited set of formatting elements, which may be interpreted differently by different devices, or ignored altogether. In general, these formatting elements correspond to similar HTML elements.

Element	Attributes	Description
<b>	(none)	Boldface text
<big>	(none)	Display text in larger font
 	(none)	Line break
	(none)	Display text with emphasis
<i>	(none)	Italicize text/
<img>	alt src localsrc height width align	Display image
<p>	align wrap	Paragraph
<small>	(none)	Display text in small font
	(none)	Display text with emphasis
<table>	columns align title	Table
<td>	(none)	Table cell
<tr>	(none)	Table row
<u>	(none)	Underline text

Variables

WML includes the concept of *variables*, which are values that can be set at runtime within a document, using the following syntax:

```
<setvar name="varName" value="varValue"/>
```

where varName is the name of the variable, and varValue is the value you are assigning to it. Variable names must begin with a letter or underscore, and the remaining characters in the name may be letters, numbers, or underscores. Variable values are always strings. So, for example, the following statement sets the variable named `userName2` to "Jerry":

```
<setvar name="userName2" value="Jerry"/>
```

Once defined, a variable is global to all decks and cards on a given microbrowser for a whole session.

You use a variable with the following syntax:

```
$varName
```

or

```
$(varName)
```

So, to use the variable defined in the previous example, you could do something like this:

```
<P>
Hello, $userName2
```

The user would then see a display like this:

```
Hello, Jerry
```

Reserved characters

Certain characters are reserved in WML to have special meaning. For example, the less than symbol (<) denotes the beginning of an element. To use any of the reserved characters in a document, you must use the WML character entity instead. Table 5.4 lists the WML character entities. Most of these are the same as HTML character entities, with the exception of the dollar sign, which you have to escape in WML but not HTML.

Table 5.4 WML character entities

Character	WML Entity	Description
<	<	less-than sign
>	>	greater-than sign
'	'	apostrophe
"	"	quotation marks
&	&	ampersand
$	$$	dollar sign

Note Unlike HTML, you **must** use the special character element `&` to specify the ampersand (`&`) character when separating parameters in URL strings.

Compact HTML (cHTML)

cHTML is a subset of HTML, with a few extra features. NTT DoCoMo defines two levels of cHTML: level one, supported by all i-mode phones, and level two, supported by 502i series phones.

For a comprehensive cHTML reference, see Appendix D, "cHTML reference."

Refer to these websites for additional information on cHTML:

- NTT DoCoMo reference site: http://www.nttdocomo.com/i/index.html
- W3c Note on cHTML: http://www.w3.org/TR/1998/NOTE-compactHTML-19980209/

Differences from HTML

NTT DoCoMo documents several features of cHTML that are not part of HTML; however, these features are not part of the W3C Note on cHTML. The extensions include the accesskey attribute and i-mode icon symbols.

accesskey attribute

cHTML includes an extra attribute, `accesskey`, for the elements:

- `<A>`
- `<INPUT>`
- `<TEXTAREA>`

The `accesskey` attribute designates a keypad key that the user can press to select the item instead of moving the cursor to the link or form element. You can designate any of the numbers 0-9 and the keys star (*) and pound (#) as an accesskey.

For example:

```
<A HREF="http://www.mysite.com/phone.chtml" ACCESSKEY="1">Call Office</A>
```

Note Some i-mode terminals (notably, the D502i model) do not allow star (*) and pound (#) keys to be used as accesskeys.

Icon symbols

i-mode defines a set of 166 special icon symbols (also called *emoji*) that you can embed in a page using a six-byte decimal code (`&#dec-code;`). For example: `` designates picture symbol No. 1, which creates the following "shining sun" icon:

For more information on the i-mode icons, see http://www.nttdocomo.com/i/tag/emoji/index.html.

HTML 1.0 elements

These elements are supported by all i-mode devices. If you want your application to be viewable on all devices, you should limit your pages to these elements. They all correspond to HTML 1.0 elements.

General elements

These elements are general document elements that define a page:

Table 5.5 General cHTML elements

Element	Attributes	Description
`<!-- comment -->`	(none)	Defines a comment that is not displayed by device.
`<BODY>`	(none)	Container for viewable cHTML content.
`<HEAD>`	(none)	Container for page header.
`<HTML>`	version	Container for all cHTML content.
`<TITLE>`	(none)	Defines page title.

Layout

Use these elements for general layout functions:

Table 5.6 cHTML layout elements

Element	Attributes	Description
` `	clear= “{left | right | all}”	Designates a line break. The clear attribute cancels character string wraparound after the element.
`<CENTER>`	(none)	Centers content text.
`<DIV>`	align= “{left | center | right}”	Aligns content text based on value of align attribute.
`<HR>`	align=”{left | center | right"}	Inserts a horizontal line.
	size=”*no. pixels*”	The size attribute specifies the thickness of the line.
	width= “{*no. pixels* | *width%*}”	The width attribute specifies the line width.
`<IMG>`	src="*filename*"	Inserts an image from file named *filename.*
	align= "{top | middle | bottom | left | right}"	The align attribute specifies horizontal alignment.
	width= "{*no. pixels* | *width%*}” height= "{*no. pixels* | *width%*}”	The width and height attributes designate width and height of image in absolute pixels or percentage of screen.
	hspace= "*no. pixels*" vspace= "*no. pixels*"	The hspace and vspace attributes specify horizontal and vertical space surrounding image, respectively.
	alt= "*text*"	The alt attribute specifies alternate text to display when image is not displayed.
`<PLAINTEXT>`	(none)	Unformatted text.

Text formatting

Use these elements for text formatting:

Table 5.7 cHTML text formatting elements

Element	Attributes	Description
<BLOCKQUOTE>	(none)	Indents content text by one full space from left and right margin.
<H1> ...<H6>	align= {"left | center | right"}	Heading text.
<P>	align= {"left | center | right"}	Displays content text as a paragraph, with specified alignment.
<PRE>	(none)	Displays content text, including linefeeds and blank spaces, exactly as entered.

In general, headings are displayed in an H4-equivalent font size for all the heading elements <H1> through <H6>.

Form elements

Use these elements to define forms and form elements:

Table 5.8 cHTML form elements

Element	Attributes	Description
<FORM>	action= "*URL*" method= "get | post"	Container for forms.
<INPUT>	type= "{text | password | checkbox | radio | hidden | submit | reset" name= "*fieldname*" value= "*default-value*" size= "*number*" maxlength= "*number*" checked accesskey= "char"	Defines form elements.
<OPTION>	value= *"value"*	Defines an option inside a <SELECT> element.
<SELECT>	name= "*fieldname*" size= "*number*"	Defines a list of optional elements; each element is defined by an <OPTION> element.
<TEXTAREA>	name= "*fieldname*" rows= "*number*" cols= "*number*"	Defines a multiline text input field with height specified by the rows attribute and width specified by the cols attribute.

Link-related elements

Use these elements to define hyperlinks:

Table 5.9 cHTML link elements

Element	Attributes	Description
<A>	name="*name*" href="*URL*" accesskey="*char*"	Defines an anchor; typically a hyperlink.
<BASE>	href="*URL*"	Defines base URL for anchor (<A>) elements.

Lists

These elements are used to define lists:

Table 5.10 cHTML list elements

Element	Attributes	Description
<DL>, <DT>, and <DD>	(none)	Definition list container, term, and definition. <DT> items start on a new line. <DD> items start on a new line and are indented.
<LI>	(none)	List item (for use with bullet lists).
<MENU> , <UL> and <DIR>	(none)	Container for bullet lists. (Display is the same for all three elements.)
<OL>	(none)	Container for a numbered list.

HTML 2.0 elements

These additional elements and attributes are supported by some i-mode phones (for details, see the DoCoMo website):

Table 5.11 i-mode compatible HTML 2.0

Element	Attributes	Description
<BLINK>	(none)	Creates a blinking character string.
<BODY> (additional attributes)	bgcolor text link	Specifies colors of the background, text and links.
<FONT>	color	Specifies font color.
<INPUT type="text"> (additional attribute)	istyle	Sets the default character input mode (alphabetic, numeric, etc.).
<LI> (additional attributes)	type value	Specifies type and number of bullet item.
<MARQUEE>	(none)	Scrolls a character string horizontally.
<OL> (additional attributes)	type start	Specifies type and starting number of bullet list.

Table 5.11 i-mode compatible HTML 2.0 (continued)

Element	Attributes	Description
<SELECT>		Makes a character string change lines automatically for full onscreen display regardless of model.
<TEXTAREA> (additional attribute)	istyle	Sets the default character input mode (alphabetic, numeric, etc.).

Voice Extensible Markup Language (VoiceXML)

VoiceXML is an XML derivative for creating applications with a verbal—as opposed to visual—user interface, typically with a standard telephone as the end-user client device. Such applications are known broadly as *voice applications*. In a voice application, input consists of spoken commands or telephone touch-tone keys (also known as dual-tone multi-frequency or DTMF), and output consists of listening to synthesized and recorded speech output.

By its very nature, the design of a voice application interface is completely different from a graphical interface, such as that of a WAP or Web application. The design principles and concerns for voice applications are very different from the design principles and concerns for GUI applications. This section presents the fundamental concepts of VoiceXML and voice applications.

For a comprehensive VoiceXML reference, see Appendix E, “VoiceXML reference,” on page 163 (online only).

Refer to these websites for additional information on VoiceXML:

- Tellme Networks developer site: http://studio.tellme.com
- VoiceXML Forum site: http://www.voicexml.org

Fundamental concepts and terminology

Some of the fundamental VoiceXML concepts are applications, documents, and dialogs. An application contains documents, and a document contains dialogs.

Applications

A VoiceXML application consists of one or more documents sharing the same application root document. When a user interacts with a document in an application, the system loads the application root document; it unloads it when the user interacts with a document in a different application. While the application root document is loaded, its variables are available to any documents in the application.

Documents

A VoiceXML document is composed primarily of top-level elements called *dialogs*. Users are always in one conversational state, or dialog, at a time. Each dialog determines the next dialog to transition to. Transitions are specified using Universal

Resource Indicators (URIs), which define the next document and dialog to use. Execution is complete when a dialog does not specify a successor, or when a dialog has an element that explicitly exits the conversation.

In addition to forms and menus, a document may also have `<meta>` elements, `<var>` and `<script>` elements, `<property>` elements, `<catch>` elements, and `<link>` elements.

Dialogs

There are two kinds of dialogs: *forms* (defined by the `<form>` element) and *menus* (defined by the `<menu>` element). Forms present information and gather input; menus offer choices of what a user can do next. A menu is a convenient syntactic shorthand for a form containing a single anonymous field that prompts the user to make a choice and then transitions to different places based on that choice.

A form can contain various elements to present prompts to the user, gather input, and handle error conditions. Based on the user's actions, elements of the form may be used to determine which subdialog to execute next.

A form contains:

- A set of form items, elements that are visited in the main loop of the form interpretation algorithm. Form items include:
  - Field items, which define the form's field item variables
  - Control items, which help control the gathering of the form's fields
- Declarations of non-field item variables.
- Event handlers.
- "Filled" actions, blocks of procedural logic that execute when certain combinations of field items are filled in.

A first VoiceXML document

The following is a very simple VoiceXML document.

```
<?xml version="1.0"?>
<vxml version="1.0">
   <form>
      <block>Hello World!</block>
   </form>
</vxml>
```

This document contains a single form dialog that contains a block that synthesizes (speaks) "Hello World!" to the user.

Further concepts

Additional VoiceXML constructs include subdialogs, sessions, grammars, events, and links.

Subdialogs

A subdialog is like a function call, in that it provides a mechanism for invoking a new interaction and returning to the original form. For example, subdialogs can be used to create the following:

- A confirmation sequence that requires a database query
- A set of components that can be shared among documents in a single application
- A reusable library of dialogs shared among many applications

Sessions

A session begins when the user starts to interact with a VoiceXML interpreter context, continues as documents are loaded and processed, and ends when requested to do so by the user, a document, or the interpreter context.

Grammars

Each dialog has one or more speech or dual-tone multi-frequency (DTMF) grammars associated with it. In machine-directed applications, a dialog's grammar is active only when the user is in that dialog. When the user and computer alternate in determining what to do next (mixed initiative applications), some dialogs are flagged to make their grammars active even when the user is in another dialog in the same document (or on another loaded document in the same application). If the user says something matching another dialog's active grammars, execution transitions to that other dialog, with the user's utterance treated as if it were said in that dialog.

A grammar defines the set of valid expressions that a user can say when interacting with your phone application. You can define valid expressions for your application within a single grammar, but we recommend that you categorize expressions into separate grammars that have limited scopes. For example, a typical phone application provides instructions for:

- Navigating through menus and choosing an option
- Obtaining more information about an option
- Asking for help

To simplify your application, you should divide the commands for each type of action into a separate grammar. In addition to making your application easier to maintain, this approach enables you to control the scope of each grammar.

Events

VoiceXML provides a form-filling mechanism for handling standard user input. VoiceXML also defines a mechanism for handling events not covered by the form mechanism.

Events are triggered by certain circumstances. For example, when the user

- Does not respond
- Does not respond intelligibly
- Requests help

The interpreter also triggers events if it finds a semantic error in a VoiceXML document.

Events are *caught* by `<catch>` elements. Each element in which an event can occur may specify catch elements. Catch elements are also inherited from enclosing elements "as if by copy." In this way, common event handling behavior can be specified at any level, and it applies to all lower levels.

Links

A link supports applications in which both the computer and the user determine what happens. It specifies a grammar that is active whenever the user is in the scope of the link. If user input matches the link's grammar, control transfers to the link's destination URI. A <link> element can be used to throw an event to go to a destination URI.

Example

The following VoiceXML example shows a simple application. It prompts the user to choose from two options: Order or Support. If the user says Order, it runs order.vxml; if the user says Support, it runs support.vxml.

The <default> element handles any response that's not an Order or Support command. Here, it includes a <reprompt> command that directs the server to replay the items in the prompt list.

```
<?xml version="1.0"?>
<vxml version="1.0">
<menu id="Order_Example">
<prompt>
<audio>Welecome to the Order by Voice system.  Say Order to place an order or Support to
reach a technical support representative.</audio>
</prompt>
<choice next="order.vxml">
Order
</choice>
<choice next="support.vxml">
Support
</choice>
<default><reprompt/></default>
</menu>
</vxml>
```

VoiceXML Elements

This section provides a summary of the elements in VoiceXML, grouped functionally.

Document elements

These elements are general elements for defining VoiceXML documents and document structure.

Table 5.12 VoiceXML document elements

Element	Description
<dtmf>	Defines the permissible DTMF keys for a given user interaction. Use in <field>, <form>, <link>, <transfer>, and <vxml> elements.
<form>	Contains grammars, prompts, event handlers, and directives.
<grammar>	Defines the permissible vocabulary for a given user interaction. Use within <field>, <form>, <link>, <transfer>, and <vxml> elements.
<link>	Specifies one or more grammars and a destination to go to when one is matched.
<meta>	Specifies general information about the VoiceXML document.
<var>	Declares a local variable. Use within <block>, <filled>, <noinput>, <result>, <catch>, <help>, <nomatch>, <vxml>, <default>, <initial>, and <prompt> elements.
<vxml>	Identifies a document as a VoiceXML document.

Form elements

These elements are used to define forms and form elements.

Table 5.13 VoiceXML form elements

Element	Description
<block>	Specifies a block of directives that are executed in document order.
<field>	Specifies input to be gathered from the user.
<filled>	Specifies an action to perform when a <field> variable is assigned a value.
<initial>	Specifies a set of directives that are executed in document order.
<record>	Records user input.
<result>	Specifies an action to take for a specific recognition match within <filled> element.
<subdialog>	Jumps to a URL for specific application processing.
<transfer>	Transfers the caller to another phone number.

Menu elements

These elements define menus and menu choices. Menus can also contain event handlers.

Table 5.14 VoiceXML menu elements

Element	Description
<choice>	Defines one choice in a menu.
<menu>	Presents a list of choices to the user and transitions to the chosen information.

Event handlers

These elements define event handlers. These elements can occur within <field>, <menu>, <transfer>, and <subdialog> elements.

Table 5.15 VoiceXML event handlers elements

Element	Description
<catch>	Catches events thrown by a VoiceXML application or the Tellme Platform.
<help>	Handles the case of a user saying "help."
<noinput>	Handles cases where the user does not speak.
<nomatch>	Handles user input that is not recognized as part of the active grammar(s).
<prompt>	Plays audio to the user and listens for a response.

Directives

Directives perform a specified action or manipulate a variable.

Table 5.16 VoiceXML directive elements

Element	Description
<assign>	Assigns a value to a variable.
<audio>	Plays an audio file or converts text to speech within a prompt.
<clear>	Clears a variable.
<disconnect>	Terminates the user's phone call.
<goto>	Jumps to the specified URL.
<if><elseif/><else/>	Performs conditional logic.
<reprompt>	Replays the previously played prompt.
<return>	Returns control to the last document added to the <subdialog> stack.
<script>	Includes a block of JavaScript code.
<submit>	Obtains a new document via an HTTP GET or POST request.
<throw>	Generates a system- or user-defined event to catch with an event handler.

Extensible HyperText Markup Language (XHTML)

XHTML is a reformulation of HTML as an XML application. Although XHTML is almost identical to HTML 4.0, it has more rules and is cleaner than HTML because it conforms with XML. XHTML 1.0 specifies three XML document types: Strict, Transitional, and Frameset. This section focuses on XHTML Basic, a version of XHTML for small mobile devices. For more information, see http://www.w3.org/TR/xhtml-basic.

For a comprehensive XHTML Basic reference, see Appendix F, “XHTML Basic reference,” on page 195 (online only).

Differences between XHTML and HTML

Although XHTML is similar to HTML, there are some key differences between the two. XHTML requires:

- Elements and attributes must be in lowercase.
- All elements must be nested properly and closed.
- Attribute values must be quoted and minimization is forbidden.
- The id attribute must replace the name attribute.
- The script element must have a type definition.
- Documents must conform to XML rules.
- Documents must have some mandatory elements

In contrast, HTML is more liberal in its requirements, and web browsers tend to be very forgiving. The following sections briefly explain each of these differences.

Lowercase elements and attributes

XHTML is case-sensitive for all elements and attributes. The XHTML DTD defines all former HTML elements using lowercase, thus `<p>` and `<P>` are two different elements; `<body>` is recognized as the HTML body element.

For example, the following statement is incorrectly formatted because the elements are uppercase:

```
<P><B>This</B> is a paragraph</P>
```

This is correct:

```
<p><b>This</b> is a paragraph</p>
```

Proper nesting

Elements must be nested correctly: Elements should be closed in the order that you opened them; that is, an inner element should be closed before an outer element. This is not a new restriction to HTML; it is just more strictly enforced in XHTML.

For example, the following statement is not well-formed because the italics element should be closed off before specifying the bold element.

```
<b>This statement is incorrectly formatted <i> because of </b> incorrect nesting </i>
```

This is correctly formatted:

```
<b> This statement is <i> correctly </i> formatted. </b>
```

Element closure

All XHTML elements must be closed: non-empty elements must have an end element; empty elements must either have an end element or the start element must end with />.

For example, the following statements are not well-formed; they should have end elements:

```
<p>This is a paragraph
<p>This is another paragraph
This is a break<br>
```

These are correct:

```
<p>This is a paragraph</p>
<p>This is another paragraph</p>
This is a break<br></br>
```

Attributes

Attribute values must be quoted. For example, the following statement is incorrect because the value is not quoted:

```
<table rows=3>
```

This is correct:

```
<table rows="3">
```

You must specify attribute values explicitly.

Some HTML attributes have no values; their presence within an element causes a different behavior. Usually these types of attributes are Boolean. In XHTML, every attribute must have a value.

The following statements are incorrect because they use their minimized form; the values are implicit in HTML:

```
<dl compact>
<input checked>
<option selected>
```

These are correct:

```
<dl compact="compact">
<input checked="checked">
<option selected="selected">
```

id attribute

The id attribute replaces the name attribute: HTML 4.0 defined the name attribute for the following elements <a>, <applet>, <frame>, <iframe>, <img>, and <map>. In XHTML, the name attribute is deprecated; use the id attribute instead.

The following statement is incorrect because it uses the deprecated name attribute:

```
<img src="picture.gif" name="picture1" />
```

This is correct:

```
<img src="picture.gif" id="picture1" />
```

Script element

In XHTML, special characters such as less than (<) and ampersand (&) are treated as a part of the markup. To avoid this inside scripts or other elements, wrap the content with a CDATA definition like this:

```
<script>
  <![CDATA[
  ... script content ...
  ]]>
</script>
```

Mandatory elements

XHTML documents have some mandatory elements:

- All XHTML elements must be nested within the <html> root element. All other elements can have sub-elements. Sub-elements must be in pairs and correctly nested within their parent element. The basic document structure is:

```
<html>
<head> ... </head>
<body> ... </body>
</html>
```

- All XHTML documents must have a DOCTYPE declaration. The <html>, <head>, and <body> elements must be present. The title must be present in the head element. The DOCTYPE declaration is not part of the XHTML document itself. It is not an XHTML element, so it should not have a closing element.

  This is a minimum XHTML document template:

```
<!DOCTYPE html>
<html>
<head>
<title>This is a Title</title>
</head>
<body>
.
.
Body text goes here
.
.
</body>
</html>
```

XHTML Basic elements

This section provides a summary of all the elements in XHTML Basic, grouped functionally.

Structural and general elements

The following are structural and general elements:

Table 5.17 XHTML structural and general elements

Element	Description
body	Defines the body element.
head	Specifies information about the document.
html	Defines an HTML document.
title	Specifies the document title.
meta	Provides meta-information about a page, such as descriptions and keywords for search engines.

Text-formatting elements

The following are text-formatting elements:

Table 5.18 XHTML text formatting elements

Element	Description
abbr	Defines an abbreviation.
acronym	Defines an acronym.
address	Defines an address element.
blockquote	Defines a long quotation.
br	Inserts a single line break.
cite	Defines a citation.
code	Defines computer code text.
dfn	Defines a definition term.
div	Defines a section in a document.
em	Defines emphasized text.
h1-h6	Defines header 1 to header 6.
kbd	Defines keyboard text.
p	Defines a paragraph.
pre	Defines preformatted text.
q	Defines a short quotation.
samp	Defines sample computer code.
span	Defines a section in a document.
strong	Defines strong text.
var	Defines a variable.

Hypertext and related elements

The following are hypertext and related elements:

Table 5.19 XHTML hypertext and related elements

Element	Description
a	Defines an anchor.
base	Defines a default reference to external resources.

List elements

The following are list elements:

Table 5.20 XHTML list elements

Element	Description
dl	Defines a definition description.
dt	Defines a definition list.
dd	Defines a definition term.
li	Defines a list item.

Table 5.20 XHTML list elements (continued)

Element	Description
ol	Defines an ordered list.
ul	Defines an unordered list.

Form elements

The following are form elements:

Table 5.21 XHTML form elements

Element	Description
form	Defines a form.
input	Defines an input field.
label	Defines a label.
option	Defines an item in a list box.
select	Defines a selectable list.
textarea	Defines a text area.

Table elements

The following are table elements:

Table 5.22 XHTML table elements

Element	Description
caption	Defines a table caption.
table	Defines a table.
td	Defines a table cell.
th	Defines a fixed table header.
tr	Defines a table row.

Image and other elements

The following are image and other elements:

Table 5.23 XHTML image and other elements

Element	Description
img	Defines an image.
link	Defines a resource reference.
object	Defines an embedded object.
param	Defines a parameter for an object.

Chapter 6

Using wireless development tools

This chapter describes the wireless development tools included with Lutris Enhydra 3.5, and explains how to use them in application development.

Enhydra tools

Lutris Enhydra 3.5 includes tools that you can use to develop any applications (wireless or Web), plus some special tools especially for wireless application development. The general tools are:

- JBuilder and Kelp
- Forte for Java Community Edition
- Enhydra XMLC
- DODS

JBuilder and Kelp

JBuilder is a visual development tool for building pure Java applications. JBuilder includes a configurable code editor, a Java compiler with a dependency-checking system, and a debugging environment. Written in Java itself, JBuilder Foundation is a platform-independent integrated development environment (IDE).

The following figure shows a sample JBuilder project:

Kelp

Kelp is a set of tools that extend a Java IDE to accelerate Enhydra application development. Kelp is integrated with both Borland JBuilder and Oracle JDeveloper™ development environments.

For detailed information on Kelp, see the *Developer's Guide*.

Kelp includes:

- Enhydra Application wizard, which generates the basic framework of an Enhydra application. You can also run this wizard independently of an IDE, to create a new application.
- XMLC Compiler wizard, which lets you set options for the XML compiler (XMLC), select HTML files, and call XMLC from within the IDE
- Enhydra Deployment wizard, which sets up your project to launch your application with Enhydra Multiserver and updates configuration files to match your current environment
- (JBuilder only) Enhydra XMLC property pages, which give you control over how XMLC builds Document Object Model (DOM) classes from your HTML files
- (JBuilder only) Enhydra template property pages, which facilitate editing application configuration files

- Build integration, which lets you invoke XMLC and the Enhydra configuration processes whenever you rebuild your project
- Kelp sample projects, which demonstrate techniques for creating dynamic Web pages with XMLC

Enhydra application wizard

The Enhydra application wizard enables you to quickly start development of an Enhydra application. It will generate code and make files for both an Enhydra super-servlet (PO-based) application and for a servlet application. It will generate presentation layer code and markup language templates in:

- HTML
- WML
- cHTML
- XHTML

Figure 6.1 Dialog box in the Enhydra application wizard

Forte for Java Community Edition

Forte™ for Java Community Edition is an extensible, cross-platform IDE for the Java™ 2 platform. The Forte for Java product family offers a complete set of development and integration tools for creating entry- to enterprise-level applications. Forte for Java Community Edition provides a fully modular environment that delivers integrated visual design, editing, compilation, and debugging capabilities to create applets and Java applications.

Figure 6.2 Forte for Java Community Edition IDE:

XMLC

Enhydra XMLC is a compiler for documents written in eXtensible Markup Language (XML) based languages (such as WML, XHTML, cHTML, and voiceXML), or Hypertext Markup Language (HTML).

For detailed information on XMLC, see the *Developer's Guide*.

What XMLC does

XMLC takes a document as input and creates a Java class that represents that document. A Java application can use this class to modify the document programmatically, then pass the resulting dynamic content to a Web server, save it to the file system, or use it in some other way. From a programming perspective, the HTML or XML-based pages become resources. Your code can manipulate the document content without affecting its layout.

One way to think of XMLC is as a *template engine* for HTML and XML-based documents because it creates an API from a template document. Unlike other template engines, XMLC is highly extensible. In addition to compiling existing document types, such as Wireless Markup Language (WML), you can extend it to compile other XML document types (for example, MathML).

How to use XMLC

The most common use of XMLC is for developing a presentation layer for an Enhydra application; however, you can also use XMLC without Enhydra. For example, you can use XMLC to create a standalone application that generates a semi-dynamic website, such as a catalog that is updated whenever you run the application.

DODS

The Data Object Design Studio (DODS) allows you to design the data-layer classes of an application that uses the Enhydra application framework. DODS then generates the Java source code for the data-layer classes.

DODS consists of three parts: an object-oriented GUI design tool that reads and writes DOML files, a set of code generators that reads DOML files to generate data-access code, and a set of underlying Enhydra classes to facilitate data access in Enhydra applications.

For detailed information on DODS, see the *Developer's Guide*.

Wireless tools and device emulators

Lutris Enhydra 3.5 includes some special wireless tools and device emulators to help you develop and test wireless applications. They include:

- YoSpace SmartPhone emulator
- Motorola iDEN SDK
- Nokia Toolkit
- Pixo microbrowser
- WAPtor WML editor

Note These tools were not developed by Lutris Technologies, and are provided for developers' convenience; they are not supported by Lutris. For detailed information, on each tool, see the product CD for corresponding documentation provided by the vendor.

YoSpace SmartPhone emulator

The YoSpace SmartPhone emulator lets you test your wireless WML applications on a development system. You can enter a URL that contains the WML, and the emulator will browse to that location and show you what the page looks like in a wireless device, such as the Nokia 7110 shown in Figure 6.3.

Figure 6.3 Yospace Nokia 7110 emulator

The YoSpace SmartPhone emulator simulates a wireless phone. It lets you see how a WML page will look on a variety of wireless devices. By default, the device is the Nokia 7110 (head zoom). To change the device, select Options | Browser and choose a different device.

Note The release of SmartPhone emulator included in Lutris Enhydra is an early access release that has a 10-minute time-out period. After 10 minutes of use, the program automatically exits.

Figure 6.4 Nokia 7110 alphanumeric entry screen

You can use the Navi Roller to scroll through the letters: scroll up and down the list by moving the roller up and down, and select a letter by pressing the Navi Roller. Use the right softkey to delete a letter, and the left softkey to select OK and enter the word. When viewing this screen, you can also type letters using your computer keyboard.

Nokia WAP toolkit

The Nokia WAP toolkit is a Windows tool for creating, testing, and demonstrating WAP applications. The Nokia WAP toolkit includes tools for creating WML and WMLScript content, debugging WAP applications, and simulating the WAP content on WAP-enabled devices. You can write, test, debug, and run applications on PC-based simulations of both the Nokia 7100 series phone as well as a Nokia concept phone prototype.

Figure 6.5 Nokia toolkit

WAP server simulator

The Nokia WAP toolkit includes a WAP Server Simulator, which is a pre-configured, limited version of Nokia's WAP Server. The WAP Server Simulator permits only local client connections (connections from the same machine on which the Toolkit was installed) and so cannot be used as a fully-operational WAP Gateway.

Note The WAP Server Simulator supports only WAP 1.1, not WAP 1.2. Servlets cannot (currently) be added to the Server Simulator, and that only HTTP and local file access is supported.

For more information on the Nokia WAP Server Simulator, see the Nokia toolkit documentation.

Motorola iDEN SDK

The iDEN™ Software Development Kit (SDK) for Windows contains a set of 32-bit DLL's for Windows® 95, Windows® 98, and Windows® NT 4.0 environments. The iDEN Foundation Library is one of the provided DLLs that provides an application programming interface (API) for applications to access features of data-capable iDEN Subscriber Units through a serial cable connection.

Applications connect to the phone through one of several connection modes. They also access phone functions through provided function calls and receive notification of phone activity through Windows events provided by the iDEN Foundation Library.

The Motorola iDEN SDK also provides an iDEN phone emulator, shown in Figure 6.6.

Figure 6.6 Motorola iDEN phone emulator

Pixo Internet microbrowser

Lutris Enhydra includes a Windows-based emulator of the Pixo™ Internet microbrowser. The Pixo Internet microbrowser:

- Supports Internet standards. The Pixo Internet microbrowser is based on the same standards used on the Internet today, including:
  - HTML 4.0 tags and attributes appropriate for small screen devices
  - cHTML
  - HTTP 1.0
  - Cookies
- Provides an intuitive user experience, including:
  - User-defined bookmarks
  - Forward and Back navigation keys
  - Selectable hyperlinks
  - Page cache for offline viewing
- Supports both color and grayscale phone displays.
- Provides secure transactions with integrated SSL support.

Figure 6.7 Pixo Internet microbrowser emulator

WAPtor

EasyPad WAPtor, developed by WAPtop Software, is a simple tool for editing Wireless Markup Language (WML) pages. The following figure shows a sample WML page in the WAPtor graphical user interface:

Chapter 7

Enhydra programming techniques

This chapter describes Enhydra programming techniques for wireless applications. It assumes you are already familiar with the basics of Enhydra programming.

For a general introduction to Enhydra programming, see *Getting Started with Lutris Enhydra*; for more in-depth information, see the *Developer's Guide*.

Using wireless DOMs

In addition to HTML, Enhydra XMLC includes DOMs for all the major wireless document types:

- WML
- cHTML
- XHTML
- voiceXML

Specifying DOMs

When you compile a document with XMLC, you must specify the appropriate DOM for the document using the `-domfactory` command-line option:

```
xmlc -domfactory domFactoryClass
```

where *`domFactoryClass`* is the fully-qualified name of the DOM factory class to use. For example, to compile a WML document, use the command:

```
xmlc -domfactory org.enhydra.wireless.wml.WMLDomFactory
```

Like any XMLC command-line option, you can specify it in an XMLC options file or an XMLC metadata file. Table 7.1 summarizes the DOM factory classes for wireless presentation languages.

Table 7.1 DOM Factory Classes

Presentation language	DOM Factory class
WML	org.enhydra.wireless.wml.WMLDomFactory
cHTML	org.enhydra.wireless.chtml.CHTMLDomFactory
XHTML	org.enhydra.xml.xhtml.XHTMLDomFactory
voiceXML	org.enhydra.wireless.voicexml.VoiceXMLDomFactory

Manipulating DOMs

Since many wireless presentation languages do not support the SPAN tag, you must use the ID attribute of a supported tag instead. For example:

```
<P ID="firstPara">Some text to replace at runtime</P>
```

XMLC automatically creates a `setText` method for every tag that has a direct text child, so this tag will have a method, `setTextFirstPara()`, that you can use to set its text.

In general, XMLC generates a `setTextXXX()` method for a tag if the tag meets all these conditions:

- It has an ID attribute whose value is a legal Java identifier.
- It can legally have a text child.
- It actually has a text child in the document template.

The last condition means that you must provide "dummy text" to replace—you can't just use an empty tag.

Multi-profile presentation logic

You can use the same presentation logic to manipulate content in multiple profiles (for example, WML and cHTML), simply by using the same ID attribute for the corresponding content in the markup templates. The AirSent application does this: it uses a single PO to manipulate all the different document types.

Using writeDOM()

The best way to generate content with presentation objects is to use the method `writeDOM()`, in the `HttpPresentationResponse` interface. This method will work for all document types that Enhydra supports: HTML, WML, cHTML, VoiceXML, and XHTML. It will automatically format the document according to its document type and set the MIME type of the HTTP response.

In general, you call `writeDOM()` with an `XMLObject` argument:

```
comms.response.writeDOM(options, returndoc);
```

where `comms` is an `HttpPresentationComms` object.

Note Since most clients do not yet support MIME type "text/xhtml", you may want to manually set the MIME type to "text/html" for XHTML documents. Fortunately, `writeDOM()` has a signature that takes an `OutputOptions` object in addition to an `XMLObject`:

```
org.enhydra.xml.io.OutputOptions options = new org.enhydra.xml.io.OutputOptions();
options.setMimeType("text/html");
comms.response.writeDOM(options, returnHTML);
```

Using the HTML DOM for XHTML documents

Because XHTML and HTML are structurally similar, it may be convenient to use the same presentation logic to manipulate equivalent XHTML and HTML documents. However, XHTML document objects are subclasses of their HTML equivalents. If you use the `org.enhydra.xml.xhtml.HTMLDomFactory` instead of the default `XHTMLDomFactory`

will cause XMLC to generate `getElementXXX()` methods that return `org.w3c.dom.html` objects instead of `org.enhydra.xml.xhtml` objects. These objects will be compatible with existing HTML presentation code. To do this, use the following make file directives:

```
XMLC_XHTML_OPTS = -keep -domfactory org.enhydra.xml.xhtml.HTMLDomFactory
XMLC_XHTML_OPTS_FILE = options.xmlc
```

For an example of this technique, see the DiscRack sample application.

Presentation techniques

In creating wireless applications, you can apply many techniques from standard Web application development. This section describes some techniques that are particularly useful for wireless applications.

Session maintenance

By default, Enhydra stores a user's session identifier in a browser cookie. However, since many microbrowsers do not support cookies, a wireless application generally needs to use URL-encoding for session maintenance instead of cookies. To do this, add the following setting to the application configuration file:

```
SessionManager.SessionEncodeUrlState = auto
```

The `writeDOM()` method of the `HttpPresentationResponse` interface outputs a document object (DOM) formatted according to its type and sets the MIME type of the response automatically. It also automatically encodes session information into the response URL.

In general, for wireless applications, you should use `writeDOM()` to transparently URL-encode session information.

Detecting device types

You will often need to determine the nature of the device that is requesting a PO or servlet, so that the application can return content appropriate for that device. For example, WML would be sent to a WAP phone, HTML to a Web browser, and so on.

In a PO, you can use the `getHeader()` method of the `HttpPresentationComms` object to get header information from the HTTP request. Pass this method a string that is the name of the header item you are interested in. For example, the "Accept" header indicates the type of content that the client will accept. You can then use string functions on the value returned by `getHeader()` to determine the type of content to return to the client.

For example, the following code searches for the strings "text/xml" and "text/vnd.wap.wml" in the "Accept" header, and then redirects to the appropriate page:

```
protected void rerouteForContent(HttpPresentationComms comms) {
   try {
      String header = comms.request.getHeader("Accept");
```

```
        if (header.indexOf("text/xml") != -1) {
           throw new ServerPageRedirectException(AirSentConstants.XML_PAGE);
        } else if(header.indexOf("text/vnd.wap.wml") != -1) {
           throw new ServerPageRedirectException(AirSentConstants.WAP_PAGE);
        }
     }
}
```

In some cases, the "Accept" header may not be sufficient to determine the type of content to return. You may want to check for a specific device, if for example it has some particular display characteristic that your application can target. You can test the "User-Agent" header string to determine the exact nature of the client:

```
String userAgent = comms.request.getHeader("User-Agent");
System.out.println( userAgent );
```

This code would print out the following to the console, if the client were a standard Web browser:

```
Mozilla/4.7 [en] (WinNT; I)
```

In a standard servlet, the `HttpServletRequest` object has a similar method, also called `getHeader()`, that you can use to get the HTTP request header information. Use the techniques illustrated previously to determine the type of content expected by the requesting device and the user agent.

Disabling caching

By default, most microbrowsers automatically cache all pages to conserve bandwidth. However, you usually don't want dynamic pages generated by an application to be cached, so you should disable client caching.

Because different microbrowsers handle caching in different ways, there are two techniques available for disabling caching: the document technique and the programmatic technique. To ensure that the application works on the widest range of microbrowsers, use both techniques.

Note For some microbrowsers (such as the Sprint NeoPoint), neither of these techniques works, so you must use a completely different URL for each page.

Document technique

You can disable caching with a META tag in a document's `HEAD`:

```
<meta forua="true" http-equiv="Cache-Control" content="max-age=0"/>
```

The `forua` attribute is a valid only in WML, and specifies that the `meta` tag will be sent to the client, and not removed at an intermediate stage.

For HTML and XHTML, use

```
<meta http-equiv="Cache-Control" content="max-age=0"/>
```

Programmatic technique

The other technique is to specify no caching in the HTTP header. To do this, add the following line in each PO:

```
comms.response.setHeader("Cache-Control", "no-cache");
```

where `comms` is the `HttpPresentationComms` object for the PO.

If you are using servlets, use the following statement:

```
response.addHeader("Cache-Control", "no-cache");
```

where `response` is an object that implements `HttpServletResponse`.

Character encoding for i-mode

Since i-mode currently works only with Japanese devices and systems, you must set the character encoding to Japanese (shift-JS). When using XMLC, you must use the Swing parser, which can handle shift-JS encoding.

To specify the Swing parser and shift-JS encoding, use these XMLC command-line parameters:

```
-parser swing  -html:encoding Shift_JIS
```

In addition, you must set the content type in your servlets or POs as follows:

```
response.setContentType("text/html; charset=x-sjis");
response.setContentType("text/html; Shift_JIS");
```

Setting MIME types

The Multipurpose Internet Mail Extension (MIME) type specifies a files's content type. Every file that a Web server sends to a browser has a MIME type in the header that tells the browser how to handle the file. So, to configure your server to serve wireless files correctly, you need to specify the correct MIME type. The specific procedure for adding a MIME type depends on the Web server you are using—see you server documentation.

Table 7.2 MIME types for wireless files

File type	Extension	MIME type
WML	.wml	text/vnd.wap.wml
Compiled WML	.wmlc	application/vnd.wap.wmlc
WMLScript	.wmls	text/vnd.wap.wmlscript
Compiled WMLScript	.wmlsc	application/vnd.wap.wmlscriptc
Wireless bitmap	.wbmp	image/vnd.wap.wbmp
Compact HTML	.chtml, .html	text/chtml
XHTML	.xhtml	text/xhtml
XML	.xml	text/xml
Java Application Descriptor (MIDP application)	.jad	text/vnd.sun.j2me.app-descriptor

If you are using POs, the `response.writeDOM()` method automatically sets the correct MIME type. If you are using servlets, set the content type with:

```
response.setContentType("text/vnd.wap.wml");
```

where `response` implements the `ServletResponse` interface.

Note Most clients do not yet support MIME type “text/xhtml”, so you may have to use “text/html” instead for XHTML documents.

J2ME/MIDP programming

Although creating a client application for a Java device (for example, a Java phone) is not Enhydra programming, it is nevertheless part of the overall application development. The Sun Java site has a good introduction to programming with MIDP:

http://developer.java.sun.com/developer/technicalArticles/wireless/midpgetstart

Using XML with J2ME

The AirSent application illustrates a useful technique: using a custom XML document type (with a specialized DTD) as the data interchange format between the Enhydra application and the J2ME client. The J2ME client application uses the KXML parser to parse the XML and then uses this information to generate a UI on the Java device. For more information, see “J2ME profile” on page 114.

For more information on KXML, see http://kxml.enhydra.org.

Chapter 8

Understanding the AirSent sample application

This chapter describes the AirSent sample application, an Enhydra application designed to illustrate many of the basic principles of Enhydra wireless application development.

Note In this chapter, the root AirSent directory is referred to as `<airsent_root>`. This is equivalent to `<enhydra_root>/examples/AirSent`.

For instructions on how to build and run AirSent, see "Getting started with wireless application development" on page 33.

The scenario

AirSent is a bicycle document and small package delivery service in Fog City. Even in the Internet age, some things need to be physically delivered: legal documents, timely gifts, medicine, and so on. Traffic and parking in Fog City are so bad that bicycles are usually quicker than cars.

AirSent wants customers to be able to enter orders and check on their status, and for the central office to dispatch messengers quickly and reliably. They also want to be able to get up-to-the-minute status on deliveries.

Fortunately, they've equipped all their couriers with Web phones, so the company can use the Web phones to dispatch couriers. However, some couriers have WAP phones and others have Java-enabled phones—but they all need to be able to use the application.

The AirSent application addresses all these requirements, and is extensible in case AirSent decides to expand to Tokyo, London, or other cities.

User roles

There are three user roles in the AirSent application:

- Customers, who place orders for deliveries using a standard PC Web browser
- Dispatchers (administrators), who assign couriers to each order
- Couriers (messengers), who actually make the deliveries

Dispatchers work at the central office, and use a PC with a standard Web browser to enter new delivery orders, monitor the status of deliveries, and perform other administrative tasks.

Couriers are out on the street riding bicycles—and dodging cars and buses—as they pick up and deliver packages. After picking up a package, a courier accesses the AirSent application on his or her cellphone, and indicates the package has been picked up. Likewise, after delivering a package, the courier enters this information in the application via cellphone. This way, AirSent has up-to-the-minute information on the status of deliveries.

The application

AirSent is a good candidate for a wireless application because

- Couriers are inherently mobile users
- Package delivery is time-sensitive by nature
- Couriers' wireless transactions are short

AirSent illustrates how to partition work between the wired Web and the wireless Web. All the extensive text entry occurs in the Web portion of the application: Web phone users (couriers) are not expected to enter text information. All the courier has to do is press one button to indicate he or she has picked up or delivered a package.

Figure 8.1 AirSent application flow

AirSent's application flow is illustrated in Figure 8.1, "AirSent application flow." The flow generally goes like this:

1. A customer enters an order for a delivery, using a PC Web browser.
2. The AirSent administrator, working from a PC Web browser, sees the order appear on the AirSent Administrator page, and assigns the order to a courier.
3. Using a cellphone, the courier accesses the AirSent Web site and logs in using his or her badge number. The AirSent application displays all the pickups and deliveries currently assigned to the courier.
4. The courier acknowledges that he or she has picked up the package by clicking a single link in the wireless app accessed by cellphone.
5. After delivering the package, the courier returns to the wireless app to acknowledge doing so.

   Throughout the process, the administrator and customers can check on the status of deliveries using their Web browsers.

Application layers

Like any good application, AirSent contains three primary parts or layers:

- Data layer, which communicates with the database
- Business layer, which contains business rules and other application logic
- Presentation layer, which controls how the application appears on the end-user device

Additionally, the presentation layer contains profiles (sometimes called "skins") for different mobile devices. For more information on AirSent's profiles, see "Presentation layer" on page 110.

Storyboard

One of the key initial steps in creating and understanding an application is making the storyboard. As described in *Getting Started with Lutris Enhydra*, the storyboard provides a graphical overview of an application's flow. A storyboard is sometimes also referred to as a *sitemap*.

As illustrated in Figure 8.2, the AirSent storyboard consists of two major branches:

- Customer
- Administrator/dispatcher

The wireless page for couriers consists of a single page, so it is not shown on the sitemap.

Figure 8.2 AirSent storyboard

Data layer

The data layer contains all the code that interacts with the database.

Database schema

The AirSent database has the following tables:

- Messenger, containing information on couriers and dispatchers (dispatcher information could be in a separate table, but for simplicity it is combined with courier information).
- Customer, containing information on customers; each customer is uniquely identified by an ID number.
- Delivery, containing all the information about one delivery order, including the customerID for the customer who ordered it, and the messengerID of the messenger assigned to it.
- Person, containing detailed information on any people in the database, including couriers (messengers) and customers.
- Address, containing information on a delivery address.

Figure 8.3 illustrates the database schema for the AirSent application.

Figure 8.3 AirSent database schema

Data objects

The data layer is defined by the DOML file named `genDods` in `<airsent_root>/po/src/com/lutris/airsent/dods/AirSent.doml`.

Using this DOML file, DODS generates the data layer, containing two packages: delivery and person. The delivery package contains a single DO, DeliveryDO. The person package contains three DOs: CustomerDO, MessengerDO, and PersonDO.

Note For ease of use, the DO java files are included in the AirSent application, even though they can be regenerated by DODS from the DOML file.

Each DO contains data members for the columns in the corresponding table. For example, MessengerDO contains a `badge` data member and a `getBadge()` method.

DODS creates three other Java files for each data object, in addition to the actual DO class (for example for a DO named foo):

- Data object interface, that the DO implements (for example, named fooDOI)
- Data object data structure (named fooDataStruct),a container for data members of the DO class.
- Query class (named fooQuery), used to query the database and return fooDO objects.

Business layer

The business layer contains classes that "wrap" data objects, plus classes for sending email from the application.

Each set of wrapper classes is contained in two packages:

- One package that contains an interface for the business object and a factory interface for creating instances of the object.
- One subpackage that contains implementations of these interfaces.

For example, the `messenger` package contains the `Messenger` interface and the `MessengerFactory` interface; the `messenger.impl` package contains the `MessengerImpl` and `MessengerFactoryImpl` classes that implement the two interfaces.

Sending email

The `business.email` and `business.email.impl` packages contain classes for sending email. When a courier delivers a package, the application sends email to the customer stating that the package has been delivered and citing the address to which it was delivered.

The `presentation.messenger.Display` class has methods that are called when a courier indicates that he or she has delivered a package. The `handleComplete()` method calls the `send()` method of the `EmailFactoryImpl` class to send email to the customer.

Presentation layer

The presentation layer consists of an HTML-based portion for the dispatchers and customers using PCs, and a wireless portion for couriers with mobile devices. The wireless portion generates output in

- WML, for WAP phones; this is the *WAP profile*.
- cHTML, for i-mode phones; this is the *i-mode profile*.
- AirSentXML, a dialect of XML, for JavaPhones; this is the *J2ME profile*. This profile includes client-side Java code using the MIDP libraries. For more information, see "AirSent J2ME client" on page 115.
- XHTML, for future XHTML-capable devices; this is the *XHTML profile*. Because this is an emerging technology, and no emulators strictly support it, this chapter does not focus on the XHTML profile.

Utility classes

The presentation layer uses several utility classes:

- `AirSentConstants`, which contains all the static constants used by the application, primarily strings corresponding to URLs and form field names (request variables).

- DeviceUtils, which has device detection and related methods.
- AirSentSessionData, a simple class for containing a user's session information.
- AirSentPresentationException, a class derived from com.lutris.appserver.server.httpPresentation.HttpPresentationException.

Base classes

The presentation layer has two abstract classes that are the superclasses of all the presentation objects (POs):

- BasePO, the parent class for all POs, including DeviceBasePO
- DeviceBasePO, the parent class for wireless POs

Both implement the com.lutris.appserver.server.httpPresentation interface.

Some of BasePO's key methods are:

- initSessionData(), to initialize the user' session data, if it is not already initialized
- checkAuthLevel(), to check the session data to see if the user has the authorization to access the page
- handleEvent(), to perform event handling (see "Event handling" on page 112).

Document object instantiation

DeviceBasePO defines a create() method that returns an XMLObject corresponding to the document object:

```
public XMLObject create(String poName) throws AirSentPresentationException {
   XMLObject page = null;
   try {
      page =
         getComms().xmlcFactory.create(DeviceUtils.getPageName(getComms(), poName));
   } catch (Exception ex) {
      throw new AirSentPresentationException("Trouble getting authorization level", ex);
   }
   return page;
}
```

The create() method calls DeviceUtils.getPageName(), which returns a string corresponding to the name of the XMLC-generated document class, based on the "Accept" HTTP request header. For example, a request for MessengerLogin.po from a WAP phone would cause getPageName() to return the string "com.lutris.airsent.presentation.messenger.MessengerLoginWML," the name of the DOM class corresponding to the WML page MessengerLogin.wml.

This technique works because of the convention of having XMLC create a class named xxxWML for the document xxx.wml, xxxHTML for document xxx.html, and so on. This convention is enforced by the HTML_CLASSES, WML_CLASSES, and XHTML_CLASSES (and so on) directives in the make file of the messenger presentation directory.

The POs call the `create()` method to instantiate document objects for the application to manipulate. For example, the `MessengerMain` PO calls the method as follows:

```
MessengerMainPage page =
   (MessengerMainPage)create("com.lutris.enhydra.examples.airSent.presentation.
   messenger.MessengerMain");
```

Once a PO has instantiated the document object, it uses common methods to manipulate it, regardless of whether it's a WML page, cHTML page, or AirSentXML page.

Event handling

In AirSent, an "event" is an action the user takes that is passed to the application through a request variable. AirSent performs event handling the same way the DiscRack sample application does. For a detailed description of this event handling implementation, see *Getting Started with Lutris Enhydra*.

Briefly, the `handleEvent()` method gets the string event name from the HTTP request, and passes it on to `getPage()`, or calls `handleDefault()` if there is no event parameter in the request. Then, `getPage()` uses Java reflection to convert the event to a method call of the form `handle`*`EventName`*, for example `handleLogout()`.

Detecting device types

AirSent determines what kind of content to provide (wireless or HTML) by inspecting the HTTP request header. The `BasePO.run()` method calls `DeviceUtils.rerouteforContent()`. This method examines the "Accept" string in the HTTP request header to determine what kind of content a wireless device can display, and redirects the client to the appropriate page. In this case, all the wireless devices are redirected to the same PO, because the same PO manipulates content for all the profiles:

```
public static void rerouteForContent(HttpPresentationComms comms)
   throws AirSentPresentationException {
   String header = null;
   String userAgent = null;

   try {
      userAgent = comms.request.getHeader("User-Agent");
      if ((header = comms.request.getHeader(AirSentConstants.ACCEPT)) == null) {
         // Just display HTML
         return;
      } else if (header.indexOf(AirSentConstants.XML_CONTENT) != -1) {
         throw new ClientPageRedirectException(AirSentConstants.XML_PAGE);
      } else if (header.indexOf(AirSentConstants.WAP_CONTENT) != -1) {
         throw new ClientPageRedirectException(AirSentConstants.WAP_PAGE);
      } else if (header.indexOf(AirSentConstants.XHTML_CONTENT) != -1) {
      } else if (header.indexOf(AirSentConstants.CHTML_CONTENT) != -1) {
      } else if(userAgent != null && userAgent.indexOf("Pixo") != -1) {
         throw new ClientPageRedirectException(AirSentConstants.CHTML_PAGE);
      }

   } catch (Exception e) {
       throw new AirSentPresentationException("Trouble rerouting header:" + header, e);
   }
}
```

The method has a special case for the Pixo microbrowser, which can display cHTML content, but does not indicate so in its request header.

Displaying dynamic content

Every PO has a `showPage()` method that returns an `XMLObject` representing the dynamically-modified document object. This method instantiates a document object using the `create()` method, previously described in "Document object instantiation" on page 111.

For example, the PO `messenger.Details` creates a document object like this:

```
DetailsPage page =
   (DetailsPage) create("com.lutris.airsent.presentation.messenger.Details");
```

Notice that `create()` returns a generic `XMLObject`, but the POs cast the returned object to the appropriate interface, in this case `DetailsPage`.

Once a PO has created the appropriate XMLObject, it can then use the standard `getTextXXX()` and `setTextXXX()` methods created by XMLC to retrieve andmodify content in the document. For example, the Details PO uses sets the name address, and directions for an pickup order like this:

```
page.setTextName(name);
page.setTextAddress(address);
page.setTextDirections(directions);
```

For more information on how XMLC works, see the *Developer's Guide*.

WAP profile

The WAP profile generates WML. The AirSent WML templates are in:

`<airsent_root>/po/src/com/lutris/airsent/resources/messenger/wml`

i-mode profile

The i-mode profile generates cHTML. The AirSent cHTML templates are in:

`<airsent_root>/spo/src/com/lutris/airsent/resources/messenger/chtml`

Because Enhydra has a unified DOM for HTML, cHTML, and XHTML, you can use the same Java code to manipulate content for devices using any of these languages.

Authentication

The `MessengerLogin` class performs authentication for messengers in the `handleLogin()` method. The crux of the method is:

```
if ((messenger = getApplication().getHomeFactory().getMessengerFactory().
      validatePassword(badge, password)) == null) {
   return showPage("Invalid badge!!");
} else {
   getSessionData().setMessenger(messenger);
   getSessionData().setUserAuth(AirSentConstants.MESSENGER_USER);
   System.out.println("In MessengerLogin....redirecting");
   throw new ClientPageRedirectException(AirSentConstants.MESSENGER_MAIN_PAGE);
}
```

This code uses the `validatePassword()` method in `MessengerFactoryImpl`, which queries the database to find a messenger with the specified badge number, and checks if the entered password matches the password in the database; if so, `validatePassword()` returns a `Messenger` BO. If no messenger with that badge number exists in the database, or if the password does not match, `validatePassword()` returns null.

J2ME profile

For JavaPhones, such as the Motorola iDEN™ phone, AirSent uses a dialect (or "application") of XML called *AirSentXML* as a data-interchange format; the J2ME client application parses the AirSentXML and generates the end-user display directly. Thus, the J2ME profile includes:

- server code that processes requests from the Java phone, and generates XML.
- client J2ME code that runs on the Java phone (or emulator).

AirSentXML

AirSentXML is a custom dialect (technically, an application) of XML, created especially for AirSent. Like any XML dialect, it is defined by a document type definition (DTD). The AirSentXML DTD is in:

`<airsent_root>/po/src/com/lutris/examples/airsent/resources/messenger/xml/dtd/AirSent.dtd`.

The AirSent DTD defines twelve elements and their attributes: MessengerLogin, MessengerMain, Details, Display, DisplayTemplateRow, DisplayTemplateName, DisplayTemplateComplete, ErrorText, NameField, Name, Address, and Directions.

```
<!--The MessengerLogin element represents the login for the messenger -->
<!ELEMENT MessengerLogin ( ErrorText)>

<!--The MessengerMain element is the element that represents a 'home' for the messenger. -->
<!ELEMENT MessengerMain (ErrorText)>

<!--The details element contains the details of a delivery. -->
<!ELEMENT  Details       ( ErrorText,  Name,  Address,  Directions)>

<!--The display element contains a link to details or a link to a delivery action. -->
<!ELEMENT  Display       ( ErrorText, NameField, DisplayTemplateRow)>

<!--The DisplayTemplateRow element decribes a row in the Display type -->
<!ELEMENT  DisplayTemplateRow  ( DisplayTemplateName,  DisplayTemplateComplete)>
<!ATTLIST  DisplayTemplateRow  id ID #IMPLIED>

<!--The DisplayTemplateName decribes a link to delivery details in the Display element. -->
<!ELEMENT  DisplayTemplateName  (#PCDATA)>
<!ATTLIST  DisplayTemplateName
           id ID #IMPLIED
           href CDATA #IMPLIED>

<!--The  DisplayTemplateComplete describes an action to perform a delivery or pickup.-->
<!ELEMENT  DisplayTemplateComplete  (#PCDATA)>
<!ATTLIST  DisplayTemplateComplete
           id ID #IMPLIED
           href CDATA #IMPLIED>
```

```
<!--The element for errors -->
<!ELEMENT  ErrorText         (#PCDATA)>
<!ATTLIST  ErrorText         id ID #IMPLIED
           jsessionid CDATA #IMPLIED>

<!--The element for a delivery name field -->
<!ELEMENT  NameField         (#PCDATA)>
<!ATTLIST  NameField          id ID #IMPLIED>

<!--The The element for a delivery name -->
<!ELEMENT  Name         (#PCDATA)>
<!ATTLIST  Name          id ID #IMPLIED>

<!--The element for an address -->
<!ELEMENT  Address      (#PCDATA)>
<!ATTLIST  Address       id ID #IMPLIED>

<!--The element for directions -->
<!ELEMENT  Directions   (#PCDATA)>
<!ATTLIST  Directions    id ID #IMPLIED>
```

The following XML document uses the AirSent DTD. It uses two elements: MessengerMain and ErrorText, which have been defined in the AirSent DTD.

```
<?xml version="1.0"?>
<!DOCTYPE MessengerMain SYSTEM "dtd/AirSent.dtd">
<MessengerMain>
<ErrorText id="ErrorText">Error messages go here</ErrorText>
</MessengerMain>
```

AirSent J2ME client

The AirSent application includes a Java™2 Micro Edition (J2ME) client application that is intended to run on the Motorola Java Phone. For development and demonstration purposes, it runs on the Motorola iDEN emulator for Windows.

The client application code is located in the directory `<airsent_root>/client/kvm/src`. A JBuilder4 project file, `airsent-jclient.jpx`, is also provided.

Packages

The application root package is `com.lutris.airsent`, and the package structure is:

- `app`: global application classes for initialization and flow control, utility classes, and an exception subclass.
  - `app.form.messenger`: UI components that are subclasses of `javax.microedition.lcdui.Form`
- `net`: contains the `Network` class that establishes HTTP connections and receives data from the server application.
- `test`: classes for testing
- `xml`: contains the `XMLParser` class that parses XML; it is a subclass of `de.kxml.parser.DefaultParser`

The AirSent client XML Parser is based on an open-source XML parser called kXML, created specifically for the Java KVM. For more information on kXML, see http://kxml.enhydra.org.

AirSent class

`AirSent`, a subclass of `javax.microedition.midlet.MIDlet`, is the application's central class. Running the application calls `AirSent.initApp()`, which:

- instantiates the `AppServices` object, that provides access to network and XML parsing functionality. The constructor for `AppServices` calls `getLoginPage()`, which attempts to make the initial network connection, creates the login page and sets the session ID given by Enhydra.
- instantiates the `AirSentController` object, that performs event handling.
- creates the application's display, a `javax.microedition.lcdui.Display` object.
- pre-builds the login screen.

AppServices class

The `AppServices` class performs some of the fundamentals application tasks. One of the key ones is retrieving a document from the server application, performed by the `getDocument()` method.

This method calls the `Network.getDocumentFromURL()` method, that opens an HTTP connection to thte specified URL, sets the HTTP request header information, and opens an `InputStream` from the HTTP response. Then, it uses the `XMLParser` on the resulting `InputStream`:

```
InputStream is = hc.openInputStream();
XMLParser parser = new XMLParser(new InputStreamReader(is));
doc = new Document();
doc.parse(parser);
```

Finally, after closing the network connection and input stream, `getDocumentFromURL()` returns a `de.kxml.kdom.Document`, which is passed on to `AppServices.getDocument()`.

Various methods, such as `getLoginPage()` and `handleLogin()`, in `AppServices` call `getDocument()` to get information from the server application.

AppController

The `AppController` class implements the interface `javax.microedition.lcdui.CommandListener`. The `commandAction()` method in this class responds to the following command events from the screens:

- nextCommand
- prevCommand
- loginCommand
- selectCommand
- detailsCommand
- deliverCommand
- configCommand
- logoutCommand

Appendix A

Using XSLT with Enhydra

Extensible Stylesheet Language Transformations (XSLT) is a W3C standard language for transforming XML documents into other document types. Lutris Enhydra 3.5 includes an implementation of XSLT based on *Xalan-Java* from the Apache XML project.

Overview

XSLT can transform an XML page into different markup languages (such as HTML and WML) through the application of an XSL stylesheet. Thus, you can use XSLT to target a single XML document to multiple devices that require different document types.

While XSLT can be useful, it is important to understand its limitations and why DOM manipulation through XMLC is usually preferable. Although transforming document types technically enables multiple devices to render the content, it cannot compensate for the navigational differences inherent in radically different devices. For this reason, a blended approach using both XMLC and XSLT manipulation is usually preferable.

When to use XSLT

Two cases in which XSLT can be useful in conjunction with XMLC are:

- If the target devices have generally the same display and UI capabilities, but require different document formats.
- When the dynamic content in an application is generated in an custom XML format that needs to be transformed to a variety of document formats.

When using XMLC, you should generally create a storyboard that maps the user interaction for each targeted device. This allows you to account for navigational differences and display optimization, and is essential for applications that will be run on radically different devices—such as PCs with full-featured Web browsers and WAP cellphones with three-line screens. When targeting devices that differ widely, you have to write presentation logic to create the dynamic content for each device. However, forward-thinking design and proper object inheritance can minimize code duplication.

For example, suppose you need to create an application that targets three wireless devices that all have similar four- to six-line textual displays, but that display xHTML, WML, and cHTML. You could build the storyboard in xHTML and use an XSLT stylesheet to transform the output into WML or cHTML for the other two devices. In general, it is better to have distinct storyboards for different devices, with some duplication of presentation logic, than to have a single "one size fits all" storyboard for all devices. It all depends on the relative complexity of the application and the number of supported devices.

Another sensible use of XSLT is when the dynamic content in an application has been generated in a custom XML format, perhaps from a database query or from a business-to-business (B2B) exchange. Rather than duplicating the presentation logic to convert this and populate an XMLC-generated DOM, it may be easier to write a set of XSLT stylesheets to perform the transformation. In this scenario, it is unlikely that the whole page will be generated with the XSLT transform, because of the need to wrap the content with different navigational models. Instead, a node within the XMLC page is populated with the output of the XSLT transform, and the rest of the page is generated from the XMLC template.

References

For further information on XSLT, see these websites:

W3C standard recommendation for XSLT: http://www.w3.org/TR/xslt

Apache Xalan project: http://xml.apache.org/xalan/index.html

Article on XMLC and XSL from the Lutris Enhydra Journal: http://www.lutris.com/journal/August2000/articles/XMLCandXSL.html

Example: Using XSLT with XMLC

Enhydra includes a simple example of how to use XSLT with XMLC in a presentation object (PO). The sample application transforms a simple XML document into an HTML table, using an XSLT stylesheet.

Building and running the application

The code for this sample application is installed in the directory:

```
<enhydra-root>/examples/XSLTdemo
```

where `<enhydra-root>` is your Enhydra root directory.

To build the example, edit `config.mk` in this directory and set ENHYDRA_DIR to the directory containing your Enhydra installation. Also, make sure that the `start` script in the `xalanDoc` subdirectory contains the line

```
<enhydra-root>/bin/multiserver ./multiserver.conf
```

Then, with `XSLTdemo` as the active directory, enter the command

```
make
```

in a (Cygwin) shell window. This builds the application.

To run the application, change the active directory to output, and enter

```
./start
```

Then, enter this URL in your Web browser:

http://localhost:9000

Click the hyperlink "Contact list generated from XML." You will see the result of the XSL transformation in an HTML table display:

Figure 8.4 Output of XSLT example application

Document type definition

The DTD `contacts.dtd` describes a simple format for contact information:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!ELEMENT ContactList (Contact?) >
<!ELEMENT Contact (FirstName, LastName, Street1, Street2, City, State, Zip, Phone,
Email) >
<!ELEMENT FirstName (#PCDATA) >
<!ELEMENT LastName (#PCDATA) >
<!ELEMENT Street1 (#PCDATA) >
<!ELEMENT Street2 (#PCDATA) >
<!ELEMENT City (#PCDATA) >
<!ELEMENT State (#PCDATA) >
<!ELEMENT Zip (#PCDATA) >
<!ELEMENT Phone (#PCDATA) >
<!ELEMENT Email (#PCDATA) >
```

This DTD defines a `ContactList` element that contains multiple `Contact` elements, each of which contains the data for a contact, including name, address, phone number, and so on.

XML document

Here is a sample XML document, `contacts.xml`, using the previous DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ContactList SYSTEM "contacts.dtd">
<ContactList>
    <Contact>
        <FirstName>Lars</FirstName>
        <LastName>Ulrich</LastName>
        <Street1>1200 Pacific Ave.</Street1>
        <Street2>Suite 300</Street2>
        <City>Santa Cruz</City>
        <State>CA</State>
        <Zip>95060</Zip>
        <Phone>(831)460-7345</Phone>
        <Email>lars@lutris.com</Email>
    </Contact>
    <Contact>
        <FirstName>Ricki</FirstName>
        <LastName>Martin</LastName>
        <Street1>1200 Sepulveda Blvd.</Street1>
        <Street2>Suite 666</Street2>
        <City>Los Angeles</City>
        <State>CA</State>
        <Zip>90432</Zip>
        <Phone>(213)666-7345</Phone>
        <Email>ricki@martin.com</Email>
    </Contact>
</ContactList>
```

XSLT stylesheet

Here is a sample XSLT stylesheet, `contact-table.xsl`, which transforms the XML document into an HTML table.

```
<?xml version="1.0"?>
<!DOCTYPE xsl:stylesheet [
<!ENTITY nbsp " ">
 ]>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSLTransform" version="1.0">
<xsl:output method="html" indent="yes" />
<xsl:template match="ContactList">
    <table>
    <tr><th>Name</th><th>Address</th><th>Phone</th><th>Email</th></tr>
    <xsl:apply-templates />
    </table>
</xsl:template>
<xsl:template match="Contact">
    <tr>
        <td>
            <xsl:value-of select="FirstName"/> 
            <xsl:value-of select="LastName"/>
        </td>
        <td>
            <xsl:value-of select="Street1"/> 
            <xsl:value-of select="Street2"/> 
            <xsl:value-of select="City"/>, 
            <xsl:value-of select="State"/>  
            <xsl:value-of select="Zip"/>
        </td>
```

```
        <td>
            <xsl:value-of select="Phone"/>
        </td>
        <td>
            <xsl:value-of select="Email"/>
        </td>
    </tr>
</xsl:template>
</xsl:stylesheet>
```

There are two templates in this stylesheet. The first template matches "ContactList," the root element of the XML document, so it only generates one match per document. This handles the beginning and ending tags of the table. The second template matches "Contact." This generates a table row with all the values for the Contact's information. This is done using the `<xsl:value-of/>` tag, which gets the text of the selected element from the children of the current node.

Note that the entity `nbsp` is declared explicitly, since this isn't declared in the XSLT DTD. You can also use

```
<xsl:text> </xsl:text>
```

to escape whitespace.

HTML page

The HTML page `ContactList.html` is compiled by XMLC and used in the PO. It contains the HTML template used by the application:

```
<html>
<head>
    <title>
        Contact List
    </title>
</head>
<body>
<div id="contactList">
    Contact List for <span id="username">xxx</span>
    <table class="discard">
        <tr>
            <th>Name</th>
            <th>Address</th>
            <th>Phone</th>
            <th>Email</th>
        </tr>
        <tr>
            <td>Joe Shmoe</td>
            <td>123 Blah St.</td>
            <td>222-222-2222</td>
            <td>shmoe@joe.com</td>
        </tr>
    </table>
</div>
</body>
</html>
```

When you compile this with XMLC, using the `-delete-class discard` option, it deletes the dummy contact table, making room for the XSLT-generated table.

Presentation object

The PO `ContactList.java` does the XSLT transformation.

```
import org.apache.xalan.xslt.XSLTProcessor;
import org.apache.xalan.xslt.XSLTInputSource;
import org.apache.xalan.xslt.XSLTResultTarget;
import org.apache.xalan.xslt.XSLTProcessorFactory;
import org.apache.xalan.xslt.StylesheetRoot;
import org.apache.xalan.xpath.xdom.XercesLiaison;
import org.w3c.dom.*;
import org.w3c.dom.html.*;
import com.lutris.appserver.server.httpPresentation.*;

public class ContactList implements HttpPresentation {
   public void run(HttpPresentationComms comms)
        throws HttpPresentationException { // Instantiate the page
      ContactListHTML page = (ContactListHTML)
         comms.xmlcFactory.create(ContactListHTML.class);
        // Get the username from session
        String username = comms.session.getUser().getName();
        // Set to anonymous if no user
        if (username == null) {
            username = "Enhydra User";
        }
        // set the name of the username, using an XMLC convenience method
        page.setTextUsername(username);
        try {
            // Create an XSLT processor; pass the factory a XercesLiaison, since we
            // are outputting to DOM
            XSLTProcessor processor = XSLTProcessorFactory.getProcessor(new
XercesLiaison());
            // Create input source document.
            XSLTInputSource xmlID = new XSLTInputSource("contacts.xml");
            // Process/compile the stylesheet
            StylesheetRoot stylesheet =
processor.processStylesheet("contact-table.xsl");
            // Process the source tree and produce the result tree.
            // We just pass it an element of the XMLC document as the result
            // target; all the output nodes just get appended
            stylesheet.process(processor, xmlID,
                new XSLTResultTarget(page.getElementContactList()));
        } catch (Exception e) {
            e.printStackTrace();
            writeError("Couldn't process XSL", comms);
            return;
        }
        // Write the page
        comms.response.writeDOM(page);
    }
    private void writeError(String msg, HttpPresentationComms comms)
        throws HttpPresentationException {
        comms.response.setContentType("text/plain");
        comms.response.write(msg);
    }
}
```

First, this PO instantiates the XMLC-compiled class and sets some values using the generated convenience methods. Next comes the fun part: using Xalan-Java to do XSL transformation.

To perform the XSL transformation, the PO first instantiates an `XSLTProcessor` using a `XercesLiaison`. Normally, Xalan uses its own DTM (document table model) parser, but to output to a DOM, you have to use a `XercesLiaison`. Next, the PO creates a new `XSLTInputSource` from the `contacts.xml` file, passing the file name to the constructor. Then, it compiles the stylesheet into a `StylesheetRoot` object using the processor's `processStylesheet` method. Finally, the `StylesheetRoot` actually processes the source document with the method call:

```
stylesheet.process(processor, xmlID, new
      XSLTResultTarget(page.getElementContactList()));
```

This call inserts the results into a node (`page.getElementContactList()`) in the document. By creating a new `XSLTResultTarget` object, the transformed DOM is simply appended to the children of the output node.

Finally, once the XSLT transformation is complete, `comms.response.writeDOM()` sends the response to the client.

Appendix B

Using the DiscRack wireless profiles

The DiscRack sample application includes presentation templates and code for wireless access; these are referred to as wireless profiles. Specifically, it includes a profile for

- WML, for access by WAP devices
- XHTML, for access by XHTML-capable devices.

The business and data layers are the same as the "standard" HTML-based DiscRack application.

Building and running wireless DiscRack

To run the DiscRack wireless profiles, you first need to get DiscRack running with the desired database, as described in *Getting Started with Lutris Enhydra*. To use InstantDB, follow the procedure outlined in Chapter 7, "Using InstantDB," of the *Developer's Guide*. Confirm that the Web application is running by accessing it in your Web browser.

After you have done this, build DiscRack by entering **make** from the `DiscRack/discRack` directory. Then enter this command from the `DiscRack/output` directory:

```
./start
```

To access the wireless DiscRack application with a wireless device emulator, access the application by choosing Options | Open Location and entering the URL:

```
http://localhost:5555/wap/personMgmt/Login.po
```

The DiskRack login page appears, as shown in Figure 8.5:

Figure 8.5 Wireless DiscRack login on YoSpace SmartPhone Emulator

Using the YoSpace SmartPhone Emulator

After starting DiscRack, as described in *Getting Started with Lutris Enhydra*, start the YoSpace SmartPhone Emulator, then choose Options | Open Location. Enter the URL http://localhost:9000. This will display the DiscRack login page. From the DiscRack login page, click the Navi Roller to select "Login." The alphanumeric entry screen appears, as shown in Figure 8.6:

Figure 8.6 Nokia 7110 alphanumeric entry screen

You can use the Navi Roller to scroll through the letters: scroll up and down the list by moving the roller up and down, and select a letter by depressing the Navi Roller. Use the right button to delete a letter, and the left button to select OK and enter the word. When viewing this screen, you can also type letters using your computer keyboard.

In general, wireless applications are not suitable for lengthy character data entry. You should use the Web-based DiscRack to enter data (users and discs) into the DiscRack database and then access the wireless application to view your discs.

Tip To reduce typing, create a simple user name and password, such as "`a`" or "`b`."

Troubleshooting

When using the Nokia WAP Toolkit with DiscRack, turn off FastEncoding (on the Preferences menu). Otherwise, you may get errors such as "Exception in Encoding."

The Sprint PCS NeoPoint 1000 phone is currently one of the most popular WAP-capable phones. DiscRack includes some special code for the NeoPoint 1000 that is commented out because NeoPoint lacks comprehensive WML support. To access DiscRack from a NeoPoint 1000, you must un-comment this code and rebuild the application.

Overview of the WAP profile

The DiscRack WAP profile is in the `presentation.wap` package. All wireless presentation objects are subclasses of `WapBasePO`, which is itself a subclass of `BasePO`. The presentation objects are:

- `personMgmt.Login`
- `personMgmt.Register`
- `discMgmt.DiscCatalog`
- `discMgmt.Edit`
- `discMgmt.Help`

These objects perform functions analogous to the standard presentation objects of the same names, except they work with the corresponding WML pages.

Appendix C

WML reference

This appendix provides reference information for WML 1.1 elements and attributes.

Document structure

In general, a WML document is structured like this:

```
<wml>
   <head>
      <meta/>
   </head>
   <card>
      tags
      ...
   </card>
   ...
   <card>
      tags
      ...
   </card>
</wml>
```

where *tags* can be any of:

```
<onevent>
<timer>
<do>
<a>
<fieldset>
<img>
<input>
<select>
<p>
```

Element reference

This section provides reference information for each tag (element) in WML, listing the syntax and attributes, and providing a simple example.

For each element, the attributes' types are listed. Table 8.1 explains these types.

Table 8.1 Attribute types

Attribute type	Explanation
string	Any string
URL	A Uniform Resource Locator, either absolute (like http://www.lutris.com/products/index.html or relative (like xyz/foo.html)
Boolean	Either "true" or "false:
MIME Type	Any valid MIME type (such as "text/html")
name	Any valid name
number	A non-negative integer

Every WML tag has the attributes listed in Table 8.2 For brevity, these attributes are omitted in this section, unless they have special meaning for the tag.

Table 8.2 WML common attributes

Attributes	Description
class	The class of the element.
id	String that identifies the object.
xml:lang	Human language for the element's contents with ISO 639 standard two-character name.

<a>

A short form for the `<anchor>` element. Developers are encouraged to use this tag instead of `anchor` where possible.

Syntax

```
<a
   href="url"
   title="label">
      content
</a>
```

where *content* can be any valid combination of

- *text* to be displayed as a hyperlink.
- `<br/>` elements: See "<br>" on page 132
- `<img>` elements: See "<img>" on page 136

Attributes

The `<a>` tag has the following attributes:

Name	Value	Required?	Description
href	URL	Required	Defines the location the browser loads when the link is selected.
title	string	Optional	Text identifier for the element.
accesskey	string	Optional	Assigns an access key to an element.

Example

```
<a href="there">follow me</a>
```

<access>

Specifies access control for a WML deck. If a deck does not include an access element, access control is disabled and cards in any deck can access this deck.

Syntax

```
<access
   domain="domain"
   path="path"/>
```

Attributes

The `<access>` tag has the following attributes:

Name	Value	Required?	Description
domain	string	Optional	Specifies the domain from which other decks may access.
path	string	Optional	Specifies the path from which other deck can access.

Example

```
<access domain="lutris.com" path="/lutris"/>
```

<anchor>

Defines a hyperlink, bound to the go, prev, or refresh task. When a user selects the link and presses ACCEPT, the device executes the task.

Syntax

```
<anchor
   title="label">
      <task>text
   </anchor>
```

where `<task>` is one of `<go>`, `<prev>`, or `<refresh>`, and represents the action to perform when the user activates the link, and *text* is the text the device will display to represent the link.

Attributes

The `<anchor>` tag has the following attribute:

Name	Value	Required?	Description
title	string	Optional	A label that identifies the link. Default title is “Link.” Devices use this attribute in a variety of ways. For example, they may use it to display a tool tip or issue a voice prompt when the user selects the link. To ensure compatibility on a wide range of devices, label should be a maximum of five characters.
accesskey	string	Optional	Assigns an access key to an element.

Example

```
<anchor>follow me
<go href="there">
</anchor>
```

<b>

Renders text with a bold font.

Syntax

```
<b>text</b>
```

where *text* is the text to display in bold font.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
This is <b>bold</b> text.
```

<big>

Renders text with a large font.

Syntax

```
<big>text</big>
```

where *text* is the text to display in large font.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
This is <big>huge</big> text!
```


Inserts a new line.

Syntax

```
<br/>
```

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
This is the end of page one.<br/>
```

<card>

Defines a single card in a deck. A card can contain text, markup, links, input-fields, tasks, images and more. The `id` attribute of the card element can be used as an anchor

Syntax

```
<card id="name"
   title="label"
   newcontext="boolean"
   ordered="true"
   onenterforward="url"
   onenterbackward="url"
   ontimer="url">
      content
</card>
```

where *content* represents the WML card definition and consists of one or more of the following elements:

- <onevent>: See "<onevent>" on page 139.
- <timer>: See "<timer>" on page 145.
- <do>: See "<do>" on page 133.
- <a>: See "<a>" on page 130.
- <fieldset>: See "<fieldset>" on page 134.
- <img>: See "<img>" on page 136.
- <input>: See "<input>" on page 137.
- <select>: See "<select>" on page 142.
- <p>: See "<p>" on page 140.

Attributes

The <card> tag has the following attributes:

Name	Value	Required?	Description
id	string	Required	Specifies a name for the card. The name acts as a fragment anchor for navigating to that card. For example, you can specify <go href="#cardname"/> to navigate to the card (see "<go>" on page 135).
onenterbackward	URL	Optional	This event occurs when the card is entered from a <prev> task.
onenterforward	URL	Optional	This event occurs when the card is entered from a <go> task.
ontimer	URL	Optional	This event occurs when a <timer> expires.
title	string	Optional	Text identifier for the element.
newcontext	Boolean	Optional	Tells the browser to reinitialize upon entry to the card.
ordered	Boolean	Optional	Tells the user agent if the card content should be ordered in the display.

Example

```
<card>This is a new card</card>
```

<do>

Defines a card-level user interface element.

Syntax

```
<do
   type="type"
   label="label"
   name="name"
   optional="boolean">
      task
</do>
```

where *task* represents the action to perform when the user activates the specified interface mechanism:

- `<go>`: See "<go>" on page 135.
- `<prev>`: See "<prev>" on page 141.
- `<noop>`: See "<noop>" on page 138.
- `<refresh>`: See "<refresh>" on page 141.

You must bind one of these task elements to the user interface mechanism.

Attributes The `<do>` tag has the following attributes:

Name	Value	Required?	Description
type	string	Required	Tells the user agent the intended use of the element.
label	string	Optional	Provides a text label for the element.
name	name	Optional	Specifies the name of the event binding.
optional	Boolean	Optional	If optional is TRUE, the user agent may ignore this element.

Example

```
<do type="accept"
   label="login">
<go href="#login"/>
</do>
```


Renders text with emphasis.

Syntax

```
<em>text</em>
```

where *text* is the text to display in emphasized font.

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
<em>Ollie</em> the sea otter.
```

<fieldset>

Allows grouping of related fields and text.

Syntax

```
<fieldset
   title="label">
      content_and_text
</fieldset>
```

where *content_and_text* represents the items to group together and consists of one or more of these elements:

- `<fieldset>`: A nested `<fieldset>`. See "<fieldset>" on page 134.
- `<input>`: See "<input>" on page 137.
- `<select>`: See "<select>" on page 142.

- *text*: the device uses this text in different ways, depending on the elements you specify (for example, to prompt the user for input or describe various options).

Attributes The <fieldset> tag has the following attribute:

Name	Value	Required?	Description
title	string	Optional	Text identifier for the element.

Example

```
<fieldset title="Full Name">
First name:
<input type="text" name="fname"/>
Last name:
<input type="text" name="lname"/>
</fieldset>
```

<go>

Declares a <go> task, indicating navigation to a URI. If the URI refers to a WML card or deck, it is displayed.

Syntax

```
<go
   href="url"
   sendreferer="boolean"
   method="method"
   accept-charset="charset"
   <postfield name="data" value="value"/>
      content
</go>
```

where *content* represents the variables to set when opening the specified URL:

- <setvar>: See “<setvar>” on page 143.

Important Specifying content for the <go> element is optional. If you do *not* specify any content, you must use the syntax <go attributes/> rather than <go attributes> content</go>.

Attributes The <go> tag has the following attributes:

Name	Value	Required?	Description
href	URL	Required	Defines the location the browser loads when the link is selected.
senreferer	Boolean	Optional	If TRUE, the user agent must specify the URI of the deck containing the task.
method	(post|get)	Optional	Specifies the HTTP submission method.
enctype	MIME Type	Optional	Specifies the content type to be used to submit the parameter to the server when the method is post. The default value for this attribute is application/x-www-form-urlencoded.

Name	Value	Required?	Description
cache-control	no-cache	Optional	If present and set to `no-cache`, the client MUST reload the URL from the origin server.
accept-charset	string	Optional	Specifies the list of character encodings for data that the origin server accepts when processing input.

Example

```
<go href="#card1"/>
```

<head>

Contains information relating to the deck as a whole, including metadata and access control elements.

Syntax

```
<head>
   content
</head>
```

where *content* represents deck-level header information, and can include the following elements:

- `<access>`: One only. See "<access>" on page 131.
- `<meta>`: One or more. See "<meta>" on page 138

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
<head><access domain="lutris.com"></head>
```

<i>

Renders text with an italic font.

Syntax

```
<i>text</i>
```

where *text* is the text to display in italic font.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
You <i>should</i> fill in all fields.
```


Indicates that an image is to be included in the text flow.

Syntax

```
<img
   alt="text"
   src="url"
   localsrc="icon"
```

```
   align="alignment"
   height="n"
   width="n"
   vspace="n"
   hspace="n"/>
```

Attributes The `<img>` tag has the following attributes:

Name	Value	Required?	Description
alt	string	Required	Alternative text for the image.
src	URL	Required	URI of the image file.
localsrc	string	Optional	An alternative internal representation of the image.
vspace	length	Optional	The amount of whitespace to be inserted above and below the image.
hspace	length	Optional	The amount of whitespace to be inserted to the left and right of the image.
align	(top|middle|bottom)	Optional	Image alignment within the text flow.
height	length	Optional	Height of the image, in pixels
width	length	Optional	Width of the image, in pixels

Example

```
<img src="pic.wbmp"/>
```

<input>

Specifies a text entry object. You can restrict the user's input by specifying the `format` and `emptyok` attributes.

Syntax

```
<input
   name="variable"
   title="label"
   type="type"
   value="value"
   format="specifier"
   emptyok="boolean"
   size="n"
   maxlength="n"
   tabindex="n"/>
```

Attributes The `<input>` tag has the following attributes:

Name	Value	Required?	Description
name	name	Required	Specifies the name of the event binding.
type	(text|password)	Optional	Specifies the type of text-input area.
value	string	Optional	Specifies the value of the input area.
format	string	Optional	Specifies the input mask for user input.
emptyok	Boolean	Optional	If TRUE, input is not required.

Name	Value	Required?	Description
size	number	Optional	Specifies the width in characters of the input area.
maxlength	number	Optional	Specifies the maximum number of characters for the input area.
tabindex	number	Optional	Specifies the number in the tab order of input areas.
title	string	Optional	Text identifier for the element.
accesskey	string	Optional	Assigns an access key to the element.

Example

```
Enter your name: <input type="text" name="uname"/>
```

```
<input name=”cname” “emptyok=”false” format=”M*M”/>
```

<meta>

Contains generic meta-information for the deck. A `<meta>` element must occur within the `<head>` elements. It may contain exactly one attribute specifying a property name.

Syntax

```
<meta
   http-equiv="Cache-Control"
   content="max-age=time"
   forua="true"/>
```

Attributes

The `<meta>` tag has the following attributes:

Name	Value	Required?	Description
http-equiv	string	Optional	Can be used in place of name. Indicates that the property should be interpreted as an HTTP header.
name	name	Optional	Specifies the name of the event binding.
forua	Boolean	Optional	If FALSE, an intermediate agent MUST remove the meta element before the document is sent to the client.
content	string	Required	Specifies the property value.
scheme	string	Optional	Specifies a form or structure that can be used to interpret the property value.

Example

```
<head>
<meta content="charset" name="character-set=ISO-10646-UCS-2"/>
</head>
```

<noop>

“No Operation” specifies that nothing should be done.

Syntax

```
<noop/>
```

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
<noop>Do nothing here</noop>
```

<onevent>

Binds a task to an event for the immediately enclosing element.

Syntax

```
<onevent
   type="type">
      task
</onevent>
```

where *task* represents the action to perform when the intrinsic event occurs, may be any of these elements:

- <go>: See "<go>" on page 135.
- <prev>: See "<prev>" on page 141.
- <noop>: See "<noop>" on page 138.
- <refresh>: See "<refresh>" on page 141.

Attributes The <onevent> tag has the following attribute:

Name	Value	Required?	Description
type	string	Required	Tells the user agent the intended use of the element.

Example

```
<onevent type="onenterbackward">
<go href="there"/>
</onevent>
```

<optgroup>

Groups option elements into a hierarchy.

Syntax

```
<optgroup
   title="label">
      content
</optgroup>
```

where *content* is one or more of the following:

- <optgroup>: A nested <optgroup>. See "<optgroup>" on page 139.
- <option>: See "<option>" on page 140.

Attributes The <optgroup> tag has the following attributes:

Name	Value	Required?	Description
title	string	Optional	Text identifier for the element.

Example

```
<optgroup>
<option title="option1"/>
<option title="option2"/>
</optgroup>
```

<option>

Defines an option input field in a select element.

Syntax

```
<option
   title="label"
   value="value"
   onpick="url">
      content
</option>
```

where *content* can be:

- An event that is triggered when a user selects the opition, with the `<onevent>` element: See “<onevent>” on page 139.
- Text to represent the selection item.

Attributes

The `<option>` tag has the following attributes:

Name	Value	Required?	Description
value	string	Optional	Specifies the value of the input area.
title	string	Optional	Text identifier for the element.
onpick	URL	Optional	Specifies the URI to go to when the user selects the option.

Example

```
<select name="select1">
<option name="option1"/>
<option name="option2/>
</select>
```

<p>

Defines a section of text as a paragraph.

Syntax

```
<p
   align="alignment"
   mode="wrapmode">
    content
</p>
```

Attributes

The `<p>` tag has the following attributes:

Name	Value	Required?	Description
align	(left|right|center)	Optional	Specifies the text alignment for the paragraph.

Name	Value	Required?	Description
mode	(wrap | nowrap)	Optional	Specifies the line-wrap mode for the paragraph.

Example

```
<p>This is a paragraph.</p>
```

<postfield>

Specifies a field name and value for transmission to an origin server during a URL request.

Syntax

```
<postfield
   name="name"
   value="value"/>
```

Attributes

The `<postfield>` tag has the following attributes:

Name	Value	Required?	Description
name	string	Required	Specifies the field name.
value	string	Required	Specifies the field value.

Example

```
<postfield name="postfield1" value="hello!"/>
```

<prev>

Declares a prev task, indicating navigation to the previous URI in the history stack.

Syntax

```
<prev> content</prev>
```

where *content* represents the variables to set when opening the previous URL, using the `<setvar>` element. See “<setvar>” on page 143.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
<anchor>
<prev/>Go Back
</anchor>
```

<refresh>

Initiates an update of the user agent as specified by the `<setvar>` elements.

Syntax

```
<refresh>content</refresh>
```

where *content* represents the variables to refresh. You must specify at least one variable in a <refresh> statement using the <var> element. See "<setvar>" on page 143.

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
<refresh>
<setvar name="var1" value="hello!"/>
</refresh>
```

<select>

Begins an option list control. Each option is specified by an option element. Option elements can be organized into hierarchical groups using the optgroup element.

Syntax

```
<select
   title="label"
   multiple="boolean"
   name="variable"
   value="default"
   iname="index_var"
   ivalue="default"
   tabindex="n">
      content
</select>
```

where *content* represents the list of items from which to choose. You can define selection content using one or more of the following elements:

- <optgroup>: See "<optgroup>" on page 139.
- <option>: See "<option>" on page 140.

Devices display these elements in the order in which you specify them

Attributes The <select> tag has the following attributes:

Name	Value	Required?	Description
title	string	Optional	Text identifier for the element.
name	name	Optional	Specifies the name of the event binding.
value	string	Optional	Specifies the field value.
iname	name	Optional	Specifies the name of the variable to be set with the index result of the selection.
ivalue	string	Optional	Specifies the default-selected <option> element.
multiple	Boolean	Optional	If TRUE, allows multiple <option> selections.
tabindex	number	Optional	Specifies the number in the tab order of input areas.

Example

```
<select name="select1">
<option name="option1"/>
<option name="option2/>
</select>
```

<setvar>

Specifies the variable to set in the current browser context after executing a task.

Syntax

```
<setvar
   name="name"
   value="value"/>
```

Attributes The `<setvar>` tag has the following attributes:

Name	Value	Required?	Description
name	string	Required	Specifies the field name.
value	string	Required	Specifies the field value.

Example

```
<setvar name="name1" value="value1"/>
```

<small>

Renders the text in a small font.

Syntax

```
<small>text</small>
```

where *text* is the text to display in small font.

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
This is <small>small</small> text.
```


Renders the text with strong emphasis.

Syntax

```
<strong>text</strong>
```

where *text* is the text to display in strongly emphasized font.

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
I <strong>really</strong> mean it!
```

<table>

Defines a table for formatting visible elements. Nesting of `table` elements is not allowed. Used along with the `tr` and `td` elements.

Syntax

```
<table
   title="name"
   align="left|right|center"
   columns="num_cols">
...
</table>
```

Attributes

The `<table>` tag has the following attributes:

Name	Value	Required?	Description
title	string	Optional	Text identifier for the element.
align	(C|L|R)	Optional	Specifies the alignment of the table.
columns	number	Required	Specifies the number of columns for the table.

Example

```
<table columns="4" align="C">
```

<td>

Defines a cell element of a table.

Syntax

```
<td>content</td>
```

where *content* is text inside the table cell, or the following elements.

- `<img>`: See "<img>" on page 136.
- `<anchor>`: See "<anchor>" on page 131.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
<table>
<tr>
<td>Cell 1</td>
</tr>
</table>
```

<template>

Declares a template for cards in the deck. Everything between the `<template></template>` tags is automatically used by every card in the deck.

Syntax

```
<template
   onenterforward="url"
   onenterbackward="url"
   ontimer="url">
      content
</template>
```

where *content* represents the general action to take when particular events occur:

- `<do>`: See "<do>" on page 133.
- `<onevent>`: See "<onevent>" on page 139.

Attributes The `<template>` tag has the following attributes:

Name	Value	Required?	Description
onenterforward	URL	Optional	This event occurs when the card is entered from a `<go>` task.
onenterbackward	URL	Optional	This event occurs when the card is entered from a `<prev>` task.
ontimer	URL	Optional	This event occurs when a `<timer>` expires.

Example

```
<template>
<do type="accept" label="Home">
<go href="#home"/>
</do>
</template>
```

<timer>

Declares a card timer, which starts at card entry and ends when the card is exited.

Syntax

```
<timer
   name="variable"
   value="value"/>
```

Attributes The `<timer>` tag has the following attributes:

Name	Value	Required?	Description
name	name	Optional	Specifies the name of the event binding.
value	string	Required	Specifies the field value.

Example

```
<timer name="timer1" value="60"/>
```

<tr>

Defines a row in a table element.

Syntax

```
<tr>
   <td>...</td>
   ...
</tr>
```

Attributes None, other than common attributes (see Table 8.2 on page 130).

Example

```
<table>
<tr>
<td>Row 1, Cell 1</td>
</tr>
</table>
```

<u>

Renders the text underlined.

Syntax

```
<u>text</u>
```

where *text* is the text to display in underlined font.

Attributes

None, other than common attributes (see Table 8.2 on page 130).

Example

```
This <u>text</u> is underlined.
```

<wml>

Defines a deck and encloses all information and cards in the deck.

Syntax

```
<wml
   xml:lang="lang">
...
</wml>
```

where *content* represents the WML elements that define the actions of the deck.

Attributes

The `<wml>` tag has the following attribute:

Name	Value	Required?	Description
xml:lang	name	Optional	Specifies the natural or formal language for the WML document. Overrides any other language specification for the document. See the XML specification at " at http://www.w3.org for more information about specifying a language value. .

Example

```
<wml xml:lang="en-us">
   <card>
   ...
   ...
   </card>
</wml>
```

Appendix D

cHTML reference

This appendix provides complete reference information for cHTML.

Element reference

This section provides reference information for each tag (element) in cHTML, listing its syntax and attributes, and providing a simple example.

<!-- --> (comment)

Used to annotate code in the document. Commented text is not displayed by the browser.

Syntax `<!-- text -->`

where *text* is the text of the comment.

Attributes None.

Example

```
<!-- This text is hidden from the browser -->
```

<!DOCTYPE>

Indicates which variation of HTML is used in the document.

Syntax `<!DOCTYPE HTML version>`

where *version* is the version of HTML used in the document.

Attributes None.

Example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD Compact HTML 1.0 Draft//EN">
```

&xx;

Escape sequence used to display special character entities in the browser.

Syntax &*xx*;

where *xx* is the ASCII value or entity name of the character entity.

Attributes None.

Example

```
&#151; <!-- displays an em dash -->
&amp; <!-- displays an ampersand -->
&gt; <!-- displays a greater-than sign -->
&lt; <!-- displays a less-than sign -->
&quot; <!-- displays a quotation mark -->
  <!-- creates a nonbreaking space -->
```

<A>

Anchor tag that defines both links and targets.

Syntax <A name="*string*"|href="*URL*">*text or image*</A>

where *string* is an arbitrary name for the anchor and *URL* is the Web address of the link.

Attributes The <A> tag has the following attributes:

Name	Required?	Description
href	Required[1]	Location the browser loads when the link is selected.
name	Required[1]	Anchor name for the element.

1. One or the other of href or name must be present, but not both.

Example <A href="http://www.lutris.com">Lutris Technologies, Inc.</A>

Note Either href *or* name is required as an attribute, not both.

<BASE>

Location against which all other URLs in the document are resolved. Must be placed in the Head section of the document.

Syntax <BASE href="*URL*">

where *URL* is the Web address of the base document.

Attributes The <BASE> tag has the following attribute:

Name	Required?	Description
href	Required	Location the browser loads when the link is selected.

Example

```
<BASE href="http://www.lutris.com">
```

<BODY>

Section of the document containing the content.

Syntax

```
<BODY>text and/or other elements</BODY>
```

Attributes None.

Example

```
<BODY>
   <P>This is a simple example.</P>
</BODY>
```

Note If your HTML code is being multi-purposed for use in other browsers and your <BODY> tag contains any color-related attributes, be aware that these are not recognized in cHTML and therefore that all non-white colors are drawn as black.

Forced line break that does not create a new paragraph.

Syntax

```
<BR [clear=all|left|right]>
```

Attributes The <BR> tag has the following attribute:

Name	Required?	Description
clear	Optional	In combination with left- or right-aligned image, cancels wraparound function for corresponding side(s) of text column.

Example

```
We can put a line break here.<BR>
```

<CENTER>

Horizontally centers HTML content; synonymous with <DIV align="center">.

Syntax

```
<CENTER>text or other elements</CENTER>
```

Attributes None.

Example

```
<CENTER>
   <P>Welcome to Enhydra!</P>
</CENTER>
```

<DD>

Definition description of a term in a definition list. (See also "<DL>" on page 151 and "<DT>" on page 151.)

Syntax

```
<DL>
   <DT>heading<DD>text
</DL>
```

where *heading* is the term being defined, and *text* is its definition.

Attributes

None.

Example

```
<DL>
   <DT>cHTML<DD>Compact HyperText Markup Language, a wireless Web protocol.
</DL>
```

Note

The end tag is optional, but may be useful to leave in when adapting for HTML or XML.

<DIR>

Creates an unordered list, possibly in a multi-column format, depending on the browser.

Syntax

```
<DIR [compact]>
   <LI>
   ...
   <LI>
</DIR>
```

Attributes

The <DIR> tag has the following attribute:

Name	Required?	Description
compact	Optional	Depending on the browser, reduces line spacing and sometimes indentation between list items.

Example

```
<DIR compact>
   <LI>Simon
   <LI>Theodore
   <LI>Alvin
</DIR>
```

<DIV>

Delineates one or more paragraphs for formatting.

Syntax

```
<DIV align=left|right|center>text or other elements</DIV>
```

Attributes

The <DIV> tag has the following attribute:

Name	Required?	Description
align	Required	Positions the content according to the attribute's value: • left • right • center

Example

```
<DIV align="center">
   <P>This is the first paragraph in the division, followed by an image.</P>
   <IMG src="image.gif">
   <P>This is the second paragraph in the division.</P>
</DIV>
```

<DL>

Definition list, in which the definition for each item is indented following its heading. (See also "<DT>" on page 151 and "<DD>" on page 150.)

Syntax

```
<DL>
   <DT>heading<DD>text
</DL>
```

where *heading* is the term being defined, and *text* is its definition.

Attributes

None.

Example

```
<DL>
   <DT>cHTML<DD>Compact Hypertext Markup Language, a wireless Web protocol.
</DL>
```

<DT>

Definition term in a definition list. (See also "<DD>" on page 150 and "<DL>" on page 151.)

Syntax

```
<DL>
   <DT>heading<DD>text
</DL>
```

where *heading* is the term being defined, and *text* is its definition.

Attributes None.

Example

```
<DL>
   <DT>cHTML<DD>Compact Hypertext Markup Language, a wireless Web protocol.
</DL>
```

<FORM>

Set of controls that enables user input.

Syntax

```
<FORM action="URL"
   method="[get|post]"
   enctype="[application/x-www-form-urlencoded|multipart/form-data|text/plain]">
controls and text</FORM>
```

Attributes The `<FORM>` tag has the following attributes:

Name	Required?	Description
`action`	Required	URL of the program receiving the data.
method	Required	Method by which the form sends data to the server for processing. Acceptable values are: • `get` • `post`
enctype	Optional	Data encoding formats to be used by the form. Acceptable values are: • `application/x-www-form-urlencoded` (the default) • `multipart/form-data` • `text/plain`

Example

```
<FORM action=”http://www.lutris.com/myapp/foobar.po”>
   <P>Yes <INPUT type=radio checked> No <INPUT type=radio>
</FORM>
```

<HEAD>

Section of an HTML document containing the document type declaration, title, and metadata.

Syntax

```
<HEAD>header information</HEAD>
```

Attributes None.

Example

```
<HEAD>
   <TITLE>The Greatest Web Page Ever</TITLE>
</HEAD>
```

<Hn>

Headings in six levels, enabling the author to structure the document.

Syntax

```
<Hn align=[left|center|right]>heading text</Hn>
```

where *n* is the number of the level, 1 through 6.

Note Current i-mode browsers do not distinguish between the various heading levels, so the value of n is irrelevant for them.

Attributes The `<Hn>` tag has the following attribute:

Name	Required?	Description
`align`	Optional	Positions the content according to the attribute's value: • `left` • `right` • `center`

Example

```
<H1 align="center">The Greatest Web Page Ever</H1>
```

<HR>

A horizontal rule used to visually divide a Web page into sections.

Syntax

```
<HR [align=left|right|center]
   [size=integer]
   [width=integer]
   [noshade]>
```

Attributes The `<HR>` tag has the following attributes:

Name	Required?	Description
`align`	Optional	Positions the rule according to the attribute's value: • `left` • `right` • `center`
size	Optional	Thickness of the rule in pixels.
width	Optional	Width of the rule as a percentage of the width of the browser window.
noshade	Optional	Renders a rule without the default shading.

Example

```
<HR size=2 width=75% align=center>
```

<HTML>

Indicates the beginning and end of the HTML coding in the document.

Syntax

```
<HTML [version="string"]>content of page</HTML>
```

Attributes

The <HTML> tag has the following attribute:

Name	Required?	Description
version	Optional	cHTML standard version used to compose the document.

Example

```
<HTML version="C-HTML 1.0">
...
</HTML>
```

<IMG>

Links to a graphic file anywhere on the Internet.

Syntax

```
<IMG src="string"
   [align="left|right"]
   [align="top|middle|bottom"]
   [width=number| percent]
   [height=number|percent]
   [hspace=number]
   [vspace=number]
   [alt="string"]
   [border=number] >
```

Attributes

The <IMG> tag has the following attributes:

Name	Required?	Description
src	Required	URL of image location.
align	Optional	Horizontally positions the content according to the attribute's value: • left • right
align	Optional	Vertically positions the content according to the attribute's value: • top • middle • bottom
width	Optional	Width of image in pixels or as a percent of browser window width.
height	Optional	Height of image in pixels or as a percent of browser window width.
hspace	Optional	Horizontal space surrounding the image, in pixels.
vspace	Optional	Vertical space surrounding the image, in pixels.

Name	Required?	Description
alt	Optional	Alternate text string to display for text-only browsers, or browsers with image loading turned off.
border	Optional	Width of border in pixels. Default is 0.

Example

```
<IMG src="red-flower.gif" height=100 width=100>
```

Note Large images are compressed automatically.

<INPUT>

Form control element that accepts input from the user.

Syntax

```
<INPUT type=text|password|checkbox|radio|hidden|submit|reset
   name="string"
   [value="string"]
   [size=integer]
   [maxlength=integer]
   [checked] >
```

Attributes The type attribute specifies the nature of the form element, and determines which other attributes are valid.

When the `type` attribute has value "text" or "password," `<INPUT>` has the following other attributes:

Name	Required?	Description
name	Required	Name of form field. Alphanumeric characters separated by underscores or hyphens are recommended.
size	Optional	Display width of the field, in number of characters.
maxlength	Optional	Maximum total length of the field, in number of characters.
value	Optional	Default value of the field.

When the `type` attribute has value "checkbox," `<INPUT>` has the following other attributes:

Name	Required?	Description
name	Required	Name of form field. Alphanumeric characters separated by underscores or hyphens are recommended.
value	Optional	Default value of the check box; text appears next to the control.
checked	Optional	Presence of this attribute indicates the check box is initially selected. Attribute does not take a value

When the `type` attribute has value "radio," `<INPUT>` has the following other attributes:

Name	Required?	Description
`name`	Required	Name of radio button group. Alphanumeric characters separated by underscores or hyphens are recommended. All elements with the same name will be in the same group. Only one element in a group can be selected.
value	Optional	Default value of the radio button; text appears next to the control.
checked	Optional	Indicator that the radio button is initialized to a selected state.

When the `type` attribute has value "hidden," `<INPUT>` has the following other attributes:

Name	Required?	Description
type	Required	Type of form control element, in this case `hidden`.
name	Required	Arbitrary name of control, for reference by scripts or programs. Alphanumeric characters separated by underscores or hyphens are recommended.
value	Optional	Default value of the hidden control.

When the `type` attribute has value "submit," `<INPUT>` has the following other attributes:

Name	Required?	Description
type	Required	Type of form control element, in this case `submit`.
name	Required	Arbitrary name of control, for reference by scripts or programs. Alphanumeric characters separated by underscores or hyphens are recommended.
value	Optional	Default value of the Submit button.

When the `type` attribute has value "reset," `<INPUT>` has the following other attributes:

Name	Required?	Description
type	Required	Type of form control element, in this case `reset`.
name	Required	Arbitrary name of control, for reference by scripts or programs. Alphanumeric characters separated by underscores or hyphens are recommended.
value	Optional	Default value of the Reset button.

Example

```
<P>Yes <INPUT type=radio checked> No <INPUT type=radio></P>
```

Note The maximum character buffer is 512 bytes.

<LI>

Item in an ordered (numbered) or unordered (bulleted) list.

Syntax

```
<LI>list item text</LI>.
```

Attributes

None.

Example

```
<UL>
   <LI>Ice cream
   <LI>Fudge sauce
   <LI>Chopped nuts
   <LI>Whipped cream
   <LI>Maraschino cherry
</UL>
```

Note

The end tag is optional.

<MENU>

Variety of the unordered list, originally designed for user menus.

Syntax

```
<MENU>
   <LI>
   ...
   <LI>
</MENU>
```

Attributes

None.

Example

```
<MENU>
   <LI>Larry
   <LI>Moe
   <LI>Curly
</MENU>
```

<META>

Provides metadata header information about the document, in user-defined `name/content` pairs.

Syntax

```
<META http-equiv="string"
   name="string"
   content="string">
```

Attributes The `<META>` tag has the following attributes:

Name	Required?	Description
`http-equiv`	Required[1]	Specifies the name of an item equivalent to data in the HTTP response header. The content attribute then specifies the value of the header item.
`name`	Required[1]	Arbitrary name for metadata content.
`content`	Required	Content for associated header, when http-equiv attribute is present.

1. One or the other of `http-equiv` or `name` must be present, but not both.

Example

```
<HEAD>
   <META http-equiv="Content-Type" content="text/html; charset=SHIFT_JIS">
</HEAD>
```

<OL>

An ordered (numbered) list.

Syntax

```
<OL>
   <LI>
   ...
   <LI>
</OL>
```

Attributes None.

Example

```
<OL>
   <LI>First, use Ctrl-N to open a new window.
   <LI>Type some text.
   <LI>Press Ctrl-S to save the file.
</OL>
```

<OPTION>

Item choice in a `<SELECT>` element.

Syntax

```
<SELECT>
<OPTION [selected]>display text</OPTION>s
</SELECT>
```

Attributes The `<OPTION>` tag has the following attribute:

Name	Required?	Description
`selected`	Optional	Designates the one option out of several that appears by default in the `<SELECT>` control.

Example

```
<SELECT name="city">
   <OPTION selected>Santa Cruz</OPTION>
   <OPTION>Scotts Valley</OPTION>
   <OPTION>Capitola</OPTION>
   <OPTION>Soquel</OPTION>
   <OPTION>Felton</OPTION>
</SELECT>
```

Note The end tag is optional.

<P>

Paragraph tag, including some linespace after the text.

Syntax

```
<P [align=left|right|center]>text and other elements</P>
```

Attributes The `<P>` tag has the following attribute:

Name	Required?	Description
`align`	Optional	Acceptable values are: • `left` • `center` • `right`

Example

```
<P>This is a paragraph, even though it contains only one sentence.</P>
```

Note The end tag is optional.

<PLAINTEXT>

Text rendered in fixed-width font with formatting tags ignored by the browser.

Syntax

```
<PLAINTEXT>text</PLAINTEXT>
```

Attributes None.

Example

```
<PLAINTEXT>Even if I <b>bold some text</b>, or make it <i>italic</i>, the PLAINTEXT
tag will turn off the formatting.</PLAINTEXT>
```

<PRE>

Preformatted text, including spaces, tabs, and line breaks.

Syntax

```
<PRE>formatted text</PRE>
```

Attributes None.

Example

```
<PRE>All of these
line breaks
will be preserved
even without
paragraph or line break tags.
Amazing!</PRE>
```

<SELECT>

In a form, a pull-down menu or list box from which the user selects an item. The `multiple` attribute allows the user to select more than one item.

Syntax

```
<SELECT name="string"
   [size=number]
   [multiple]>
   <OPTION>...</OPTION>
   ...
   <OPTION>...</OPTION>
</SELECT>
```

Attributes

The `<SELECT>` tag has the following attributes:

Name	Required?	Description
`name`	Required	Name of this instance of the control, for reference by scripts or programs.
`size`	Optional	Number of options visible to the user, that is, row height of the control.
`multiple`		Indicator that the user can select more than one option from the list.

Example

```
<SELECT name="city">
   <OPTION selected>Santa Cruz</OPTION>
   <OPTION>Scotts Valley</OPTION>
   <OPTION>Capitola</OPTION>
   <OPTION>Soquel</OPTION>
   <OPTION>Felton</OPTION>
</SELECT>
```

<TEXTAREA>

Form element that accepts memo-length user text input.

Syntax

```
<TEXTAREA name="string"
   [rows=integer]
   [cols=integer]>text or other elements</TEXTAREA>
```

Attributes The <TEXTAREA> tag has the following attributes:

Name	Required?	Description
name	Required	Name of this instance of the control, for reference by scripts or programs.
rows	Optional	Number of rows in height for the control.
cols	Optional	Number of columns (characters) of width for the control.

Example

```
<TEXTAREA name="notes" rows=20 cols=40>
```

Note The maximum character buffer is 512 bytes.

<TITLE>

Web page name that may be displayed by a browser.

Syntax

```
<TITLE>title text</TITLE>
```

Attributes None.

Example

```
<HEAD>
   <TITLE>The Greatest Web Page Ever</TITLE>
</HEAD>
```

<UL>

Unordered (bulleted) list.

Syntax

```
<UL>
   <LI>
   ...
   <LI>
</UL>
```

Attributes None.

Example

```
<UL>
   <LI>Ice cream
   <LI>Fudge sauce
   <LI>Chopped nuts
   <LI>Whipped cream
   <LI>Marashino cherry
</UL>
```

Appendix E

VoiceXML reference

This appendix provides reference information for WML 1.1 elements and attributes.

Note The W3C VoiceXML specification is available at http://www.w3.org/TR/voicexml. Platforms vary in their implementation of the specification. Check your platform documentation to determine its support for any specific feature.

Element reference

This section provides reference information for each element (tag) in VoiceXML, listing its syntax and attributes, and providing a simple example.

The syntax statements in this appendix use the following notation:

- `string`: any text string
- `identifiername="identifier"`: a name which must be a valid ECMAScript identifier; names beginning with the underscore character ("_") are reserved for internal use.
- `stringlist`: a space-delimited list of strings, each enclosed in single quotes (')
- `JS_expression`: a JavaScript expression
- `boolean`: a JavaScript expression that evaluates to true or false
- `interval`: a time interval, in units of seconds (s) or milliseconds (ms).
- `number`: a non-negative integer
- `numlist`: a comma-delimited list of integers
- `URL`: Universal Resource Locator for a file or other resource.
- `sequence`: a sequence of DTMF key input
- `event`: the name of an event
- `MIMEtype`: a standard Multipart Internet Mail Extension (MIME) type

A JavaScript expression can be a constant or a variable. String constants should be enclosed in single-quotes (`'`). Only integer constants are allowed.

The following operators are supported in an expression: ==, !=, <, <=, >, >=, +, -, *, /, %, &&, and ||. No unary operator is allowed, except the hyphen character (-) for integer values. One operator at most is allowed within each expression.

<assign>

Assigns a value to a variable, creating the variable if it does not already exist.

Syntax

```
<assign
   name="identifier"
   expr="'JS_expression'">
</assign>
```

Both name and expr attributes are required.

Parents

This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>

Children

None.

Example

```
<assign name="flavor" expr="'chocolate'"/>
<assign name="document.revision" expr="document.revision + 1"/>
```

<audio>

Plays an audio file or converts text to speech within a prompt.

Syntax

```
<audio
   src="URL"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
text
</audio>
```

Attributes

All attributes are optional.

Parents

This element can be contained within any of the following elements:

<block>	<filled>	<if>	<noinput>
<catch>	<help>	<menu>	<nomatch>
<enumerate>	<field>	<emp>	
<error>	<initial>	<choice>	<prompt>
<object>	<audio>		
<pros>	<record>	<subdialog>	<transfer>

Children This element can contain any of the following elements:

<break>		<emp>	<enumerate>
<pros>	<sayas>	<value>	

Example

```
<prompt>
   <audio src="welcome.wav">
      <emp>Welcome</emp> to Enhydra Voice Portal.
   </audio>
</prompt>
```

<block>

Specifies a block of directives that are executed if the block's form item variable is undefined and the block's cond attribute, if any, evaluates to true.

Syntax

```
<block
   name="identifier"
   expr="JS_expression"
   cond="boolean" >
child elements
</block>
```

All attributes are optional.

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<assign>	<goto>	<exit>	<audio>
<prompt>	<script>	<clear>	<if>
<var>	<reprompt>	<return>	<disconnect>
<submit>	<throw>	<enumerate>	<value>

<break>

Specifies a pause in speech output.

Syntax

```
<break
   msecs="number"
   size="[none|small|medium|large]" />
```

Either the msecs or the size the attribute can be specified, but not both.

Parents This element can be contained within any of the following elements:

<choice>	<prompt>	<enumerate>	<audio>
	<emp>	<pros>	

Children This element cannot contain other elements.

Example

```
<vxml>
   <form id="dummy1">
      <prompt> This is <emp>also</emp> computer-generated text.
         <break size="medium"/> Do you like it?
      </prompt>
   </form>
</vxml>
```

<catch>

Catches a user-defined or predefined system event and specifies what action to take next. If the event handler does not cause a change in flow control, such as by using a `<goto>`, `<reprompt>`, or `<throw>` directive, control passes to the next form.

Syntax

```
<catch
   event="event"
   count="num"
   cond="boolean" >
child elements
</catch>
```

Parents This element can be contained within any of the following elements:

<field>	<record>	<vxml>	<transfer>
<menu>	<form>	<initial>	<subdialog>
<object>			

Children This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<var>
<audio>	<goto>	<reprompt>	<clear>
<if>	<script>	<disconnect>	<enumerate>
<throw>	<return>	<submit>	<value>

Example

```
<form id="launch_missiles">
   <field name="password">
      <prompt>What is the code word?</prompt>
      <grammar>otter</grammar>
      <help>It is the name of a marine mammal.</help>
```

```
      <catch event="nomatch noinput" count="3">
         <prompt>Security violation!</prompt>
         <submit next="invalid" namelist="user_id"/>
      </catch>
   </field>
   <block>
      <goto next="#get_city"/>
   </block>
</form>
```

<choice>

Serves several purposes:

- It specifies a speech grammar fragment or a DTMF grammar fragment that determines when that choice has been selected.
- The contents are used to form the `<enumerate>` prompt string.
- It specifies the URL to go to when the choice in a menu is selected.

Syntax

```
<choice
   next="URL"
   dtmf="sequence"
   event="event"
   expr="JS_expression"
   fetchaudio="URL"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
grammar fragment
</choice>
```

Parents

This element can be contained within the following element:

<menu>

Children

This element can contain any of the following elements:

<audio>	<break>		<emp>
<enumerate>	<grammar>	<pros>	<sayas>
<value>			

Example

```
<menu>
   <property name="inputmodes" value="dtmf"/>
   <prompt>
      For otters press 1, For sealions press 2, For seagulls press 3.
   </prompt>
   <choice dtmf="1" next="http://www.lutris.com/vxml/otters.vxml"/>
   <choice dtmf="2" next="http://www.lutris.com/vxml/sealions.vxml"/>
   <choice dtmf="3" next="http://www.lutris.com/vxml/seagulls.vxml"/>
</menu>
```

<clear>

Resets one or more form items. Resetting includes:

- Setting the form item variable to ECMAScript undefined.
- Reinitializing the prompt counter and the event counters for the form item.

Syntax

```
<clear
   name="{variable}"/>
```

Parents This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>

Children This element cannot contain other elements.

<disconnect>

Terminates the user's phone call.

The `<disconnect>` directive causes the interpreter context to disconnect from the user. As a result, the interpreter context throws a `telephone.disconnected.hangup` event, which can be caught to do cleanup processing. A `<disconnect>` differs from an `<exit>` in that it forces the interpreter context to drop the call.

Syntax

```
<disconnect />
```

Parents This element can be contained within any of the following elements:

<catch>	<help>	<block>	<error>
<noinput>	<if>	<filled>	<nomatch>

Children This element cannot contain other elements.

Java API Speech Markup Language (JSML) element to classify a region of text as a particular type. Identifies the enclosed text as a particular type—either sentence or paragraph.

Syntax

```
<div
   type="[sentence|paragraph]" >
text
</div>
```

Parents This element can be contained within any of the following elements:

<choice>	<prompt>	<enumerate>	<audio>
	<emp>	<pros>	

Children This element cannot contain other elements.

Example

```
<vxml>
   <form id="dummy1">
      <prompt> This is <emp>also</emp> computer-generated text.
         <div type="sentence"> Do you like it?</div></prompt>
   </form>
</vxml>
```

<dtmf>

Used to specify a DTMF grammar that defines a set of key presses that a user can use to perform an action or supply information, and the corresponding string value that describes that information or action.

Syntax

```
<dtmf
   src="URL"
   scope="{document|dialog}"
   type="MIMEtype"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
text
</dtmf>
```

Parents This element can be contained within any of the following elements:

<form>	<field>	<link>	<transfer>

Children This element cannot contain other elements.

Example

```
<form id="getTT">
   <field name="ttinput">
      <prompt> press 2, 3, or star star 7 </prompt>
      <dtmf> 2 | 3 | **7 </dtmf>
   </field>
   <filled>
      <if cond="ttinput == '2'">
         <prompt> you entered 2 </prompt>
      <elseif cond="ttinput == '3'"/>
         <prompt> you entered 3 </prompt>
      <elseif cond="ttinput == '**7'"/>
         <prompt> you entered star star 7 </prompt>
      </if>
   </filled>
</form>
```

<else>

Used within a conditional logic statement along with the `<if>` and optional `<elseif>` elements.

Syntax

```
<else/>
```

Parents

This element can be contained within the following element:

<if>

Children

This element cannot contain other elements.

Example

```
<if cond="card_digits == 15">
   Please say or key in your 15 digit card number.
<else/>
   Please say or key in your 16 digit card number.
</if>
```

<elseif>

Used within a conditional logic statement along with the `<if>` and optional `<else>` elements.

Syntax

```
<elseif cond="boolean" />
```

Parents

This element can be contained within the following element:

<if>

Children

This element cannot contain other elements.

Example

```
<if cond="flavor == 'vanilla'">
   <assign name="flavor_code" expr="'v'"/>
<elseif cond="flavor == 'chocolate'"/>
   <assign name="flavor_code" expr="'h'"/>
<elseif cond="flavor == 'strawberry'"/>
   <assign name="flavor_code" expr="'b'"/>
<else/>
   <assign name="flavor_code" expr="'?'"/>
</if>
```

<emp>

Specifies that the enclosed text should be spoken with emphasis.

Syntax

```
<emp
   level="{string|moderate|none|reduced}" >
text
</emp>
```

Default `level` is moderate.

Parents This element can be contained within any of the following elements:

<choice>	<prompt>	<enumerate>	<audio>
	<emp>	<pros>	

Children This element cannot contain other elements.

Example

```
<form id="dummy1">
   <prompt>
      This is <emp>also</emp> computer-generated text. Do you like it?
   </prompt>
</form>
```

<enumerate>

An automatically generated description of the choices available to the user. It specifies a template that is applied to each choice in the order in which they appear in the menu. If it is used with no content, a default template that lists all the choices is used, determined by the interpreter context. If it has content, the content is the template specifier. This specifier can refer to two special variables: _prompt is the choice's prompt, and _dtmf is the choice's assigned DTMF sequence.

Syntax

```
<enumerate>
   child elements
</enumerate>
```

Parents This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<if>
<catch>	<help>	<nomatch>	<menu>
<error>	<initial>	<prompt>	<choice>
<enumerate>	<field>		<emp>
<pros>	<record>	<transfer>	<subdialog>
<object>	<audio>		

Children This element can contain any of the following elements:

<audio>	<break>		<emp>
<pros>	<sayas>	<value>	

Example

```
<menu dtmf="true">
   <prompt>
      Welcome to Otter's Voice Portal.
      <enumerate>
         For <value expr="_prompt"/>, press <value expr="_dtmf"/>.
      </enumerate>
   </prompt>
   <choice next="http://www.lutris.com/example/sports.po"> sports </choice>
   <choice next="http://www.lutris.com/example/weather.po"> weather </choice>
   <choice next="http://www.lutris.com/example/surf.po"> surf report </choice>
</menu>
```

<error>

Equivalent to `<catch event="error">` and catches all events of the type error.

Syntax

```
<error
   count="num"
   cond="boolean" >
child elements
</error>
```

Parents

This element can be contained within any of the following elements:

<vxml>	<form>	<menu>	<field>
<initial>	<record>	<transfer>	<subdialog>
<object>			

Children

This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<var>
<audio>	<goto>	<reprompt>	<clear>
<if>	<script>	<disconnect>	<enumerate>
<throw>	<return>	<submit>	<value>

Example

```
<error>
   An error has occurred -- please call again later.<exit/>
</error>
```

<exit>

Returns control to the interpreter context, which determines what to do next. This element differs from `<return>` in that it terminates all loaded documents, while `<return>` returns from a `<subdialog>` invocation. If the `<subdialog>` caused a new document (or application) to be invoked, `<return>` causes that document to be terminated, although execution resumes after the `<subdialog>`.

Once <exit> returns control to the interpreter context, the interpreter context is free to do as it wishes. For example, it might play a top-level menu for the user, drop the call, or transfer the user to an operator.

Syntax

```
<exit
   expr="JS_expression"
   namelist="stringlist" />
```

Parents This element can be contained within any of the following elements:

<catch>	<help>	<block>	<error>
<noinput>	<filled>	<nomatch>	<if>

Children This element cannot contain other elements.

```
<form id="GetPin">
   <field name="document.pin">
      <grammar>Four_digits</grammar>
      <prompt>Please enter your access code</prompt>
      <noinput><exit expr="login.failed"/></noinput>
      <filled>
         <if cond="{document.pin} == 1234">
            <exit expr="login.success"/>
         </if>
         <exit expr="login.invalid"/>
      </filled>
   </field>
</form>
```

<field>

Specifies an input to be gathered from the user.

Syntax

```
<field
   name="identifier"
   expr="JS_expression"
   cond="JS_expression"
   type="{digit|boolean|date|phone|currency|number|time}"
   slot="string"
   model="{false|true}" >
child elements
</field>
```

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<dtmf>	<enumerate>	<error>
<audio>	<filled>	<option>
<catch>	<property>	<noinput>

<link>	<value>	<nomatch>
<help>	<grammar>	<prompt>

Example

```
<field name="lo_fat_meal" type="boolean">
   <prompt>
      Do you want a low fat meal?
   </prompt>
   <help>
      Low fat means less than 12 grams of fat, and under 200 calories.
   </help>
   <filled>
      <prompt>
         I heard <emp><value expr="lo_fat_meal"/></emp>.
      </prompt>
   </filled>
</field>
```

<filled>

Specifies an action to perform when some combination of fields are filled by user input. This element can occur in two places: as a child of the `<form>` element, or as a child of a field item.

Syntax

```
<filled
   mode="{all|any}"
   namelist="stringlist" >
child elements
</filled>
```

Parents

This element can be contained within any of the following elements:

<form>	<field>	<record>	<transfer>
<subdialog>	<object>		

Children

This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<throw>
<audio>	<goto>	<reprompt>	<var>
<clear>	<if>	<return>	<disconnect>
<submit>	<script>	<enumerate>	<value>

Example

```
<field name="ready" type="boolean">
   <prompt> Are you ready to order now ? </prompt>
   <filled>
      <if cond="ready== 'TRUE'">
         <goto nextitem="entree" />
      <elseif cond="ready== 'FALSE'"/>
         <goto nextitem="notready" />
      </if>
   </filled>
</field>
```

<form>

Contains the core logic of your voice application, including grammars to use, prompts for the user, and code to execute based on user response. Specifically, can contain:

- A set of form items, elements that are visited in the main loop of the form interpretation algorithm. Form items are subdivided into field items, those that define the form's field item variables, and control items, those that help control the gathering of the form's fields.
- Declarations of non-field item variables.
- Event handlers.
- "Filled" actions, blocks of procedural logic that execute when certain combinations of field items are filled in.

Syntax

```
<form
   id="string"
   scope="{document|dialog}" >
child elements
</form>
```

Parents

This element can be contained within the following element:

<vxml>

Children

This element can contain any of the following elements:

<block>	<catch>	<transfer>	<field>
<grammar>	<error>	<filled>	<initial>
<help>	<link>	<nomatch>	<noinput>
<object>	<property>	<record>	<subdialog>
<var>	<dtmf>		

Example

```
<form>
   <field name="maincourse">
      <prompt>Please select an entree.<enumerate/> </prompt>
      <option dtmf="1" value="fish"> swordfish </option>
      <option dtmf="2" value="beef"> roast beef </option>
      <option dtmf="3" value="chicken"> frog legs </option>
      <filled>
         <submit next="/cgi-bin/maincourse.cgi" method="post" namelist="maincourse"/>
      </filled>
   </field>
</form>
```

<goto>

Used to:

- Transition to another form item in the current form
- Transition to another dialog in the current document
- Transition to another document

Syntax

```
<goto
   next="URL"
   expr= "JS_expression"
   nextitem="string"
   expritem="JS_expression"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval"
   fetchaudio="URL" />
```

Parents

This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>

Children

This element cannot contain other elements.

Example

```
<form id="maindish">
   <field name="done" type="boolean">
      <prompt> Are you done here ? </prompt>
      <filled>
         <if cond="done== 'FALSE'">
            <goto nextitem="notdone" />
         </if>
      </filled>
   </field>

...
   <block name="notdone">
   <prompt> Please continue. </prompt>
   <disconnect/>
   </block>
</form>
```

<grammar>

Defines the permissible vocabulary for a given user interaction—a speech recognition grammar. Provides a speech grammar that:

- Specifies a set of utterances that a user can speak to perform an action or supply information.
- Provides a corresponding string value (in the case of a field grammar) or set of attribute-value pairs (in the case of a form grammar) to describe the information or action.

Syntax

```
<grammar
   src="URL"
   scope="{dialog|document}"
   type="MIMEtype"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
text
</grammar>
```

Parents This element can be contained within any of the following elements:

<field>	<form>	<link>	<choice>
<record>	<transfer>		

Children This element cannot contain other elements.

Example

```
<!-- built-in grammar of type digits with length 6 -->
<grammar src ="builtin:grammar/digits?length=6"/>
<!-- external grammar -->
<grammar src="URL"/>
<!--inline specified grammar -->
<grammar type="application/x-jsgf"> red | green | yellow </grammar>
```

<help>

Catches a help event; equivalent to `<catch event="help">`.

Syntax

```
<help>
   child elements
</help>
```

Parents This element can be contained within any of the following elements:

<field>	<record>	<vxml>	<transfer>
<menu>	<form>	<initial>	<subdialog>
<object>			

Children This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<var>
<audio>	<goto>	<reprompt>	<clear>
<if>	<script>	<disconnect>	<enumerate>
<throw>	<return>	<submit>	<value>

Example

```
<help> No help is available </help>
```

<if>

Performs conditional logic. It has optional `<elseif>` and `<else>` elements.

Syntax

```
<if
   cond="boolean" >
child elements
</if>
```

Parents

This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	

Children

This element can contain any of the following elements:

<assign>	<else>	<elseif>	
<audio>	<goto>	<reprompt>	
<exit>	<script>	<disconnect>	
<throw>	<prompt>	<clear>	<enumerate>
<var>	<return>	<submit>	<value>

Example

```
<if cond="ttinput == '1'">
   <assign name="action" expr="'transfer call'">
<elseif cond="ttinput == '2'"/>
   <assign name="action" expr="'check voicemail'">
<else/>
   <assign name="action" expr="'undefined'">
</if>
```

<initial>

Declares initial logic upon entry into a (mixed-initiative) form. In a typical mixed-initiative form, the `<initial>` element is visited when the user is initially being prompted for form-wide information, and has not yet entered into the directed mode where each field is solicited individually.

Like field items, it has prompts, catches, and event counters. Unlike field items, `<initial>` has no grammars, and no `<filled>` action.

Syntax

```
<initial
   name="identifier"
   expr="JS_expression"
   cond="boolean" >
child elements
</initial>
```

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<catch>	<enumerate>	<error>	<audio>
<help>	<link>	<nomatch>	<noinput>
<prompt>	<property>	<value>	

Example

```
<form>
   <initial>
      <audio src="welcome.wav">Welcome to OtterTalk</audio>
      <assign name="document.company.flag" expr="'false'"/>
      <if cond="{document.company.flag} == 'false'">
         <assign name="document.company" expr="'OtterTalk'"/>
      </if>
      <goto next="_home"/>
   </initial>
</form>
```

<link>

Specifies one or more grammars and a destination to go to when a grammar is matched. A `<link>` element has one or more grammars, which are scoped to the element containing the `<link>`. Grammar elements contained in the `<link>` are not permitted to specify scope. When one of these grammars is matched, the link activates, and either:

- Transitions to a new document or dialog (like `<goto>`)
- Throws an event (like `<throw>`)

Syntax

```
<link
   next="URL"
   expr="JS_expression"
   event="event"
   fetchaudio="URL"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
child elements
</link>
```

Only one of the attributes next, expr, or event may be specified.

Parents This element can be contained within any of the following elements:

<vxml>	<form>	<field>	<initial>

Children This element can contain any of the following elements:

<grammar>	<dtmf>

Example

```
<!-- This link activates when you say “otters” or press “2” -->
<link next="http://www.enhydra.org/voice.vxml">
   <grammar type="application/x-jsgf"> Otters | Sea Otters </grammar>
   <dtmf> 2 </dtmf>
</link>
```

<menu>

Presents a list of choices to the user and transitions to the chosen information. A menu is a convenient syntactic shorthand for a form that contains a single anonymous field that prompts the user to make a choice and then transitions to different places based on that choice.

Like a regular form, a menu can have its grammar scoped such that it is active when the user is executing another dialog.

Syntax

```
<menu
   id="string"
   scope="dialog"
   dtmf="{true|false}" >
child elements
</menu>
```

Parents

This element can be contained within the following element:

<vxml>

Children

This element can contain any of the following elements:

<choice>	<help>	<nomatch>	<catch>
<noinput>	<prompt>	<enumerate>	<error>
<property>	<value>	<audio>	

Example

```
<menu>
   <prompt> What information do you want, sports, weather, or stock</prompt>
   <choice next="#exit"> i am done here </choice>
   <choice next="sports.vxml"> sports </choice>
   <choice next="weather.vxml"> weather </choice>
   <choice next="stock.vxml"> stock</choice>
</menu>
```

<meta>

Specifies metadata, as in HTML. Metadata is data about the document, not data about the document’s content. There are two types of `<meta>`. The first type specifies a metadata property of the document as a whole. The second type of `<meta>` specifies HTTP response headers.

Syntax

```
<meta
   name="identifier"
   content="string"
   http-equiv="string" />
```

Parents This element can be contained within any of the following elements:

<vxml>

Children This element cannot contain other elements.

<noinput>

Handles cases where the user does not speak. The `<noinput>` element is an abbreviation for `<catch event="noinput">`.

Syntax

```
<noinput>
child elements
</noinput>
```

Parents This element can be contained within any of the following elements:

<field>	<record>	<vxml>	<transfer>
<menu>	<form>	<initial>	<subdialog>
<object>			

Children This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<var>
<audio>	<goto>	<reprompt>	<clear>
<if>	<script>	<disconnect>	<enumerate>
<throw>	<return>	<submit>	<value>

Example See “<catch>” on page 166.

<nomatch>

Catches a nomatch event. The `<nomatch>` element is short for `<catch event="nomatch">`.

Syntax

```
<nomatch>
child elements
</nomatch>
```

Parents This element can be contained within any of the following elements:

<field>	<record>	<vxml>	<transfer>
<menu>	<form>	<initial>	<subdialog>
<object>			

Children This element can contain any of the following elements:

<assign>	<exit>	<prompt>	<var>
<audio>	<goto>	<reprompt>	<clear>
<if>	<script>	<disconnect>	<enumerate>
<throw>	<return>	<submit>	<value>

Example See “<catch>” on page 166.

<object>

Interacts with a custom extension. A VoiceXML implementation platform may have platform-specific functionality that an application wants to use, such as speaker verification, native components, additional telephony functionality, and so on. Such platform-specific objects are accessed using the `<object>` element, which is analogous to the HTML `<OBJECT>` element.

Syntax

```
<object
   name="identifier"
   classid="URL"
   data="URL"
   expr="JS_expression"
   cond="boolean"
   codebase="URL"
   codetype="MIMEtype"
   type="MIMEtype"
   archive="URL URL ..."
   fetchaudio="URL"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" />
```

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<audio>	<catch>	<enumerate>	<error>
<filled>	<help>	<nomatch>	<noinput>
<param>	<prompt>	<property>	<value>

Example

```
<form id="gather_pager_message">
   <object name="message" classid="builtin://keypad_text_input">
      <prompt>
         Enter your message by pressing your keypad once per letter.
         For a space, enter star. To end the message, press the pound sign.
      </prompt>
</object>
```

<option>

Specifies an option in a `<field>`. When a simple set of alternatives is all that is needed to specify the legal input values for a field, it may be more convenient to use an option list than a grammar. An option list is represented by a set of `<option>` elements contained in a `<field>` element.

Each `<option>` element contains text that is used to generate a grammar for the spoken input it accepts using the same method described for `<choice>`. It also has attributes specifying the DTMF key for selecting the option and the value to assign to the field when the option is chosen.

Syntax

```
<option
   dtmf="sequence"
   value="string" >
   text
</option>
```

Parents

This element can be contained within the following element:

<field>

Children

This element cannot contain other elements.

Example

```
<field name="maincourse">
   <prompt>Please select an entree. Today, we're featuring <enumerate/></prompt>
   <option dtmf="1" value="fish"> red snapper </option>
   <option dtmf="2" value="vegetarian"> tofu casserole </option>
   <option dtmf="3" value="chicken"> chicken kiev </option>
   <filled>
      <submit next="/dinnerapp/maincourse.po" method="post" namelist="maincourse"/>
   </filled>
</field>
```

<param>

Used to specify values that are passed to subdialogs or objects. It is modeled on the HTML `<PARAM>` element.

Syntax

```
<param
   name="identifier"
   expr="JS_expression"
   value="string"
   valuetype={data|ref}"
   type="MIMEtype" />
```

Parents This element can be contained within any of the following elements:

<subdialog> <object>

Children This element cannot contain other elements.

Example

```
<!-- param used as part of an object -->
<object  name="debit" classid="method://credit_card/gather_and_debit"
   data="http://www.recordings.example/prompts/credit/jesse.jar"/>
   <param name="amount" expr="document.amt"/>
   <param name="vendor" expr="vendor_num"/>
   <param name="application_id" value="ADC5678-QWOO"/>
   <param name="authentication_server" value="http://auth_svr.example" valuetype="ref"
      type="text/plain"/>
</object>

<!-- param used with subdialog -->
<form>
   <subdialog name="result" src="http://another.example/#getssn">
      <param name="firstname" expr="document.first"/>
      <param name="lastname" expr="document.last"/>
      <filled>
         <submit namelist="result.ssn"
next="http://myservice.example/cgi-bin/process"/>
      </filled>
   </subdialog>
</form>
```

<prompt>

Controls the output of synthesized speech and prerecorded audio. Prompts are instantaneously queued for playing, so interpretation proceeds until the user needs to provide an input. Then the prompts are played, and the system waits for user input. Once the input is received from the speech recognition subsystem (or the DTMF recognizer), interpretation proceeds.

Syntax

```
<prompt
   bargein="{true|false}"
   cond="boolean"
   count="numlist"
   timeout="interval">
child elements
</prompt>
```

Parents This element can be contained within any of the following elements:

<field>	<record>	<transfer>	<block>
<menu>	<catch>	<error>	<help>
<noinput>	<nomatch>	<if>	<filled>
<initial>	<subdialog>	<object>	

Children This element can contain any of the following elements:

<break>	<audio>		<emp>
<enumerate>	<pros>	<sayas>	<value>

Example

```
<prompt>
   <audio src="welcome.wav"><emp>Welcome</emp> to Otter's Voice Portal.</audio>
</prompt>
```

<property>

Sets a property value. Properties are used to set values that affect platform behavior, such as the recognition process, timeouts, caching policy, and so on.

Syntax

```
<property
   name="identifier"
   value="string" />
```

Parents This element can be contained within any of the following elements:

<vxml>	<form>	<menu>	<field>
<initial>	<record>	<transfer>	<subdialog>
<object>			

Children This element cannot contain other elements.

Example Standard VoiceXML properties and their default values:

```
<property name="CONFIDENCELEVEL" value="0.5" />
<property name="SENSITIVITY" value="0.5" />
<property name="SPEEDVSACCURACY" value="0.5" />
<property name="COMPLETETIOMEOUT" value="1s" />
<property name="INCOMPLETETIMEOUT" value="1s" />
<property name="INTERDIGITTIMEOUT" value="3s" />
<property name="TERMTIMEOUT" value="0s" />
<property name="TERMCHAR" value="#" />
<property name="BARGEIN" value="TRUE" />
<property name="TIMEOUT" value="10s" />
<property name="CACHING" value="FAST" />
<property name="AUDIOFETCHHINT" value="PREFETCH" />
<property name="DOCUMENTFETCHHINT" value="SAFE" />
<property name="GRAMMARFETCHHINT" value="PREFETCH" />
<property name="OBJECTFETCHHINT" value="PREFETCH" />
<property name="SCRIPTFETCHHINT" value="PREFETCH" />
<property name="FETCHAUDIO" value="buildin:background_audio.wav" />
<property name="FETCHTIMEOUT" value="30s" />
<property name="INPUTMODES" value="DTMF VOICE" />
```

<pros>

Specifies prosodic information for the enclosed text. For details about the format of attribute values, see the *Java API Speech Markup Language* specification.

Syntax

```
<pros
   rate="num"
   vol="num"
   pitch="num"
   range="num" >
text
</pros>
```

Parents This element can be contained within any of the following elements:

<choice> <prompt> <enumerate> <audio>

 <emp> <pros>

Children This element cannot contain other elements.

Example

```
<prompt>
   <pros rate="medium" vol="medium" pitch="medium" range="medium">
      This is also computer-generated text.
   </pros>
   Do you like it?
</prompt>
```

<record>

Records an audio sample from the user. The recording is stored in the field item variable, which can be played back or submitted to a server.

Syntax

```
<record
   name="identifier"
   expr="JS_expression"
   cond="boolean"
   modal="{true|false}"
   beep="{true|false}"
   maxtime="interval"
   finalsilence="interval"
   dtmfterm="{true|false}"
   type="{audio/vox|audio/basic|audio/wav}" >
</record>
```

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<audio>	<catch>	<enumerate>	<error>
<filled>	<help>	<nomatch>	<noinput>
<grammar>	<prompt>	<property>	<value>

Example

```
<record name="awaymessage" beep="true" maxtime="10s" finalsilence="4s"
      dtmfterm="true">
   <prompt> At the tone, please record your message</prompt>
</record>
```

<reprompt>

Replays the previously played prompt. The form interpretation algorithm (FIA) assumes that when a catch element is executed, it has queued appropriate prompts. Therefore the FIA normally suppresses playing of prompts on the iteration of the FIA following the execution of a catch element. However, if a `<reprompt>` is executed in the catch, this tells the FIA that when it selects the next form item to visit, it should do the normal prompt processing (which includes selection of a prompt and incrementing the prompt counter).

Syntax

```
<reprompt/>
```

Parents This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>

Children This element cannot contain other elements.

Example

```
<form id="moreorder">
   <field name="whatnext">
      <prompt> do you want anything else? </prompt>
      <prompt count="2"> Any desserts? </prompt>
      <prompt count="3"> Say desserts or bring my bill </prompt>
      <grammar type="application/x-jsgf">
         yes | no| bring my bill | desserts
      </grammar>
      <noinput>
         <reprompt/>
   </field>
</form>
```

<return>

Returns control to the last application added to the subdialog stack. If the event attribute is not specified, it returns control to the <filled> element of the appropriate <subdialog> element. If the event attribute is specified, it returns

control to the other event handlers of the <subdialog> element by throwing the specified event.

Syntax

```
<return
   event="event"
   namelist="stringlist" />
```

Parents

This element can be contained within any of the following elements:

<block>	<catch>	<error>	<help>
<noinput>	<nomatch>	<if>	<filled>

Children

This element cannot contain other elements.

Example

```
<form id="basic">
   <field name="acctnum" type="digits">
      <prompt> What is your account number? </prompt>
   </field>
   <field name="acctphone" type="phone">
      <prompt> What is your home telephone number? </prompt>
      <filled>
      <return namelist="acctnum acctphone"/>
      </filled>
   </field>
</form>
```

<sayas>

Specifies how a word or phrase is spoken.

Syntax

```
<sayas
   phon="string"
   sub="string"
   class="{phone|date|digits|literal|currency|number|time} >
text
</sayas>
```

Parents

This element can be contained within any of the following elements:

<choice>	<prompt>	<enumerate>	<audio>
	<emp>	<pros>	

Children

This element cannot contain other elements.

Example

```
<prompt>You are calling
   <sayas class="phone">312-555-1212</sayas>
</prompt>
```

<script>

Includes a block of JavaScript code.

Syntax

```
<script
   src="URL"
   charset="encoding"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
text
</script>
```

Parents This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>
<vxml>			

Children This element cannot contain other elements.

<subdialog>

Invokes another dialog as a subdialog of the current dialog, identified by its src attribute. The subdialog executes in a new execution context. The subdialog proceeds until the execution of a `<return>` element, which causes the subdialog to return.

When the subdialog returns, its execution context is deleted, and execution resumes in the calling dialog with any appropriate `<filled>` elements. An execution context includes all declarations and state information for the dialog, the dialog's document, and the application root (if present). Subdialogs can permit the reuse of a common dialog, or build libraries of reusable applications.

Syntax

```
<subdialog
   name="identifier"
   src="URL"
   expr="JS_expression"
   cond="boolean"
   modal="{true|false}"
   namelist="stringlist"
   method="{get|post}"
   enctype="MIMEtype"
   fetchaudio="URL"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval" >
child elements
</subdialog>
```

Parents This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<audio>	<catch>	<enumerate>	<error>
<filled>	<help>	<nomatch>	<noinput>
<param>	<prompt>	<property>	<value>

Example

```
<form>
   <subdialog name="result" src="#getdriverslicense">
      <param name="birthday" expr="'2000-07-27'"/>
      <filled>
         <submit next="http://foo.lutris.com/myapp/process.po"/>
      </filled>
   </subdialog>
</form>
```

<submit>

Submits values to a document server. The `<submit>` element is similar to `<goto>` in that it results in a new document being obtained. Unlike `<goto>`, it lets you submit a list of variables to the document server via an HTTP GET or POST request.

Syntax

```
<submit
   next="URL"
   expr="JS_expression"
   namelist="stringlist"
   enctype="MIMEtype"
   method="{get|post}"
   caching="{safe|fast}"
   fetchhint="{prefetch|safe|stream}"
   fetchtimeout="interval"
   fetchaudio="URL" />
```

Parents This element can be contained within any of the following elements:

<block>	<catch>	<error>	<help>
<noinput>	<nomatch>	<if>	<filled>

Children This element cannot contain other elements.

Example

```
<submit next="/cgi-bin/login.po" method="post" namelist="acct_num pin"
fetchtimeout="1000" />
```

<throw>

Generates a system or user-defined event to catch with an event handler.

Syntax

```
<throw
   event="event" />
```

where *event* is the user-defined event to throw (required).

Parents

This element can be contained within any of the following elements:

<catch>	<help>	<block>	<error>
<noinput>	<if>	<filled>	<nomatch>

Children

This element cannot contain other elements.

Example

```
<if cond="password=='goodbye'">
   <throw event="goodbye"/>
<else/>
This is correct.
</if>
```

<transfer>

Transfers the caller to another phone number. Occasionally, it is appropriate to suspend the session between the user and the interpreter and initiate a session with another entity. The most common use for this capability is to connect a user to a third party through the telephone network. The `<transfer>` element directs the interpreter to make such a third-party connection. Two scenarios are supported:

- **Bridging:** The original caller resumes his or her session with the interpreter.
- **Blind transfer:** No resumption is possible; as soon as the call connects, the platform throws a `telephone.disconnect.transfer`.

Syntax

```
<transfer
   name="identifier"
   expr="JS_expression"
   cond="boolean"
   dest="URL"
   destexpr="JS_expression"
   bridge="{true|false}"
   connecttimeout="interval"
   maxtime="interval" >
child elements
</transfer>
```

Parents

This element can be contained within the following element:

<form>

Children This element can contain any of the following elements:

<catch>	<audio>	<dtmf>	<enumerate>
<error>	<filled>	<grammar>	<help>
<nomatch>	<noinput>	<prompt>	<property>
<value>			

Example

```
<?xml version="1.0"?>
<transfer name="newcall" dest="phone://7360905x247" connecttimeout="10s"
bridge="true">
   <filled>
      You call last <value expr="newcall$.duration"/> seconds.
      <if cond="newcall == 'busy'">
         <prompt>
            All our agents are currently busy, please call back later.
         </prompt>
      </if>
   </filled>
</transfer>
```

<value>

Inserts the value of a expression in a prompt.

Syntax

```
<value
   expr="JS_expression"
   mode="{tts|recorded}"
   class="sayas_class"
   recsrc="URL"
/>
```

Parents This element can be contained within any of the following elements:

<menu>	<choice>	<prompt>	<enumerate>
<field>	<initial>	<audio>	
<emp>	<pros>	<record>	<transfer>
<subdialog>	<object>		

Children This element cannot contain other elements.

Example

```
<var name="hi" expr="'Hello World!'"/>
<form>
   <block>
      <value expr="hi"/>
      <goto next="#say_goodbye"/>
   </block>
</form>
<form id="say_goodbye">
   <block> Goodbye! </block>
</form>
```

<var>

This element declares a local variable. It can occur in executable content or as a child of <form> or <vxml>.

Syntax

```
<var
   name="identifier"
   expr="JS_expression" />
```

Parents This element can be contained within any of the following elements:

<block>	<filled>	<noinput>	<error>
<catch>	<help>	<nomatch>	<if>
<form>	<vxml>		

Children This element cannot contain other elements.

<vxml>

Identifies a document as a VoiceXML document. All VoiceXML documents start with the <vxml> element and end with the </vxml> element.

Syntax

```
<vxml
   application="URL"
   version="1.0"...
   base="URL"
   lang="string" >
child elements
</vxml>
```

Parents This is the top-level VoiceXML element, and cannot be contained within any other elements

Children This element can contain any of the following elements:

<form>	<link>	<meta>	<catch>
<menu>	<var>	<error>	<help>
<nomatch>	<noinput>	<property>	<script>

Appendix F

XHTML Basic reference

Every XHTML element has the attributes listed in Table F.1, except for base, head, html, meta, param, script, style, and title.

Table F.1 XHTML Basic common attributes

Name	Description
class	The class of the element.
id	String that identifies the object.
title	Text to display in tool tip.
xml:lang	Human language for the element's contents with ISO 639 standard two-character name.
accesskey	Accessibility character
tabindex	Position in tabbing order

Element reference

This section provides a comprehensive reference for every element in XHTML Basic.

<a>

Defines an anchor element. An anchor can be used in two ways:

- To create a link to another document by using the href attribute.
- To create an anchor in a document, by using the name or id attribute.

Note A linked page is normally displayed in the current browser window, unless you specify another target:

Attributes The following attributes are optional:

Name	Value	Description
charset	character_encoding	Sets the character encoding of the page where the link points.
href	url	Creates a link to a document.

Name	Value	Description
hreflang	language_code	Sets the base language of the page where the link points.
name	section_name	Names the anchor so it can be used as the target of a URL in the href attribute. This attribute makes it possible to jump to a specific place in a document. To use this position, add the character #, followed by the name, to the URL in the link. The id attribute can be used instead of the name attribute.
type	mime_type	Gives a hint about the content type of the content where the link points.
rel	alternate designates stylesheet start next prev contents index glossary copyright chapter section subsection appendix help bookmark	Defines a link relationship from the current document to the linked document.
rev	alternate designates stylesheet start next prev contents index glossary copyright chapter section subsection appendix help bookmark	Defines a link relationship from the linked document to the current document.

Example

Link to Lutris web site:

```
<a href="www.lutris.com">Lutris</a>
```

Opening a new window:

```
<a href="www.lutris.com" target="_blank>Lutris</a>
```

<abbr>

Indicates an abbreviated form, like “Inc.” or “etc.”. By marking up abbreviations you can give useful information to browsers, spell checkers, translation systems, and search-engine indexers.

Attributes None, other than common attributes

Example

```
<abbr title="United Nations">UN</abbr>
```

<acronym>

Defines the start of an acronym, like “NATO” or “FBI.” By marking up acronyms you can give useful information to browsers, spell checkers, translation systems and search-engine indexers.

Attributes None, other than common attributes.

Example

```
<acronym title="World Wide Web">WWW</acronym>
```

<address>

Defines the start of an address. You should use it to define addresses, signatures, or authorships of documents.

Note The address usually renders in *italic*. Most browsers add a line break before and after the address element, but you have to insert any line breaks inside the text yourself.

Attributes None, other than common attributes.

Example

```
<address>
Ollie Otter<br />
1200 Pacific Ave<br />
Santa Cruz, CA
</address>
```

<base>

Defines a default reference to external resources.

This element (with the href attribute) defines a default reference to external resources such as documents, images, and style sheets.

Note The <base> element must go inside the head element.

Attributes The following attributes are optional:

Name	Value	Description
href	url	Defines the default base address for relative links.

Example The absolute address for an image (on this website) is:

```
<img src="http://www.lutris.com/images/back40.gif" />
```

When you use the <base> element, you can specify the default reference in the head section:

```
<base href="http://www.lutris.com/" />
```

Then, when you want to insert an image later in the document, you just write the relative address, instead of the absolute address:

```
<img src="images/smile.gif" />
```

<blockquote>

Defines the start of a long quotation. It creates whitespace on both sides of the text.

Attributes The following attributes are optional:

Name	Value	Description
cite	url	URL of the quote, if it is taken from the Web.

Example

```
Here comes a long quotation:
<blockquote> here is a long quotation here is a long quotation here is a long
quotation here is a long quotation here is a long quotation here is a long quotation
here is a long quotation here is a long quotation here is a long quotation here is a
long quotation here is a long quotation </blockquote>
```

<body>

The body element defines the document's body. It contains all the contents of the document (text, images, colors, graphics, and so on).

Attributes None, other than common attributes.

Example

```
<html>
<head>
</head>

<body>
The content of the document......
</body>

</html>
```


Inserts a single line break. Use the <br> element to enter blank lines, not to separate paragraphs.

Attributes None, other than common attributes.

Example

```
This text contains<br />
a line break
```

<caption>

Defines a table caption. The <caption> element must be inserted immediately after the <table> element. You can specify only one caption per table. The caption is usually centered above the table.

Attributes None, other than common attributes.

Example

```
<table border="1">
<caption>This is a caption</caption>
<tr>
<td>Cell 1</td>
<td>Cell 2</td>
</tr>
</table>
```

<cite>

Defines a citation.

Attributes None, other than common attributes.

Example

```
<cite>Citation</cite>
```

<code>

Defines computer code text.

Attributes None, other than common attributes.

Example

```
<code>Computer code text</code>
```

<dfn>

Defines a definition term.

Attributes None, other than common attributes.

Example

```
<dfn>Definition term</dfn>
```

<dd>

Defines the description of the term in a definition list.

Attributes None, other than common attributes.

Example

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink<dd>
</dl>
```


Defines the start of a division/section in a document. Browsers usually place a line break before the element. Use the element to group block-elements.

Attributes None, other than common attributes.

Example

```
This is some text <div>
<h4>This is a header in a div section</h4>
<p>This is a paragraph in a div section</p>
</div>
```

<dl>

Defines the start of a definition list.

Attributes None, other than common attributes.

Example

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink<dd>
</dl>
```

<dt>

Defines the start of a term in a definition list.

Attributes None, other than common attributes.

Example

```
<dl>
<dt>Coffee</dt>
<dd>Black hot drink</dd>
<dt>Milk</dt>
<dd>White cold drink<dd>
</dl>
```


Renders as emphasized text.

Attributes None, other than common attributes.

Example

```
<em>Emphasized text</em>
```

<form>

Creates a form for user input. A form can contain text fields, check boxes, radio buttons, and so on. With forms the user can pass data to the server. The <form> element can be used without any attributes, but most forms require the action attribute to do something meaningful.

Attributes The following attribute is required:

Name	Value	Description
action	url	Specifies where to send the data when the user pushes the submit button in a form.

The following attributes are optional:

Name	Value	Description
method	get post	The HTTP method of passing the data to the Web server. When you use `method="get"`, the form input is submitted as an request with the form data appended to the URL, following a question mark (?). This is the default method. But if the form input contains non-ASCII characters or is more than 100 characters you **must** use `method=post`. With `method="post"`, the form input is submitted as a request with the form data sent in the body of the request. Most browsers are unable to bookmark post requests.
enctype	MIME type	The MIME type used to encode the content of the form.

Example

```
<form enctype=”multipart/form-data” method=”post” action=”processform.po”>
...
</form>
```

<head>

Defines the document header.

Attributes The following attributes is optional:

Name	Value	Description
profile	url	The address of the profile containing document metadata.

Example

```
<head>
   <title>Document title</title>
</head>
```

<hn>

The <h1> to <h6> elements define headers.

Attributes None, other than common attributes.

Example

```
<h1>This is header 1</h1>
<h2>This is header 2</h2>
<h3>This is header 3</h3>
<h4>This is header 4</h4>
<h5>This is header 5</h5>
<h6>This is header 6</h6>
```

<html>

Container element for an XHTML document.

Attributes The following attributes are optional:

Name	Value	Description
version	string	The version of the XHTML standard being used.
xmlns	string	The primary XML namespace used in the document.

Example

```
<html version="-//W3C//DTD XHTML 1.0 Strict//EN" xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```


Defines an image.

Attributes The following attributes are required:

Name	Value	Description
src	url	The address of the image you want to insert.
alt	text	A short description of the image. Use it for text-only browsers.

The following attributes are optional:

Name	Value	Description
longdesc	url	An address to a document that contains a long description of the image (for text-only browsers).
width	*pixels* %	Sets the width of an image.
height	*pixels* %	Sets the height of an image.

Example

```
<img src="../images/bulbon.gif" alt="Light" />
```

<input>

Defines the start of an input field where the user can enter data. The input element is empty; it contains attributes only.

Attributes The following attributes are optional:

Name	Value	Description
type	text checkbox radio password hidden submit reset	Indicates the type of the input field. Required. The default value is `text`.
name	field_name	The name of the input field. Radio buttons are grouped together with this attribute when you give them the same name—with the result being that you can select only one of the radio buttons in the group.
value	value	For check boxes and radio buttons, it specifies the value to be returned when the control is turned on. For buttons, reset buttons, and submit buttons it specifies the text of the button. For hidden, password, or text type elements, it specifies the default value of the control.
size	number_of_char	The width of the input field. The default value is 20 characters. Used with `type="password"` and `type="text"`.

Name	Value	Description
maxlength	number	Used with `type="text"` or `type="password"` to set the maximum number of characters the user can enter in the field.
checked	checked	Indicates that a radio button or a check box is "checked" (selected) when the form loads.

Example

```
<form method="post" action="html_form_action.po">
Height <input type="text" name="height">
</input>
<br />
Weight <input type="text" name="weight">
</input>
<br />
<input type="submit" value="Submit">
</input>
<input type="reset" value="Reset">
</input>
</form>
```

<kbd>

Defines keyboard text.

Attributes None, other than common attributes.

Example

```
<kbd>Keyboard text</kbd>
```

<label>

Defines a label to a control. If you click the text within the label element, it is supposed to toggle the control.

Note The "for" attribute binds a label to another element. Set the value of the "for" attribute equal to the value of the "id" attribute of the related element.

Attributes The following attributes are optional:

Name	Value	Description
for	id_of_another_field	Relates the label to a field in the form.

Example

```
<label for="lname">Last Name:</label>
<input type="text" name="lastname" id="lname"></input>
```


Defines the start of a list item. The <li> element is used in both ordered (<ol>) and unordered (<ul>) lists.

Attributes None, other than common attributes.

Example

```
<ol>
<li>Coffee</li>
<li>Tea</li>
</ol>
<ul>
<li>Coffee</li>
<li>Tea</li>
</ul>
```

<link>

Defines the relationship between two linked documents. This element goes only in the head section, but can appear any number of times.

Note The link element is an empty element; it contains attributes only.

Attributes The following attributes are optional:

Name	Value	Description
href	url	A URL to the linked resource.
hreflang	language_code	The base language of the linked resource.
type	*MIME_type* text/css text/javascript	The MIME type.
charset	charset	Sets the character encoding.
media	screen tty tv projection handheld print braille aural all	Which medium the link applies to.
rel	alternate appendix bookmark chapter contents copyright glossary help index next prev section start stylesheet subsection	Defines a link relationship from the current document to the linked document.

Name	Value	Description
rev	alternate appendix bookmark chapter contents copyright glossary help index next prev section start stylesheet subsection	Defines a link relationship from the linked document to the current document.

Example This example demonstrates how to use the <link> element to link to an external style sheet.

```
<head>
<link rel="stylesheet" type="text/css" href="theme.css"></link>
</head>
```

<meta>

Provides meta-information about your page, such as descriptions and keywords for search engines.

Note The <meta> element always goes inside the head element.

Attributes The following attributes are required:

Name	Value	Description
content	text	Sets meta information to be associated with http-equiv or name.

The following attributes are optional:

Name	Value	Description
name	author description keywords generator revised *others*	The name of the meta-object.
scheme	text	Defines a scheme to be used.
http-equiv	content-type expires refresh set-cookie	Sets whether the content of the <meta> element is bound to an http response header.

Example This meta element defines keywords for search engines:

```
<meta name="keywords" content="XHTML-Basic"></meta>
```

This meta element defines a description of your page:

```
<meta name="description" content="Free Web tutorials on Enhydra"></meta>
```

This meta element defines the last revision of your page:

```
<meta name="revised" content="YourName, 6/10/99"></meta>
```

<object>

Defines an embedded object. Use this element to insert multimedia into your XHTML page. This element allows you to specify the data and parameters for objects inserted into HTML documents, and the code that can be used to display/manipulate that data. To include applets, use the object element (because the <applet> element is deprecated).

Note An object element can appear inside the head or the body element. The text between the <object> and </object> is the alternate text, for browsers that do not support this element. The <param> element defines runtime settings for an object.

Attributes The following attributes are optional:

Name	Value	Description
archive	urls	A space-separated list of addresses to archives. The archives contains resources relevant to the object.
classid	url	The location of an object's implementation.
codebase	url	The base path to resolve relative URLs specified by the `classid`, `data`, and `archive` attributes.
codetype	mimetype	The Internet media type.
data	url	The URL that refers to the object's data.
declare	declare	Sets an object only as a declaration.
height	*pixels* %	The height of the object.
width	*pixels* %	The width of the object.
standby	text	Gives the user a message while the object is loading.
type	MIME_type	MIME type of data specified in the data attribute.

Example

```
<object classid="clsid:F08DF954-8592-11D1-B16A-00C0F0283628" id="Slider1" width="100"
height="50">
<param name="BorderStyle" value="1"></param>
<param name="MousePointer" value="0"></param>
<param name="Enabled" value="1"></param>
<param name="Min" value="0"></param>
<param name="Max" value="10"></param>
</object>
```


Defines the start of an ordered list.

Attributes None, other than common attributes.

Example

```
<ol>
<li>Coffee</li>
<li>Tea</li>
</ol>
```

<option>

Defines an option in the pull-down list. Use this element in the select element (to accept user input); it is meaningless elsewhere.

Note The <option> element can be used without any attributes, but you usually need the value attribute, which indicates what is sent to the server.

Attributes The following attributes are optional:

Name	Value	Description
value	text	Sets the value of the option.
selected	selected	The default selected option.

Example

```
<select>
<option value ="volvo">Volvo</option>
<option value ="saab">Saab</option>
<option value ="opel">Opel</option>
<option value ="audi">Audi</option>
</select>
```

<p>

Defines a paragraph.

Note All the “presentation attributes” of the p element have been deprecated, in favor of style sheets.

Attributes None, other than common attributes.

Example

```
<p>This is some text in a very short paragraph</p>
```

<param>

Specifies the runtime settings for an object inserted into XHTML documents.

Attributes The following attributes are optional:

Name	Value	Description
name	text	The name of the parameter.
value	value	The value of a runtime parameter.
valuetype	data ref object	The type of the value attribute.
type	mimetype	The Internet media type.

Example

```
<object classid="clsid:F08DF954-8592-11D1-B16A" id="Slider1" width="100" height="50">
   <param name="BorderStyle" value="1"></param>
   <param name="MousePointer" value="0"></param>
   <param name="Enabled" value="1"></param>
   <param name="Min" value="0"></param>
   <param name="Max" value="10"></param>
</object>
```

<pre>

Defines preformatted text. The text enclosed in the pre element usually preserves spaces and line breaks. The text renders in a fixed-pitch font.

Attributes None, other than common attributes.

Example

```
<pre>
This text is
in a fixed-pitch
font, and it preserves
both      spaces and
line breaks
</pre>
```

<q>

Defines the start of a short quotation.

Note The q element does not render text in a special way; you must use styles to format the text.

Attributes None, other than common attributes.

Example

```
Here comes a short quotation:
<q>here is a short quotation here is a short quotation</q>
```

<samp>

Defines sample computer code.

Attributes None, other than common attributes.

Example

```
<samp>Sample computer code text</samp>
```

<select>

Creates a pull-down list.

Note Use this element in the form element (to accept user input); it is meaningless elsewhere.

Attributes The following attributes are optional:

Name	Value	Description
multiple	true false	Sets whether multiple items can be selected.
name	text	Sets the name of the list.
size	number	Sets the number of visible items in the list.

Example

```
<select>
<option value ="volvo">Volvo</option>
<option value ="saab">Saab</option>
<option value ="opel">Opel</option>
<option value ="audi">Audi</option>
</select>
```


Defines the start of a section in a document.

Note Use the element to group inline elements so you can format them with styles.

Example

```
<p>This is a paragraph
<span style="color:#0000FF;">This is a paragraph</span>
This is a paragraph
This is a paragraph
This is a paragraph
This is a paragraph</p>
```


Renders as strong emphasized text.

Attributes None, other than common attributes.

Example

```
<strong>Strong text</strong>
```

<table>

Defines a table.

Attributes The following attributes are optional:

Name	Value	Description
summary	text	Summary of table contents
width	number	Width of the table in pixels

Example

```
<table summary="marine mammals" width="200">
....
</table>
```

<td>

Defines a cell in a table.

Attributes The following attributes are optional:

Name	Value	Description
abbr	text	Abbreviated version of the content in the cell.
align	left right center justify char	Horizontal alignment of cell content.
axis	category_names	A comma-separated list of category names. The list categorizes cells.
char	character	Sets which character is to be used to center the text around, when you set `align="char"`.
colspan	number	Number of columns the cell must cover. If the value is 0, the cell starts at the current column and spans to the last column in the table.
headers	header_cells_id	A space-separated list that specifies the header cells that apply to <td>. This attribute allows text-only browsers to render the header information for a given cell.
rowspan	number	Number of rows the cell must cover. If the value is 0, the row spans from this row to the last row in the table.

Name	Value	Description
scope	col colgroup row rowgroup	Specifies the cells for which the element provides header information. This attribute is a simple alternative to using headers.
valign	top middle bottom baseline	Vertical alignment of cell contents.

Example

```
<table border = "1">
<tr>
<td>Cell A</td>
<td>Cell B</td>
</tr>
</table>
```

<textarea>

Defines a text area (a multi-line text input control, written as “textarea”). Users can write text in the text area, using an unlimited number of characters. The default font in the textarea is fixed pitch.

Attributes

The following attributes are required:

Name	Value	Description
cols	number	The width of the textarea, in characters.
rows	number	The height of the textarea, in rows.

The following attributes are optional:

Name	Value	Description
name	textarea_name	Sets the name of the textarea.

Example

```
<textarea rows="2" cols="20">
The cat was playing in the garden. Suddenly a dog showed up.....
</textarea>
```

<th>

Defines a header in a table.

Attributes

The following attributes are optional:

Name	Value	Description
abbr	text	Abbreviated version of the content in the cell.

Name	Value	Description
align	left right center justify char	Horizontal alignment of cell content.
axis	category_names	A comma-separated list of category names. The list categorizes cells.
char	character	Sets which character is to be used to center the text around, when you set `align="char"`.
colspan	number	Number of columns the cell must cover. If the value is 0, the cell starts at the current column and spans to the last column in the table.
headers	header_cells_id	A space-separated list that specifies the header cells that apply to <th>. This attribute allows text-only browsers to render the header information for a given cell.
rowspan	number	Number of rows the cell must cover. If the value is 0, the row spans from this row to the last row in the table.
scope	col colgroup row rowgroup	Specifies the cells for which the element provides header information. This attribute is a simple alternative to using headers.
valign	top middle bottom baseline	Vertical alignment of cell contents.

Example

```
<table border = "1">
<tr>
<th>Header 1</th>
<th>Header 2</th>
</tr>
<tr>
<td>Cell A</td>
<td>Cell B</td>
</tr>
</table>
```

<title>

Defines a title for the document.

Attributes None, other than common attributes.

Example

```
<title>My wireless page</title>
```

<tr>

Defines a row in a table.

Attributes The following attributes are optional:

Name	Value	Description
align	left right center justify char	Horizontal alignment of cell contents.
valign	top middle bottom baseline	Vertical alignment of cell contents.

Example

```
<table border = "1">
<tr>
<td>Cell A</td>
<td>Cell B</td>
</tr>
</table>
```


Defines the start of an unordered list.

Attributes None, other than common attributes.

Example

```
<ul>
<li>Coffee</li>
<li>Tea</li>
</ul>
```

<var>

Defines a variable.

Attributes None, other than common attributes.

Example

```
<var>Variable</var>
```

Glossary

3G
Third generation wireless. This term refers to future developments in wireless technology. 3G is expected to reach maturity in 2003-2005, and to include such things as enhanced multimedia, usability on a variety of modes (cellphones, email, paging, fax, Web browsing, and so on), high bandwidth, routing flexibility, and increased roaming capability.

ALI
Automatic Location Identification is a technology that makes it possible to automatically display information about the geographical location (for example, a street address) of a cellphone call.

API
Application Programming Interface, a set of functions (or methods in Java) for building software applications. For example, Enhydra provides an API for building web applications. In general, programs use APIs to make requests of the operating system or another application.

bandwidth
Measurement of the transmission capacity, or size, of a communication channel. In general terms, a greater bandwidth indicates the ability to transmit a greater amount of data over a given period of time.

Bluetooth
An protocol for wireless communication of voice and data within a ten-meter range. It is intended to allow mobile devices to share information and applications without cables or interface incompatibilities—for example, it enables PCs to connect wirelessly to phones or other wireless devices.

Bps
Bits per second. Measures how quickly data can be moved from one place to another; that is, the number of pieces of information (bits) transmitted or received per second.

broadband
Telecommunication that provides multiple channels of data over a single communications medium, typically using some form of frequency or wave division multiplexing.

CDMA
Code Division Multiple Access, a spread spectrum wireless technology in which a signal is spread across multiple frequencies using a pseudo-random code. Alternative technologies are GSM and TDMA.

cHTML
Compact HTML, the markup language used in i-mode. It is a subset of HTML.

circuit switched network
A type of network, such as the regular voice telephone network, in which a physical path is obtained for and dedicated to a single connection between two end-points in the network for the duration of the connection.

CLDC
The Connected Limited Device Configuration, part of Sun's J2ME specification that outlines the most basic set of libraries and Java virtual machine features that must be present in each implementation of a J2ME environment on consumer devices.

DOM
Document Object Model, a programmatic object structure based on a document's structure and content. The World Wide Web Consortium (W3C) is developing a standard DOM API to let programs create and modify HTML and XML documents as full-fledged program objects.

DPN
Dynamic proxy navigation, a technology that provides end-to-end encryption and is included in the proposed WAP 2.0 standard.

DTD
Document Type Definition, A DTD is a specification that defines an XML document type. Every dialect (or application) of XML, such as WML or VoiceXML has a corresponding DTD.

DTMF
Dual-tone multi-frequency (also known as touch-tone), the audible sounds you hear when you press keys on a telephone.

EDGE
Enhanced Data Rates for Global Evolution, a more powerful version of GPRS, permitting data transmission speeds of up to 384 kilobits per second (Kbps). It is an evolution of the GSM standard projected to be available from GSM operators in 2002.

element
The fundamental logical unit of an XML (or HTML) document. All content in XML documents must be contained within elements. The term tag is sometimes used as a synonym (especially in HTML).

EPOC
An operating system developed by the Symbian consortium for a range of wireless devices. It includes customizable user interfaces, color support, fit-for-purpose

application suites, Internet connectivity, software development kits and PC connectivity software.

FCC
Federal Communications Commission, a regulatory body that governs communications technologies in the U.S.

FDMA
Frequency Division Multiple Access is a 30-channel portion of the frequency band allocated for wireless cellphone communication. Each channel can be assigned to only one user at a time and can carry either a voice conversation or, with digital service, digital data.

Gateway
Generically, an intermediary system connecting two different types of networks, enabling clients on one network to communicate with servers on another. For example, a WAP gateway enables a WAP cellphone to communicate with a server connected to the Internet.

GPRS
General Packet Radio Service, a radio technology for GSM networks that adds packet-switching protocols, and enables providers to charge by amount of data sent rather than connection time. GPRS promises to support flexible data rates up to 30 Kbps (with a theoretical maximum of 171.2 Kbps), and continuous network connections.

GPS
Global Positioning System. A collection of satellites that orbit the Earth and make it possible for people with ground receivers to pinpoint their geographic location.

GSM
Global System for Mobile communications, a digital cellular phone system based on TDMA, used throughout Europe and in much of the U.S. Alternative technologies are CDMA and TDMA.

HSCSD
High Speed Circuit Switched Data, an upgrade to GSM networks that enables data rates up to 57.6Kbps. HSCSD was introduced in 1999 to upgrade the GSM data rate from 14.4Kbps.

HDML
The predecessor of WML, created by Phone.com. In the U.S., most existing Internet-capable cellphones support HDML. However, many WAP gateways automatically convert WML content to HDML when necessary.

HTML
Hypertext Markup Language, used to format and display content on the World-Wide Web.

IMT-2000
International Mobile Telecommunications 2000 is the International Telecommunications Union (ITU) concept of a family of technological solutions that will enable the implementation of third-generation wireless communications.

i-mode
Wireless service launched in Japan in 1999 by NTT DoCoMo, accessed by a wireless packet network. Uses cHTML as the presentation markup language.

ITU
International Telecommunication Union, the international organization within which governments and corporations coordinate telecommunications networks and services.

J2ME
Java 2 Micro Edition, the version of Java for consumer devices.

Kbps
Kilobits per second, a unit measure of communications data rate equivalent to one thousand (1,000) bits per second. Some sources define a kilobit to be 1,024 (2^{10}) bits. However, bits in data communications are discrete signal pulses and historically have been counted using the decimal number system.

LBS
Location Based Services, end-user applications and content based on location. services.

LIF
Location Inter-operability Forum, a forum created by Motorola, Nokia, and Ericsson to develop and promote LBS technology through global standards bodies.

location services
Emerging technologies that provide ALI to cellphones and other wireless devices. A location is specified by latitude and longitude, although other conventions such as zip code and area code are sometimes used.

KVM
Kilobyte Virtual Machine, a scaled down Java virtual machine for consumer devices. See also: J2ME, MIDP.

Mbps
Megabits per second, a unit measure of communications data rate equivalent to one million (1,000,000) bits per second. Some sources define a megabit to be 1,048,576 (2^{20}) bits. However, bits in data communications are discrete signal pulses and historically have been counted using the decimal number system.

microbrowser
Software that renders Web content on a Web phone. It is analogous to a standard Web browser on a PC.

MIDP
Mobile Information Device Profile, a set of Java APIs which, together with the Connected Limited Device Configuration (CLDC), provides a complete J2ME runtime environment for mobile information devices. The MIDP specification addresses issues such as user interface, persistent storage, networking, and application model.

MIME type
Multipurpose Internet Mail Extension type. It provides a standard way to classify file types on the Internet. Internet programs such as Web servers and browsers have a list of MIME types, so that they can transfer files of the same type in the same way, no matter what operating system they are working in.

packet switched network
A type of network in which relatively small units of data (called packets) are routed through a network based on the destination address contained within each packet. This allows the same data path to be shared among many users in the network.

Palm OS
The operating system used by Palm and HandSpring PDAs. By far the most popular PDA operating system.

PCS
Personal Communications Service, a wireless phone service similar to cellphone service but emphasizing personal service and extended mobility.

PDA
Personal Digital Assistant, a compact handheld computer, generally used for personal organization, contact management, and calendar tasks. Many PDAs now come with wireless Internet access capability.

PDC
Personal Digital Cellular, a Japanese standard for digital mobile telephony in the 800 MHz and 1500 MHz bands.

PDC-P
Personal Digital Cellular-Packet. A packet-switched method of data transmission over the existing PDC network. i-mode uses this method.

Persona
A category or description of application user. A persona may be very detailed (for example, “spanish-speaking executives using NeoPoint cellphones”) or general (for example, “users with WAP devices”).

Pocket PC
Microsoft’s new name for Windows CE, a scaled-down version of Windows for mobile devices.

Profile
A user interface (or “skin”) targeted to a particular persona. For example, an application might have a WAP profile for users who have WAP devices.

SIM
Subscriber Identity Module. A SIM card is an advanced "smart card" with an embedded microchip that contains data, such as a subscriber’s phone number, service details, and memory for storing messages. With a SIM card, calls can be made from any valid mobile phone because the subscriber data, not the telephone's internal serial number, is used to make a call. This approach enables SIM cards to work in many different countries and networks.

SMS
Short Message Service, a “store and forward” text messaging system for cellphones popular in Europe.

Symbian
A consortium established by Ericsson, Motorola, Nokia, and Psion to develop an operating system for handheld devices.

TDMA
Time Division Multiple Access, a technology used in digital cellphone communication. It divides each cellular channel up into three time slots to increase the amount of data that can be carried. Alternative technologies are CDMA and GSM.

two-way pager
A pager that allows you to both send and receive data. For example, two-way pagers allow users to send and receive email and connect to the Web.

Universal Mobile Telecommunications System (UMTS)
A third generation (3G) technology that promises bandwidth of up to 2 megabits per second (Mbps), enabling transmission of video and other media-rich content. Initially

bandwidth may be significantly lower, due to network capacity restrictions. It is expected to become commercially available in Europe in 2003.

URL
Uniform Resource Locator, the address by which you locate content on the World-Wide Web. For example, http://www.lutris.com is a URL.

voice portals
Web services that deliver information by audio rather than visually. Designed to circumvent the often-tiny screens on mobile devices, voice portals use speech recognition and text-to-speech technology to deliver a variety of Web content, such as news, email, sports scores, stock quotes, and voice mail.

VoiceXML
A markup language used for creating voice dialogs with which a user can interact through speech recognition and voice synthesis.

W3C
Worldwide Web Consortium, the Web's primary standard-setting body. It is an international industry consortium that seeks to promote standards for the evolution of the Web and interoperability between Web products by producing specifications and reference software.

WAE
Wireless Application Environment. The part of WAP that specifies a general-purpose application environment based on Web technologies and philosophies.

WAP
Wireless Application Protocol, a wireless Internet standard championed by the WAP Forum.

WAP Forum
Industry consortium that promotes WAP, including Motorola, Nokia, Ericsson, Phone.com, and over 200 other companies.

WAP gateway
A computer system that connects an WAP network to the Internet.

WCDMA
Wideband Code Division Multiple Access, the third-generation (3G) standard offered to the International Telecommunication Union by GSM proponents.

Web phone
A broad category of devices, generally encompassing digital phones that can connect to the Internet through a wireless device.

Windows CE
A scaled-down version of Microsoft Windows for consumer devices, such as PDAs. Now called Pocket PC or Handheld PC.

WML
Wireless Markup Language, the markup language defined by WAP, for use on cellphones and other mobile devices.

WMLScript
The scripting language defined by WAP for use on cellphones and other mobile devices.

XHTML
Extensible HyperText Markup Language, a reformulation of HTML as an XML application. Although nearly identical to HTML, XHTML has more rules and is "cleaner" because it conforms with XML.

XHTML Basic
A version of XHTML suitable for wireless presentations, containing a subset of the elements in the complete XHTML language.

XML
Extensible Markup Language, a data format for structured document interchange that allows tags to be defined by the developer (unlike HTML, which has predefined tags).

XSL
eXtensible Stylesheet Language, a language for creating a stylesheet that describes how data sent over the Web using XML is to be presented to the user.

XSLT
XSL Transformations, designed for use as part of XSL, is a language for transforming (changing) XML documents from one type to another.

Index

Symbols

&xx; tag 148
/ (forward slash) in paths 3
<> (angle brackets) 66
\ (backslashes) in paths 3

Numerics

3G technology 25

A

<A> tag 148
<a> tag 130, 195
<abbr> tag 197
abbreviations 61, 197
<access> tag 131
access controls, specifying for WML 131
access protocols 19, 21
accessing
  data 93
accesskey attribute 195
<acronym> tag 197
acronyms 197
actions 83
  defining vocabulary for 176, 188
  specifying 174
activating VoiceXML dialogs 80, 189
<address> tag 197
addresses 197
Advanced Mobile Phone Service (AMPS) 17
AirSent J2ME client application 44
AirSent sample application 105
  building 36
  changing 46
  overview 35
  prerequisites for running 35
  rebuilding 47
  running 39
AirSent Web pages 40
algorithm, form interpretation (FIA) 187
AMPS standards 17
analog technologies 17
<anchor> tag 131
  alternate for 130, 195
angle brackets 66
application development tools 34, 89
application server. *See* Enhydra Application Server
Application wizard 90
  creating wireless applications with 49, 53
applications
  designing wireless 57, 62
  developing wireless 33, 99, 105
  options for wireless 13
  support for large scale 7
  support for voice 16
  testing 93
archives (Enhydra.org mailing list) 7
<assign> tag 164
attributes 66
  specific to XHTML 85, 130, 195
<audio> tag 164
audio files 164
  prompting for prerecorded output 184
  recording user samples for 186
  replaying 187
authorship 197

B

<b> tag 132
backslashes (\) in paths 3
bandwidth 17, 18
  conserving 102
<BASE> tag 148
<base> tag 197
bearer networks 16, 24
<big> tag 132
<block> tag 165
<blockquote> tag 198
Bluetooth standard 11
<BODY> tag 149
<body> tag 198
bold font attribute
  WML formats 132, 143
  XHTML formats 210
Borland JBuilder. *See* JBuilder

 tag 149

 tag 132, 199
<break> tag 165
broadband technologies 25
browsers 14, 97
  disabling caching 102
  displaying special character sequences 85, 148
  tasks defined for micro 70
bug reports 5
building
  wireless applications 52, 55
  wireless sample application 36
builds 91

bulleted lists
cHTML formats 150, 157, 161
XHTML formats 204, 214

C

cache 102, 185
calendars 12
<caption> tag 199
captions 199
<card> tag 132
cards. *See* WML decks
case sensitivity, of XHTML elements 84
<catch> tag 166
CDMA technology 18
cellphones 9, 18
analog technologies for 17
automatic location information for 23
digital technologies for 17
markup language support for 19
Web-enabled 13
<CENTER> tag 149
centering text 149
changing
defaults 94
sample application 46
character encoding 103
character sequences 85, 148
characters (reserved) 73
<choice> tag 167
cHTML 22, 27, 74
cHTML pages 97
cHTML tags 74, 147
HTML 1.0 elements in 75
HTML 2.0 elements in 77
circuit-switched networks 17, 18
citations 199
<cite> tag 199
class attribute, XHTML tags 130, 195
classes 92, 93
<clear> tag 168
client applications, running AirSent J2ME sample 44
client presentation technologies 26
<code> tag 199
code 92
Code Division Multiple Access (CDMA) 18
code font type
cHTML formats 159
XHTML formats 199, 210
command-line options 99
compact HTML. *See* cHTML
compiler 92
Compiler wizard 90
computers 11
conditional statements 170, 178
configuration files 90
connection modes 96
connections 11
wireless application protocol for 19
constants 163
contacting
Lutris Documentation 4
Lutris Technical Support 5
Lutris Training 6
controls, labeling 204
conventions 1, 3
converting text to speech 164
converting XML documents 117, 118
cookies 101
core development tools 34
creating
card templates 144
list controls 142
presentation layers 93
VoiceXML documents 193
wireless applications 49, 53
WML tables 143
XHTML tables 88
customizing XML formats 118

D

data
wireless applications and 126
data access 93
Data Object Design Studio. *See* DODS
data services 25
datagrams 17, 20
data-layer classes 93
<DD> tag 150
<dd> tag 200
decks. *See* WML decks
defaults 94
definitions
cHTML formats 150
XHTML formats 199, 200
designing wireless applications 57, 62
desktop computers 11
developing wireless applications 33, 99, 105
development tools 34, 89
device emulators 93
devices 9, 11
constraints for wireless 60
detecting type 101
<dfn> tag 199
dialogs 79, 80
activating 189

exiting from 187
passing values to 183
digital phones 13
digital technologies 17
<DIR> tag 150
directives 83
specifying 165
directory paths 3
disabling caching 102
<disconnect> tag 168
displaying
cHTML pages 97
HTML pages 97
special character sequences 85, 148
wireless content 41
WML pages 94
<DIV> tag 151
 tag
voiceXML formats 168
XHTML formats 200
<DL> tag 151
<dl> tag 200
DLLs 96
<do> tag 133
<!DOCTYPE> tag 147
document conversions 117, 118
Document Object Models. *See* DOMs
Document Type Definitions 66, 68
example for 67
documentation 4
conventions used in 1
updates and release notes 4
documentation set 3
DODS 93
–domfactory option 99
DOML files 93
DOMs 90
specific to XHTML documents 101
wireless applications 99, 100
downloading
Enhydra open-source software 8
multiple screens 68
<DT> tag 151
<dt> tag 200
DTDs 66, 68
example for 67
DTMF 80
<dtmf> tag 169
dual-tone multi-frequency grammar 80
dynamic proxy navigation (DPN) 21
dynamic-link libraries 96

E

EasyPad Waptor 98
editing WML documents 98
elements 66, 68, 75
See also tags
<else> tag 170
<elseif> tag 170
 tag 134, 201
embedded objects 182, 183, 207
emoji (i-mode icon symbols) 74
<emp> tag 170
emphasis typestyles
VoiceXML formats 170
WML documents 134, 136
XHTML formats 201
emulators 93
encryption 20
end clients 11
end-user interface 11
engines 92
Enhydra
accessing source code for 7
documentation set 3
downloading open-source software 8
mailing lists 7
online documentation 4
product registration 5
reporting bugs 5
support for large scale applications 7
technical support 5
training courses 6
website (Enhydra.org) 6
working groups (Enhydra.org) 7
Enhydra Application Server 1
Enhydra applications 1
See also applications
accessing data 93
presentation layers for 93
Enhydra development environment 1
Enhydra tools 89
Enhydra.org 6
community documentation 8
mailing list archives 7
mailing lists 7
working groups 7
entities 66
<enumerate> tag 171
EPOC systems 13
<error> tag 172
error events 172

events
binding to tasks 139
catching VoiceXML 166, 172, 177, 181
throwing VoiceXML 191
voice-activated applications 80, 83
WML documents 70
<exit> tag 172
expressions 192
eXtensible Markup Language. *See* XML
Extensible Stylesheet Language Transformations 117
XML compiler and 118
external resources 197

F

FDMA technology 17
FIA (form interpretation algorithm) 187
<field> tag 173
fields
adding input to VoiceXML 173
adding input to XML documents 203
adding WML option input 140
grouping related WML 134
grouping WML option input 139
<fieldset> tag 134
file types 103
files 93
<filled> tag 174
firing events 80
fixed-width fonts
cHTML formats 159
XHTML formats 199, 210
font attributes
VoiceXML formats 170
WML documents 132, 134, 136, 143, 146
XHTML formats 201, 210
fonts
monospaced 159, 199, 210
specifying large 132
specifying small 143
<FORM> tag 152
<form> tag
VoiceXML formats 175
XHTML formats 201
form interpretation algorithm (FIA) 187
formatting 100
forms
cHTML input 152
i-mode devices 76
mixed-initiative 178
voice-activated applications 79, 80, 82, 175
XHTML input 88, 201
Forte for Java 91
forward slash (/) in paths 3
Frequency Division Multiple Access (FDMA) 17
functions. *See* methods

G

gateways 19
General Packet Radio Service (GPRS) 25
getHeader() method 101
Global Positioning System (GPS) 23
Global System for Mobile Communications (GSM) 18
<go> tag 135
<goto> tag 176
GPRS (General Packet Radio Service) 25
GPS (Global Positioning System) 23
<grammar> tag 176
grammars 80
specifying 169, 176
graphical user interfaces. *See* GUIs
graphics files. *See* images
grouping
related WML fields 134
WML input fields 139
GSM networks 18, 25
GUIs
defining WML elements for 133
voice-activated applications 78
wireless applications 57, 61, 72
wireless devices and 12

H

handheld computers 9, 12
Handheld Device Markup Language (HDML) 19
HDML (Handheld Device Markup Language) 19
<HEAD> tag 152
<head> tag 136, 202
header information 152
See also meta-information
getting 101
header tags
cHTML formats 153
XHTML formats 202
headers
adding to XML tables 212
defining for WML decks 136
headings 202
"Hello World" example 68, 79
<help> tag 177
help events 177
horizontal rules 153
<HR> tag 153
<HTML> tag 154

HTML 26
cHTML vs. 74
integrating with wireless applications 147, 152, 154
integrating with XHTML 84, 202
support for wireless devices 27, 52
support for XML-enabled devices 27
<html> tag 202
HTML pages 97
HTTP 20
getting request headers for 102
setting MIME type for 100
HTTP conversions 19
HttpPresentationComms object 101
HTTPS 22
HttpServletRequest object 102
hyperlink history stack, moving through 141
hyperlinks
defining for wireless devices 59, 148
defining WML 71, 131
i-mode devices 77
voice-activated applications 81, 179
XHTML documents 87
hypertext markup languages
conversions for 117, 118
i-mode devices 75, 77
support for microbrowsers 14
wireless devices 22, 26, 65
Hypertext Transfer Protocol 20

I

<i> tag 136
icons 74
id attribute 100
XHTML tags 85, 130, 195
iDEN SDK 96
<if> tag 178
images
adding to wireless applications 63, 64, 154
specifying for WML documents 136
specifying for XHTML documents 88
specifying for XML documents 202
<IMG> tag 154
<img> tag 136, 202
i-mode icons 74
i-mode wireless service 22
character encoding 103
limitations 63
markup language for 75, 77
indented text formats 198, 209
informational resources
XSLT conversions 118
<initial> tag 178
<INPUT> tag 155
<input> tag 137, 203
input 92, 184
input fields
adding to Voice XML 173
adding to WML documents 140
adding to XML documents 203
grouping WML 139
input forms
cHTML formats 152
VoiceXML formats 175
XHTML formats 88, 201
interface elements. *See* UI elements
Internet
accessing with wireless devices 19, 21, 33
technology 9
italic font attribute
VoiceXML formats 170
WML formats 134, 136
XHTML formats 201

J

J2ME devices 15, 28
J2ME programming techniques 104
Java 2 platforms 91
Java APIs 29
Java classes 92
Java Database Connectivity. *See* JDBC
Java devices 104
Java phones 15
JavaScript 189
JBuilder
creating wireless applications with 49
overview 89
JBuilder Foundation. *See* JBuilder

K

<kbd> tag 204
Kelp sample project 91
Kelp tools
creating wireless applications with 49
overview 90
key presses 169
keyboard text 204

L

<label> tag 204
labels, defining 204
laptop computers 11
large fonts 132
large scale applications 7
LBS. *See* location services

<LI> tag 157
 tag 204
line breaks
  cHTML formats 149
  WML formats 132
  XHTML formats 199
<link> tag
  VoiceXML formats 179
  XHTML formats 205
link history stack, moving through 141
links
  defining for wireless devices 59, 148
  defining WML 71, 131
  i-mode devices 77
  voice-activated applications 81, 179
  XHTML documents 87
lists
  adding items 157, 204
  creating for wireless applications 150, 158, 161
  creating for WML documents 142
  creating for XML documents 208, 214
  creating pull-down 160, 210
localization 58
location services 23, 58
Lutris documentation 4
Lutris technical support 5

M

mailing lists (Enhydra.org) 7
mailing lists Enhydra.org)
  archives 7
marking acronyms 197
markup languages
  conversions for 117, 118
  i-mode devices 75, 77
  support for microbrowsers 14
  wireless devices 22, 26, 65
<MENU> tag 157
<menu> tag 180
menus
  voice-activated applications 79, 82, 180
  wireless applications 157, 160
message centers 24
messages 24
<META> tag 157
<meta> tag
  VoiceXML formats 180
  WML formats 138
  XHTML formats 206
meta-information
  VoiceXML applications 180
  wireless applications 157
  WML decks 69, 138
  XML applications 206
microbrowsers 14, 97
  disabling caching 102
  tasks defined for 70
MIDP 29, 104
MIME types 100
  setting 103
mixed-initiative forms 178
mobile devices 9, 24
Mobile Information Device Profile 29, 104
monospaced fonts
  cHTML formats 159
  XHTML formats 199, 210
multi-frequency grammar 80
multi-line text input controls 160, 212
Multipurpose Internet Mail Extension. *See* MIME
multitasking operating systems 13

N

Navi Roller 126
Navi Rollers 94
nested tags 66, 84
networks
  dedicated frequency bands for 19
  encryption services for 20
  global system for 18
  wireless systems and 13, 15, 16, 24
new lines
  cHTML formats 149
  WML formats 132
  XHTML formats 199
No Operation setting 138
<noinput> tag 181
Nokia 1.2 Toolkit 127
Nokia WAP Toolkit 95
<nomatch> tag 181
nomatch events 181
<noop> tag 138
notebooks 11
notifications 96
numbered lists
  cHTML tags for 157, 158
  XHTML tags for 204, 208

O

<object> tag
  VoiceXML formats 182
  XHTML formats 207
objects
  defining runtime settings for 209
  embedding in VoiceXML documents 182, 183
  embedding in XML documents 207

<OL> tag 158
 tag 208
onenterbackward event 70
onenterforward event 70
<onevent> tag 139
online
  documentation 3, 4
  registration 5
online documentation
  conventions used in 1
onpick event 70
ontimer event 70
operating systems 13
operators 164
<optgroup> tag 139
<OPTION> tag 158
<option> tag
  VoiceXML formats 183
  WML formats 140
  XHTML formats 208
option input fields
  adding to WML documents 140
  grouping 139
option list controls 142
ordered lists
  cHTML tags for 157, 158
  XHTML formats 204
  XHTML tags for 208
output 94
  pausing speech 165

P

<P> tag 159
<p> tag 140, 208
packets 17
packet-switched networks 17, 22
pagers 9, 15
Palm OS systems 12
palm-top computers 12
paragraphs
  cHMTL formats 159
  WML formats 140
  XHMTL formats 208
<param> tag
  VoiceXML formats 183
  XHTML formats 209
partitioning 59
path names 3
pausing speech output 165
PCs 11
PCS networks 19
PDAs 12
Personal Communication Service (PCS) 19
personal computers 11
personal digital assistants 12
personal organizers 12
personalization 58
phone functions 96
Pixo Internet Microbrowser 97
<PLAINTEXT> tag 159
playing audio files 164
Pocket PC systems 13
<postfield> tag 141
<PRE> tag 159
<pre> tag 209
preformatted text
  cHTML formats 159
  XHTML formats 209
presentation formats 101
presentation layers 93
  Wireless DiscRack application 127
presentation objects
  detecting device types for 101
  disabling caching for 103
  generating content 100
presentation technologies 26
<prev> tag 141
product registration 5
profiles 57, 60, 62
programming for wireless applications 99
projects 91
prologs 68
<prompt> tag 184
prompts 184, 187
  defining 192
properties 185
<property> tag 185
property pages 90
<pros> tag 186
prosodic information 186
pull-down lists
  creating 160, 210
  defining options in 208

Q

<q> tag 209
quotations 198, 209

R

<record> tag 186
recording audio samples 186
referencing external resources 197
<refresh> tag 141
registering Enhydra 5
release notes 4
replaying audio files 187

reporting bugs 5
<reprompt> tag 187
requests 102
reserved characters 73
resources 92
    referencing external 197
responses 100
<return> tag 187
root directory 3
rows
    creating for WML tables 145
    creating for XML tables 214
rules 153
running
    wireless applications 52, 55, 117
    Wireless DiscRack application 125
    wireless sample application 35, 39
runtime settings 209

S

<samp> tag 210
sample application 105
    building 36
    changing 46
    overview 35
    prerequisites for running 35
    rebuilding 47
    running 39
    Web pages for 40
sample project 91
<sayas> tag 188
screen shots 3
<script> tag 189
SDKs 96
Secure Sockets Layer 20
security 20
<SELECT> tag 160
<select> tag 142, 210
servers 1, 95
servlets 102
    disabling caching for 103
    setting content type 104
session identifiers 101
SessionEncodeUrlState argument 101
sessions 80, 101
setContentType method 104
setText method 100
setting MIME types 103
setting properties 185
<setvar> tag 143
shared voice channels 18
Short Message Service 24
short-range wireless systems 11
signatures 197
SIM cards 18
skins 33
<small> tag 143
small fonts 143
smartphones 14
SMS messages 24
Software Development Kit 96
source code 92
source code, accessing 8
 tag 100, 210
special character sequences 85, 148
speech recognition grammar 80
    specifying 169, 176
speech technologies 16
speech-activated applications. *See* voiceXML
spread spectrum 18
Sprint PCS NeoPoint 1000 127
SSL 20
storage units 66
storyboards 62, 117
 tag 143, 210
stylesheets 120
<subdialog> tag 189
subdialogs 80
    activating 189
    exiting from 187
    passing values to 183
<submit> tag 190
submitting bug reports 5
Subscriber Identity Module 18
subscribing to Enhydra mailing lists 7
support 5
synthesized speech 184, 187, 188
system events 191

T

tabindex attribute 195
<table> tag 143, 211
tables
    adding captions for 199
    adding cells 144, 211
    adding headers 212
    adding rows 145, 214
    creating WML 143
    creating XHTML 88
TACS standards 17
tag attributes 66
    specific to XHTML 85, 130, 195
tags 66
    case sensitivity for XHTML 84
    cHTML reference 147
    VoiceXML reference 163

WML reference 129
XHTML reference 195
targets 148
tasks 70
binding to events 139
<td> tag
WML formats 144
XHTML formats 211
TDMA technology 18
technical support 5
telephones 15
analog technologies for 17
digital technologies for 17
simulating wireless 94
terminating calls 168
transferring calls 191
<template> tag 144
template engines 92
template property pages 90
templates
creating card 144
voice-activated applications 171
terminating calls 168
testing applications 93
text 100
adding to VoiceXML documents 168
adding to WML documents 137
centering 149
converting to speech 164
defining as multi-line 160, 212
defining body 198
defining preformatted 159, 209
formatting for i-mode devices 76
formatting XHTML 87
indenting 198, 209
<TEXTAREA> tag 160
<textarea> tag 212
<th> tag 212
third-generation (3G) systems 25
<throw> tag 191
Time Division Multiple Access (TDMA) 18
timeouts 185
<timer> tag 145
timers 145
<TITLE> tag 161
<title> tag 213
title attribute 195
tools 34, 89
Total Access Communications System (TACS) 17
<tr> tag 145, 214
training courses 6
<transfer> tag 191
transferring calls 191
triggering events 80
two-way pagers 15
typestyles. *See* font attributes

U

<u> tag 146
UI elements
defining WML card-level 133
wireless applications 57, 61, 72
UI widgets 12
<UL> tag 161
 tag 214
UMTS 25
unary operators 164
underlined font attribute, WML documents 146
Uniform Resource Locators. *See* URLs
Universal Mobile Telecommunications System 25
UNIX directory paths 3
unordered lists
cHTML formats 150, 157, 161
XHTML formats 204, 214
updating user agents 141
URLs 101, 102
defining WML 141
documentation conventions for 3
moving to previous 141
navigating to 135
setting location for wireless 148
voice-activated applications 79
Wireless DiscRack application 125
user agent
setting 143
updating 141
user input forms
cHTML formats 152
VoiceXML formats 175
XHTML formats 88, 201
user interfaces. *See* GUIs; UI elements
user-defined events 191

V

<value> tag 192
<var> tag
VoiceXML formats 193
XHTML formats 214
variables
defining VoiceXML 83, 163, 164, 193
defining WML 73
defining XHTML 214
verbal applications. *See* voiceXML applications
viewing
cHTML pages 97
HTML pages 97
special character sequences 85, 148

wireless content 41
WML pages 94
voice channels 18
voice communications 16
Voice eXtensible Markup Language. *See* VoiceXML
voice Web 16
voice-activated applications. *See* voiceXML applications
VoiceXML 27, 78
VoiceXML applications 16, 78
activating dialogs 80, 189
adding menus 79, 82, 180
adding mixed-initiative forms 178
clearing forms 168
defining actions for 174, 176, 188
defining conditional statements for 170, 178
defining key presses for 169
defining prompts 192
defining user interfaces for 78
defining vocabulary for 176, 188
moving through 78, 80
pausing speech output 165
prompting for user input 184, 187
setting meta-information for 180
setting properties for 185
setting prosodic information for 186
setting user options 167, 171
shutting down 172
submitting values 190
terminating calls 168
user not responding 181
voiceXML applications 78
VoiceXML directives 83
specifying 165
VoiceXML documents 78, 82
adding user input forms 175
creating 193
defining input fields 173
embedding platform-specific objects 182, 183
moving through 176
placing divisions in 168
setting font attributes 170
VoiceXML events 80, 83
catching 166, 172, 177, 181
throwing 191
VoiceXML specification 163
VoiceXML tags 79, 163
example 79, 81
summarized 81
<vxml> tag 193

W

W3C VoiceXML specification 163
WAP applications 13
WAP gateways 19
WAP servers 95
WAP technology 19, 21
compared to i-mode 22
limitations 63
WAP Toolkit 95
WAPtor utility 98
WDP (Wireless Datagram Protocol) 20
Web browsers 14, 97
disabling caching 102
displaying special character sequences 85, 148
tasks defined for micro 70
Web clipping applications 13
Web pages
creating for wireless applications 52, 55
defining i-mode 75
displaying cHTML 97
displaying HTML 97
displaying in wireless applications 161
sample application 40
Web profiles 57, 60, 62
Web servers 1, 95
Web-enabled phones 13
website (Enhydra) 6
websites, and XMLC 93
widgets 12
Windows platforms
32-bit DLLs for 96
directory paths on 3
WML editor for 98
Wireless Application Protocol. *See* WAP
wireless applications 13
adding menus 157, 160
adding multi-line text to 160
building 52, 55
centering HTML content in 149
connecting to 11
creating 49, 53
creating lists for 150, 157, 158, 160, 161
data entry 126
defining option elements for 158
defining preformatted text in 159
defining presentation formats 101
defining URL locations 148
defining Web page to display for 161
designing 57, 62

developing 33, 99, 105
development tools for 34, 89
displaying character sequences in 148
enabling user input 152, 155
including images in 63, 64, 154
integrating HTML with 147, 152, 154
localizing 58
navigating 62
partitioning content in 59
placing divisions in 151
providing user interface for 57, 61, 72
references for 29
running 52, 55, 117
setting fonts 159
setting meta-information for 157
structuring documents in 153
Wireless Datagram Protocol (WDP) 20
wireless development tools 34
wireless devices 11
accessing Internet with 19, 21, 33
automatic location information for 23
constraints 60
detecting type 101
displaying output for 94
downloading multiple screens for 68
user interfaces for 12, 72
Wireless DiscRack application
presentation layer for 127
running 125
wireless DOMs 99
manipulating 100
wireless markets 10
Wireless Markup Language. *See* WML
wireless networks 13, 15, 16, 24
wireless presentation languages 65
wireless services 22, 23
wireless session protocol 20
wireless storyboards 62
wireless systems 9, 11
wireless technologies 10, 11, 16
limitations 63
Wireless Telephony Application Interface (WTAI) 72
wireless tools 93
wireless transaction protocol 20
Wireless Transport Layer Security 20
wizards 90
WML 19, 26, 68
<wml> tag 146
WML applications
testing 93
viewing pages 94
WML conversions 19
WML decks 68
creating 146
declaring card timers for 145
defining card-level UI elements 133
defining single card for 69, 132
header information for 136
moving through 71
setting meta-information for 69, 138
setting up card templates 144
WML documents
See also WML decks
adding list controls 142
adding option input fields 140
defining hyperlinks for 71, 131
defining text entry objects for 137
defining URLs for 141
editing 98
formatting elements for 72
grouping input fields 139
grouping related fields 134
including images in 136
setting font attributes 132, 134, 136, 143, 146
structuring 68, 129
WML editor 98
WML events 70
binding to tasks 139
WML reference 129
WML tables
adding cells 144
creating 143
defining rows 145
WML tags 68, 129
reserved characters for 73
WMLScript 27, 72
working groups
Enhydra.org 7
World Wide Web. *See* Web
writeDOM method 100, 101
WSP. *See* wireless session protocol
WTAI (Wireless Telephony Application Interface) 72
WTLS services 20
WTP. *See* wireless transaction protocol

X

XHTML 27, 83
XHTML applications, displaying character sequences in 85
XHTML documents 86
defining hyperlinks in 87
formatting text 87
including images in 88
XHTML DOMs 101

XHTML tables, creating 88
XHTML tags 195
basic elements 86
case sensitivity for 84
closing 84
XML 65
XML applications
defining addresses for 197
entering abbreviations for 197
marking acronyms 197
referencing external resources 197
setting meta-information for 206
voice-response systems and 16, 27
XML documents
adding headings 202
adding lists 204, 208, 210, 214
adding multi-line text to 212
adding titles 213
adding user input forms 201
converting to other types 117, 118
defining body text in 198
defining citations for 199
defining header for 202
defining input fields 203
defining preformatted text in 209
defining relationship between linked 205
defining sections in 210
embedding objects 207
including images in 202
indenting text in 198, 209
placing divisions in 200
providing definitions for 199, 200
setting font attributes 201, 210
XML entities 66
XML tables
adding cells 211
adding headers 212
creating 211
defining captions for 199
defining rows 214
XML tags 66
xml:lang attribute
XHTML formats 130, 195
XMLC
markup language conversions and 117, 118
overview 92
XMLC command-line options, wireless DOMs 99
XMLC Compiler wizard 90
XMLC property pages 90
XMLObject argument 100
XSLT 117
XML compiler and 118

Y

YoSpace SmartPhone Emulator 93