

A Guide to WebEnterprise

iPlanet™ Unified Development Server

Version 5.0

August 2001

Copyright (c) 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, U.S.A. All rights reserved.

Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at <http://www.sun.com/patents> and one or more additional patents or pending patent applications in the U.S. and in other countries.

This document and the product to which it pertains are distributed under licenses restricting their use, copying, distribution, and decompilation. No part of the product or of this document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any.

Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Sun, Sun Microsystems, the Sun logo, Forte, iPlanet, Unified Development Server, and the iPlanet logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Federal Acquisitions: Commercial Software - Government Users Subject to Standard License Terms and Conditions.

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright (c) 2001 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303, Etats-Unis. Tous droits réservés.

Sun Microsystems, Inc. a les droits de propriété intellectuels relatants à la technologie incorporée dans le produit qui est décrit dans ce document. En particulier, et sans la limitation, ces droits de propriété intellectuels peuvent inclure un ou plus des brevets américains énumérés à <http://www.sun.com/patents> et un ou les brevets plus supplémentaires ou les applications de brevet en attente dans les Etats - Unis et dans les autres pays.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a.

Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Sun, Sun Microsystems, le logo Sun, Forte, iPlanet, Unified Development Server, et le logo iPlanet sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays.

UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

LA DOCUMENTATION EST FOURNIE "EN L'ÉTAT" ET TOUTES AUTRES CONDITIONS, DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE CONTREFAÇON.

Contents

List of Figures	11
List of Procedures	13
List of Code Examples	15
Preface	17
Product Name Change	17
Audience for This Guide	18
Organization of This Guide	18
Text Conventions	19
Other Documentation Resources	20
iPlanet UDS Documentation	20
Express Documentation	21
WebEnterprise and WebEnterprise Designer Documentation	21
Online Help	21
iPlanet UDS Example Programs	22
Viewing and Searching PDF Files	22
Chapter 1 Overview	25
About HTML Support	26
Chapter 2 WebEnterprise and HTML	29
About iPlanet UDS and HTML	29
Terminology	32
fortecgi Program and the iPlanet UDS NSAPI Plug-in	33
Web Access Service Object	35
Page Factory Service Objects	38

Application Design Considerations	40
Structuring Your Web Application	40
Creating a Web Interface for a New iPlanet UDS Server	41
Adding a Web Interface to an iPlanet UDS Client Application	41
Sharing an iPlanet UDS Server	42
Scaling Your Web Application	43
Differences between Window- and Web-Based Applications	44
Performing Data Validation and Other Window Processing	45
Keeping State Information	47
Structuring the User Interface	48
Quick Tutorial: EasyWeb	50
Chapter 3 Setting Up a Web Application	57
Summary of Steps for Creating an iPlanet UDS Web Application	57
Suggested Project Hierarchy	59
When to Include the Optional HTML Projects	61
The HTML Project	61
The HTMLWindow Project	61
The HTMLSQL Project	62
Project Structure for the SoftWear Application	62
Defining a Web Access Project and Service	64
Creating a Web Access Service Object	64
Enabling the Web Access Service Object	65
Using a Start Method to Enable Access	67
Using an Administration Window to Enable Access	67
Disabling Web Access	69
Defining a Scanner Service Object	69
Defining a Web Page Builder Project	70
Defining a Page Builder Service Object	72
Defining a Shared Windows Project	73
Writing Methods for Shared Windows	74
Chapter 4 Planning Web Pages	75
Roles of the Web Author and Web Programmer	75
Initial Decisions About Pages	78
Static and Dynamic Web Pages	78
Using the iPlanet UDS HTML Projects to Create Pages	79
Determining which Services will Provide Pages	80
Identifying Session Management Requirements	82

Special Purpose Pages	83
Entry Point Page	83
Session Creation Page	85
Using Links	85
Format of URLs used by WebEnterprise	86
The \$\$FORTE.ExecURL Variable	87
Constructing Links	88
Using Images and Graphics	90
Error Handling	91
The Default Web Error Page	92
Error Handling Icons	92
Testing a Web Application	94
Chapter 5 Creating Pages Using Templates	97
About iPlanet UDS Templates	97
About the HTML Scanner Service	102
The iPlanet UDS HTML Tags	103
Tag Handlers	104
Purpose of the HandleTag and HandleCondition Methods	105
Result Sets and iPlanet UDS Variables	106
Summary of Steps for Using Templates	108
Designing a Template with iPlanet UDS Tags	108
Using HTML Editors	109
Putting iPlanet UDS Tags in a Template	110
Defining the HandleTemplateRequest Method	111
Creating TagHandlers	112
Using Subclasses of HTMLScanner	113
Using Custom Classes	114
Writing Tag Code	116
Defining the HandleTag Method	117
Constructing a Result Set	118
Using ITERATE to Add Tables and Lists	120
Defining the HandleCondition Method	123
Register or Load Tag Handlers	123
Choosing Static Registration or Dynamic Loading	124
Using Static Registration	124
Using Dynamic Loading	125
The Handler File	125
Testing a Template	127

Reference for iPlanet UDS HTML Tags	127
FORTE EXECUTE Tag	128
FORTE IF ... ELSE Tags	130
FORTE ITERATE Tag	131
FORTE INCLUDE Tag	133
FORTE REDIRECT Tag	134
Chapter 6 Creating Pages Using Page Builder Methods	135
Using a Page Builder Service	135
Page Builder Methods	136
Techniques for Writing Page Builder Methods	138
Using HTML Tag Markup Directly	139
Using HTML Classes	139
Using the WindowConverter Class	140
Using the SQLConverter Class	142
Saving Generated Pages to HTML Files	143
Defining the HandleRequest Method	143
A Sample HandleRequest Method	145
Adapting iPlanet UDS Windows with WindowConverter	146
Designing a Window for Use as a Web Page	148
Using the HTML Options... Command	149
Converting a Window	150
Sharing Window Code with the Web Page	151
Chapter 7 Using Session Management	153
The Benefits of Session Management	153
The Meaning of Session and State Management	155
Session Management Features	156
Session Properties for Web Pages	157
Web Session Manager	157
Session Objects and the Session Table	158
Session IDs	159
Validating and Tracking a Session	160
Typical Session Management Scenarios	163
About SESSION_REQUIRED	166
All Pages are Available to All Users	166
Different Pages are Available to Different Users	167
About SESSION_AUTOCREATE	169

Implementing Session Management	170
Initializing Session Management Attributes	171
Setting the Encrypt Key	171
Setting the Session Timeout Interval	171
Setting the SessionCreationURL	172
Enabling Session Management	173
Multiple Web Access Services Sharing Sessions	173
Optional Customizations	174
Deleting or Timing Out Sessions	174
Making Session IDs Persistent	175
Specifying a Non-Default Session Manager	175
Mixing Secure Sockets Layer (SSL) and non-SSL	175
Setting Session Properties for Pages	176
Setting a Default Session Property	176
The Session Property File	177
Overriding Session Properties for Individual Pages	178
Pages in Files and Directories	178
Page Builder Pages	178
Working with State Information	179
Defining What Constitutes State Information	180
Using a Subclass of HttpSession for State Information	180
Using Persistent Storage for State Information	181
Modifying URL Links for Session Management	182
Setting the DefaultCookie Attribute	182
Using the \$\$FORTE.ExecURL Variable in URLs	183
Session IDs in URLs	184
Alternate Ways to Manage State Information	185
Using Hidden Form Elements	185
Using Page Parameters in Generated URLs	185
Using Cookies	186
Chapter 8 Partitioning and Deployment	187
About Partitioning iPlanet UDS Web Applications	187
About Web Application Service Objects	188
Web Access Service Object	188
Page Builder Service Object and Scanner Service Object	189
Default Configuration for Web Applications	190

Modifying the Configuration	191
Creating a New Logical Partition	191
Modifying Service Object Definitions	191
Assigning Partitions	192
Moving Partitions	192
Replicating Partitions	192
Creating a Compiled Partition (Code Generation)	194
Deploying the Application	195
Chapter 9 Managing iPlanet UDS Web Applications	197
About iPlanet UDS CGI and iPlanet UDS Web Server Plug-in Programs	197
Choosing between fortectgi and the iPlanet UDS NSAPI Plug-In	199
Setup Options for fortectgi and iPlanet UDS Plug-ins	200
Autoregistration	201
Setting a Port for Autoregistration	201
Autoregistration Requires fortectgi Program	202
Manual Registration	202
Use of fortectgi.dat by iPlanet UDS NSAPI Plug-in	203
Using an iPlanet UDS Web Server Plug-in During Development	203
Maintaining the iPlanet UDS Web Site Files	205
The fortectgi Executable	205
The fortensapi DLL	206
Administrative Files	206
The fortectgi.dat File	206
The Session Property File	208
The Handler File	209
Template Files	210
The iPlanet UDS Document Root Directory	210
Graphic, Image, and Binary Data Files	211
Using an Administration Window	212
Initialization Tasks	212
Security Considerations	213
Using Basic Authentication	213
Using Secure Sockets Layer	214
Client Errors Reaching a Secure Server	215
Diagnosing Problems with fortectgi or a Plug-in	215
“Fortectgi Usage” Page	216
“iPlanet UDS NSAPI Plug-in Usage” Page	217
“Attempt to Authorize Web User”	217
“Not Found” Message	218
“Garbage Characters” on Screen	218

Troubleshooting Web Client Errors	218
ForteCGI Usage Page	219
Client Request Failure Errors	219
“No ServiceName parameter found in the request URL”	219
“Service Not Available”	219
“Service Not Found”	220
“The iPlanet UDS service you requested is busy, please try again”	220
fortecgi Runtime Errors	220
“Socket error:”	220
Client iPlanet UDS Errors	221
Object with NIL value returned by iPlanet UDS	221
Client Security Errors	221
“Status Code 401: Unauthorized...”	221
Troubleshooting Web Administrator Errors	222
“De-Registration Failure: Cannot write to fortectgi data file”	222
“De-registration Failure: fortectgi data file not found”	222
“Incoming registration or de-registration message is invalid”	222
“Registration Failure: Cannot create fortectgi data file”	222
“Registration Failure: Cannot write to fortectgi data file”	223
“Registration Failure: Duplicate iPlanet UDS server with the same port number”	223
Calling iPlanet UDS Technical Support	223
The WebEnterprise Release Number	224
Appendix A Example Programs	225
About the Examples	225
EasyWeb Example	225
SoftWear Example	225
The ShopCart Example	226
SQLDemo Example	227
Components of the SoftWear Example	227
Projects and .pex Files	227
iPlanet UDS Windows and Corresponding Web Forms	228
Page Builder Methods Used by SoftWear	229
The SoftWear Data Files	230
Installing the SoftWear Example	230
Components of the ShopCart Example	234
Projects and .pex Files	234
The ShopCart HTML Files	235
The ShopCart Image Files	235
Distribution of ShopCart Components	235
Installing and Running the ShopCart Example	235
Using the SQLDemo Example	240

Appendix B Environment Variables	245
Environment Variables	245
FORTE_CGI_REG_PORT	245
FORTE_CGI_REG_FILE	246
FORTE_WW_DOCUMENT_ROOT	247
FORTE_WW_HANDLER_CONFIG_FILE	247
Index	249

List of Figures

Figure 1-1	iPlanet UDS Internet Solutions	26
Figure 2-1	Using iPlanet UDS with the World Wide Web	30
Figure 2-2	The Web Server and the fortectgi Program	34
Figure 2-3	The fortectgi program and the Web Access Service Object	34
Figure 2-4	Using the Page Builder Service	36
Figure 2-5	Accessing Dynamically Created Web Page	37
Figure 2-6	Constructing Web Pages Dynamically	38
Figure 3-1	Suggested Project Hierarchy for an iPlanet UDS Web Application	60
Figure 3-2	Project Hierarchy for the SoftWear Application	63
Figure 3-3	Example Administration Window	68
Figure 4-1	Steps Involved in Building Web Pages	76
Figure 4-2	Entry Point Web Page for the SoftWear Application	84
Figure 4-3	Parts of a URL as Constructed and Parsed by Various Processes	86
Figure 4-4	Example of the Default Error Page	92
Figure 5-1	Relationship of Template and TOOL Method to Create a Web Page	98
Figure 5-2	Final Web Page Showing Generated Data (List of Products)	99
Figure 5-3	Example HTML Template Showing FORTE Tags	100
Figure 5-4	Role of the Scanner Service Object	102
Figure 5-5	Implementing Templates for Use with Scanner Service Objects	108
Figure 6-1	iPlanet UDS Client Window	141
Figure 6-2	Web Page Converted from iPlanet UDS Window	142
Figure 7-1	Validating and Tracking a Session	162
Figure 7-2	Steps for Implementing Session Management	170
Figure 9-1	Administration Window for the ShopCart Application	212
Figure 9-2	Web Page with Key	215
Figure 9-3	The fortectgi Usage Page	217
Figure 9-4	The iPlanet UDS NSAPI Plug-in Usage Page	217
Figure A-1	Administration Window for SoftWear Application	232

List of Procedures

- To copy the documentation to a client or server 22
- To view and search the documentation 23
- Create an entry point Web page for the EasyWeb application using HTML 50
- In iPlanet UDS, create a new Web project to define the Web access service object 51
- Include the iPlanet UDS HTTP library as a supplier plan for the new Web project 52
- In the EasyWeb project, create a new subclass of HTTPAccess 52
- Create a Web access service object with the new EasyAccess class as its type 52
- Override the HandleRequest method in the EasyAccess class 52
- Create a start-up class and method to start the iPlanet UDS application 54
- Register the Web access service object with the fortectgi program 54
- Test the application 55
- Exit the application 55
- To include a project as a supplier plan 60
- To create a Web access project 64
- To define a Web access service object 65
- To create the scanner service object 70
- To create a Web page builder project 71
- To create a page builder service object 72
- To create the shared windows project 73
- To add a link to a Web page 88
- To test a Web application locally 94
- To design a dynamic page using an iPlanet UDS HTML template 108
- To define the HandleTemplateRequest method 111
- To use a subclass of HTMLScanner for a tag handler class 113
- To use a custom class for a tag handler class 114
- To define a HandleTag method in a tag handler class 117
- To build a result set in a HandleTag method 118

To place a dynamically generated table in a Web page	120
To define the HandleRequest method	144
To convert an iPlanet UDS window (or widget) to an HTML document	146
To set the HTML options for a widget	149
The following steps describe how iPlanet UDS validates and tracks a session	161
To define default page-level session properties, and then override session properties as needed	164
To specify session properties using a session property file	165
To use WebEnterprise session management	170
To enable session management	173
To allow multiple Web access services to share sessions	173
To create your own session manager	175
To use a subclass of HttpSession to store session data	181
To update pages and templates to use the \$\$FORTE.ExecURL variable	183
To create a logical partition	191
To change the service object definition	192
To assign a logical partition	193
To set the Replication Count property for an assigned partition	193
To set the Compiled property for an assigned partition	194
To deploy your Web application	195
To autoregister a Web access service object	201
To manually register a Web access service object	202
To manually de-register a Web access service object	203
To use an iPlanet UDS Web server plug-in	204
To use SSL	214
To make sure that fortectgi or the iPlanet UDS plug-in is installed properly	216
To obtain your Web server name and version	223
To install and run the SoftWear example	230
To install the ShopCart example	236
To Run the ShopCart example	239
To install and run SQLDemo	240

List of Code Examples

URL that Requests a Page from a Page Builder Service	81
URL that Requests a Page from a Scanner Service	82
Sample Handler File	126
Sample Session Property File	177
A URL with an embedded Session ID	184

Preface

A Guide to WebEnterprise provides user and reference information about how to use HTML pages to create a Web interface to an iPlanet UDS application.

You can use WebEnterprise to build entirely new iPlanet UDS applications with Web interfaces, or to add Web connectivity to existing iPlanet UDS applications. This manual assumes that iPlanet UDS is already installed at your site.

For instructions on installing WebEnterprise and for information about certified platforms, versions, and products, see the *WebEnterprise Installation Guide*.

This preface contains the following sections:

- “Product Name Change” on page 17
- “Audience for This Guide” on page 18
- “Organization of This Guide” on page 18
- “Text Conventions” on page 19
- “Other Documentation Resources” on page 20
- “iPlanet UDS Example Programs” on page 22
- “Viewing and Searching PDF Files” on page 22

Product Name Change

Forte 4GL has been renamed the iPlanet Unified Development Server. You will see full references to this name, as well as the abbreviations iPlanet UDS and UDS.

Audience for This Guide

This manual is intended primarily for iPlanet UDS Web application programmers. We assume the reader is familiar with using the following:

- the iPlanet UDS application environment to create, partition, and deploy applications
- HTML to create Web pages
- the HTTP communication protocol

If you want to build a new iPlanet UDS application that includes Web connectivity, you might need to refer to other documents in the iPlanet UDS documentation set that are referenced in this manual. In addition, you might require access to a Web server for the installation of the `fortecgi` executable (and optionally the `fortensapi` executable) and to a Web browser for testing your Web client interface.

Some information in this book is intended for other persons involved in a Web site, specifically:

- Web page authors (such as text providers and graphic artists)
- Web site administrators (these persons are assumed to have some familiarity with both iPlanet UDS administration and Web site administration)

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter/Part	Contents
Chapter 1, "Overview"	A brief description of the purpose of WebEnterprise.
Chapter 2, "WebEnterprise and HTML"	A description of using WebEnterprise to implement Web browser interfaces to iPlanet UDS services.
Chapter 3, "Setting Up a Web Application"	Initial steps for setting up an iPlanet UDS Web application, including defining iPlanet UDS projects and service objects, and allowing Web browser access to the application.
Chapter 4, "Planning Web Pages"	A discussion of general issues about page design, including static and dynamic pages, special pages, links, and so on.

Chapter/Part	Contents
Chapter 5, “Creating Pages Using Templates”	Conceptual information about how templates work, and instructions for using templates to generate Web pages.
Chapter 6, “Creating Pages Using Page Builder Methods”	Conceptual information about using page builder methods, and instructions for writing page builder methods that use the HTML projects to generate Web pages.
Chapter 7, “Using Session Management”	Adding session management and managing state information to an iPlanet UDS Web application.
Chapter 8, “Partitioning and Deployment”	Partitioning and deploying your iPlanet UDS Web application.
Chapter 9, “Managing iPlanet UDS Web Applications”	Managing the files in an iPlanet UDS Web application, using <code>fortecgi</code> or the iPlanet UDS plug-ins, troubleshooting, and using security options.
Appendix A, “Example Programs”	Installing and running the sample applications provided with WebEnterprise.
Appendix B, “Environment Variables”	A description of the environment variables used by WebEnterprise.

Text Conventions

This section provides information about the conventions used in this document.

Format	Description
<i>italics</i>	Italicized text is used to designate a document title, for emphasis, or for a word or phrase being introduced.
<code>monospace</code>	Monospace text represents example code, commands that you enter on the command line, directory, file, or path names, error message text, class names, method names (including all elements in the signature), package names, reserved words, and URLs.
ALL CAPS	Text in all capitals represents environment variables (<code>FORTE_ROOT</code>) or acronyms (UDS, JSP, iMQ). Uppercase text can also represent a constant. Type uppercase text exactly as shown.

Format	Description
Key+Key	Simultaneous keystrokes are joined with a plus sign: Ctrl+A means press both keys simultaneously.
Key-Key	Consecutive keystrokes are joined with a hyphen: Esc-S means press the Esc key, release it, then press the S key.

Other Documentation Resources

In addition to this guide, there are additional documentation resources, which are listed in the following sections. The documentation for all iPlanet UDS products (including Express, WebEnterprise, and WebEnterprise Designer) can be found on the iPlanet UDS Documentation CD. Be sure to read [“Viewing and Searching PDF Files” on page 22](#) to learn how to view and search the documentation on the iPlanet UDS Documentation CD.

iPlanet UDS documentation can also be found online at <http://docs.iplanet.com/docs/manuals/uds.html>.

The titles of the iPlanet UDS documentation are listed in the following sections.

iPlanet UDS Documentation

- *A Guide to the iPlanet UDS Workshops*
- *Accessing Databases*
- *Building International Applications*
- *Esript and System Agent Reference Guide*
- *Fscript Reference Guide*
- *Getting Started With iPlanet UDS*
- *Integrating with External Systems*
- *iPlanet UDS Java Interoperability Guide*
- *iPlanet UDS Programming Guide*

- *iPlanet UDS System Installation Guide*
- *iPlanet UDS System Management Guide*
- *Programming with System Agents*
- *TOOL Reference Guide*
- *Using iPlanet UDS for OS/390*

Express Documentation

- *A Guide to Express*
- *Customizing Express Applications*
- *Express Installation Guide*

WebEnterprise and WebEnterprise Designer Documentation

- *A Guide to WebEnterprise*
- *Customizing WebEnterprise Designer Applications*
- *Getting Started with WebEnterprise Designer*
- *WebEnterprise Installation Guide*

Online Help

When you are using an iPlanet UDS development application, press the F1 key or use the Help menu to display online help. The help files are also available at the following location in your iPlanet UDS distribution:

```
FORTE_ROOT/userapp/forte/cln/*.hlp.
```

When you are using a script utility, such as Fscript or Escript, type help from the script shell for a description of all commands, or help *<command>* for help on a specific command.

iPlanet UDS Example Programs

A set of example programs is shipped with the iPlanet UDS product. The examples are located in subdirectories under `$FORTE_ROOT/install/examples`. The files containing the examples have a `.pex` suffix. You can search for TOOL commands or anything of special interest with operating system commands. The `.pex` files are text files, so it is safe to edit them, though you should only change private copies of the files.

Viewing and Searching PDF Files

You can view and search iPlanet UDS documentation PDF files directly from the documentation CD-ROM, store them locally on your computer, or store them on a server for multiuser network access.

NOTE You need Acrobat Reader 4.0+ to view and print the files. Acrobat Reader with Search is recommended and is available as a free download from <http://www.adobe.com>. If you do not use Acrobat Reader with Search, you can only view and print files; you cannot search across the collection of files.

➤ **To copy the documentation to a client or server**

1. Copy the `doc` directory and its contents from the CD-ROM to the client or server hard disk.

You can specify any convenient location for the `doc` directory; the location is not dependent on the iPlanet UDS distribution.

2. Set up a directory structure that keeps the `udsdoc.pdf` and the `uds` directory in the same relative location.

The directory structure must be preserved to use the Acrobat search feature.

NOTE To uninstall the documentation, delete the `doc` directory.

► **To view and search the documentation**

1. Open the file `udsdoc.pdf`, located in the `doc` directory.
2. Click the Search button at the bottom of the page or select `Edit > Search > Query`.
3. Enter the word or text string you are looking for in the Find Results Containing Text field of the Adobe Acrobat Search dialog box, and click Search.

A Search Results window displays the documents that contain the desired text. If more than one document from the collection contains the desired text, they are ranked for relevancy.

NOTE For details on how to expand or limit a search query using wild-card characters and operators, see the Adobe Acrobat Help.

4. Click the document title with the highest relevance (usually the first one in the list or with a solid-filled icon) to display the document.

All occurrences of the word or phrase on a page are highlighted.

5. Click the buttons on the Acrobat Reader toolbar or use shortcut keys to navigate through the search results, as shown in the following table:

Toolbar Button	Keyboard Command
Next Highlight	Ctrl+]]
Previous Highlight	Ctrl+[[
Next Document	Ctrl+Shift+]]

To return to the `udsdoc.pdf` file, click the Homepage bookmark at the top of the bookmarks list.

6. To revisit the query results, click the Results button at the bottom of the `udsdoc.pdf` home page or select `Edit > Search > Results`.

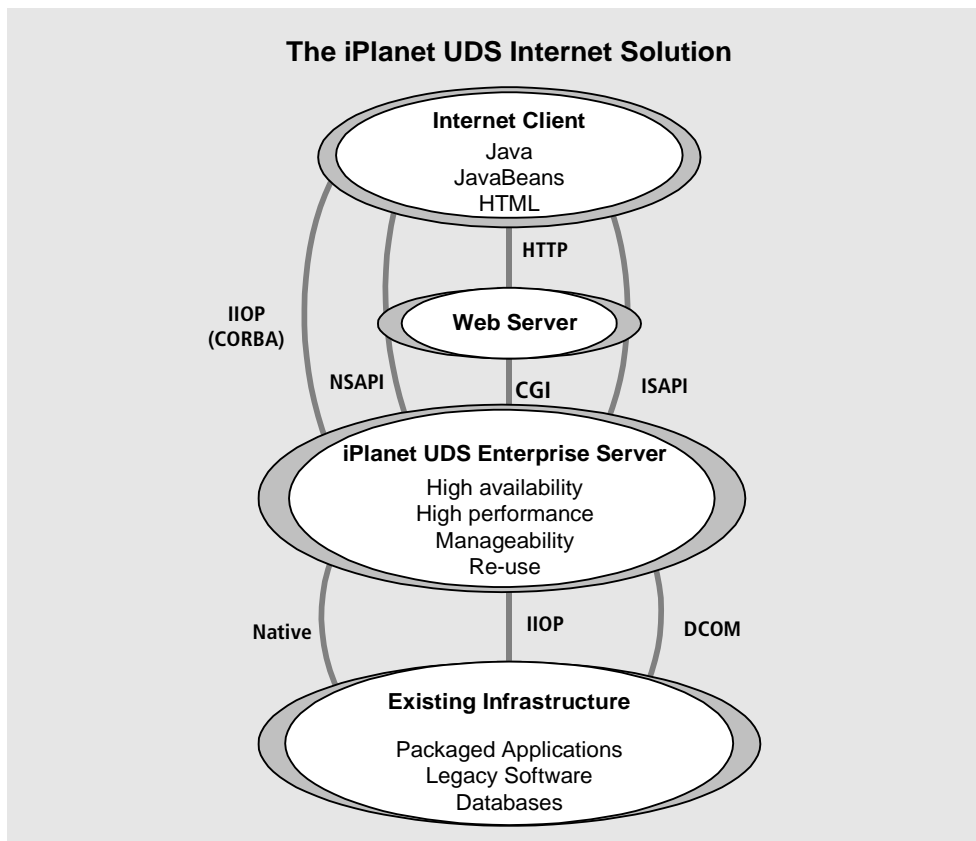
Overview

This chapter contains background information about WebEnterprise, providing a brief overview of the features that enable you to create enterprise Internet applications.

WebEnterprise provides several features for building enterprise internet applications. You can use the HTML support to create a Web user interface for an iPlanet UDS application.

The following figure illustrates iPlanet UDS internet solutions:

Figure 1-1 iPlanet UDS Internet Solutions



About HTML Support

HTML is markup language for creating Web pages that are displayed to end users by a Web browser. Web browsers provide users with a simple, consistent way to access information over the Internet. Using iPlanet UDS, you can enable users with Web browsers to access existing iPlanet UDS applications or iPlanet UDS applications you create especially for the Web.

To your end users, accessing an iPlanet UDS Web application is no different than accessing any Web site. To view an iPlanet UDS Web application, your end users can use a standard Web browser: all information is displayed to them on Web pages; they use hypertext links to move from one page to another, and so on. However, your iPlanet UDS Web application has all the power of iPlanet UDS processing behind it.

Creating Web pages dynamically Using iPlanet UDS to create a Web application enables you to create your Web pages dynamically, based on data in a relational DBMS as well as data from other sources. Most of the information on the Web is static and is served directly from HTML files. The information in these files changes only when the site administrator updates the file. When you create your Web pages dynamically, the information displayed on the Web page is automatically updated every time the page is downloaded. For example, in our SoftWear Web application, the list of catalog items offered for sale, and the text and images associated with each item, are all stored in the database. When a new catalog item is added to the database, the new item automatically appears on the Web page the next time that page is downloaded.

Exploiting iPlanet UDS processing Using iPlanet UDS to create a Web application also enables you to take advantage of all of the sophisticated computing and scaling features of iPlanet UDS. Any server you create using iPlanet UDS can provide services for the Web. Therefore, all the standard features of iPlanet UDS are available to you, including database access, event-based processing, and integration with external systems. Using load balancing and failover for the iPlanet UDS servers that provide the processing enables you to scale the application to handle large numbers of users simultaneously. Using code generation for iPlanet UDS partitions provides improved performance.

An important advantage of using a Web interface to an iPlanet UDS application is that the clients do not need to run the iPlanet UDS runtime system. Clients on the Internet need only a standard Web browser to access the full functionality of the iPlanet UDS application.

Multiple user interfaces Of course, a Web page does not provide exactly the same GUI features as those available in iPlanet UDS. If you wish to maintain your iPlanet UDS user interface and still provide Web access to the iPlanet UDS application, you can provide two interfaces for a single application. The same iPlanet UDS server, without modification, can interact simultaneously both with iPlanet UDS clients and with Web clients.

HTTP communication For communication between iPlanet UDS and the browser running the HTML pages, WebEnterprise uses HTTP with either a CGI program (fortecgi) or an NSAPI plug-in (fortensapi). [Figure 1-1](#) illustrates how the browser communicates with iPlanet UDS using HTTP.

WebEnterprise and HTML

This chapter provides background information about how you can use WebEnterprise's HTML features to create a Web-page based iPlanet UDS application. It also describes the design considerations you should keep in mind while designing your Web application.

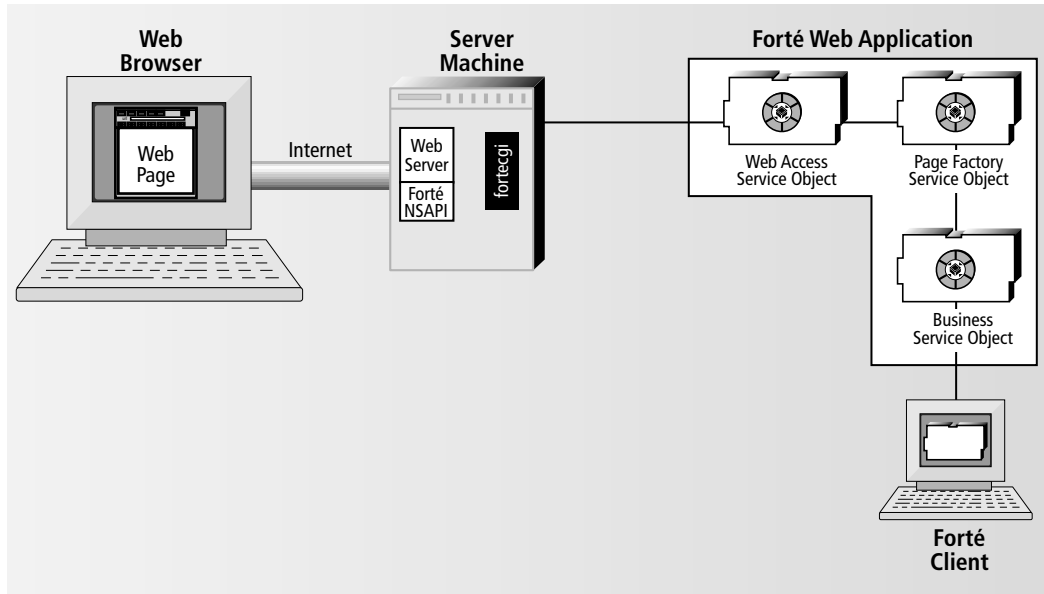
The chapter concludes with a quick tutorial that provides an overview of the basic steps for creating a simple iPlanet UDS Web site using HTML.

About iPlanet UDS and HTML

The HTML features in WebEnterprise allow you to create a Web interface for an iPlanet UDS application. Using WebEnterprise, you can enable users with Web browsers to access existing iPlanet UDS applications or iPlanet UDS applications you create especially for the Web.

To enable your iPlanet UDS application to interact with the World Wide Web, iPlanet UDS uses a combination of standard components, such as a standard Web server and Web browser, and special WebEnterprise components. **Figure 2-1** illustrates the standard components and WebEnterprise components within an iPlanet UDS Web application.

Figure 2-1 Using iPlanet UDS with the World Wide Web



Standard components The standard components you need to support an iPlanet UDS Web application include:

Web browser Both Netscape Navigator and Microsoft Internet Explorer can interface with an iPlanet UDS application.

Web server The following Web servers can provide the Web browser with access to the iPlanet UDS application: Netscape Enterprise Server, Microsoft Internet Information Server, and Process Software Purveyor Encrypt. (See the *WebEnterprise Installation Guide* for further information about Web servers.)

iPlanet UDS business server Any iPlanet UDS server can provide processing for the Web site.

WebEnterprise components The WebEnterprise components you need to build the iPlanet UDS Web application include:

fortecgi program or fortensapi fortecgi is a small CGI program that provides the gateway between the Web server and the iPlanet UDS Web access service object. When the Web server receives a request for an iPlanet UDS Web page, the fortecgi program passes that request to the iPlanet UDS Web access service object. The Web access service object then returns the appropriate Web page to the fortecgi program. The fortecgi program is included in WebEnterprise.

fortensapi is an NSAPI plug-in that provides a gateway between a Netscape Web server and the iPlanet UDS Web access service object. While the iPlanet UDS NSAPI plug-in is significantly faster than fortectgi, it runs only with Netscape Web servers. The fortectgi program and fortensapi plug-in are included in WebEnterprise.

See “[fortectgi Program and the iPlanet UDS NSAPI Plug-in](#)” on page 33 for information on fortectgi and the iPlanet UDS NSAPI plug-in.

Web access service object A special iPlanet UDS “Web-aware” service object that accepts requests for Web pages from the fortectgi program or the iPlanet UDS NSAPI plug-in, gets the appropriate Web page from the page factory object (which dynamically constructs the page), and returns the Web page to the fortectgi program or the iPlanet UDS NSAPI plug-in.

page factory service object A standard iPlanet UDS service object that dynamically creates the Web pages needed by the Web server, using information provided by the iPlanet UDS business server.

[Figure 2-1 on page 30](#) illustrates how the standard components and WebEnterprise components operate with each other. The Web browser and the Web server communicate normally with each other. The Web server communicates with the fortectgi program or the iPlanet UDS NSAPI plug-in to handle all iPlanet UDS requests.

When the Web server receives a request from the Web browser for an iPlanet UDS Web page, the fortectgi program or fortensapi plug-in passes the request to the Web access service object.

The Web access service object determines which page is needed and requests that page from the iPlanet UDS page factory service object. The page factory service object creates the Web page dynamically and returns the Web page to the Web access service object, which, in turn, sends the page to the fortectgi program or fortensapi plug-in. The fortectgi program or fortensapi plug-in returns the page to the Web server, and the Web server returns it to the Web browser as a standard HTML Web page.

The sections that follow provide definitions of basic terminology, and then present additional background information about the WebEnterprise components you need for your iPlanet UDS Web application: fortectgi, the iPlanet UDS NSAPI, the Web access service object, and the page factory service object.

Terminology

CGI program (Common Gateway Interface) A program that integrates the Web server with external programs. The Web server starts the CGI program, and the CGI program interacts with the external program (in our case, iPlanet UDS). Using a CGI program provides the ability to construct a Web page dynamically, rather than simply providing a static HTML page in response to a request from the Web browser. The CGI program used by iPlanet UDS is called “fortecgi.”

iPlanet UDS business server Any iPlanet UDS service object within an iPlanet UDS application that provides “server” processing for the application. In iPlanet UDS, you can create an iPlanet UDS Web application that integrates a standard iPlanet UDS business server with a Web site.

iPlanet UDS Web access server A special iPlanet UDS “Web-aware” service object that provides Web pages to the Web server through a CGI program or NSAPI plug-in.

HTML (Hypertext Markup Language) The most commonly used language for creating a Web page. HTML defines the content and the layout of the Web page, as well as links to other pages.

HTTP (Hypertext Transfer Protocol) The most commonly used protocol for transferring data on the World Wide Web.

Page A particular type of Web object that can contain text, inline graphics, and links to other objects. Usually, when the Web browser requests data from the Web server, it requests the data in the form of a Web “page.” A Web page is also referred to as a “document” or “HTML document.”

NSAPI plug-in A plug-in API to reference external services. The plug-in API contains external interface routines in a dynamically linked library (DLL) form. The DLL is loaded into the Web server process the first time it is referenced and then remains resident. When an external request is received from a browser, the server dispatches a worker thread which invokes the plug-in DLL. The DLL uses plug-in API functions (rather than environment variables) to get the data required for the external service call and to return an HTML response to the client.

The plug-in API avoids the overhead of process creation that is inherent in the CGI mechanism, and therefore is a higher-performance interface.

URL (Uniform Resource Locator) A scheme for specifying the exact location of a particular resource on the Internet.

Web browser A software program that allows the end user to retrieve a Web page and to follow links from one Web page to another.

Web server A software program that responds to requests from a Web browser for a Web page. Do not confuse the term “Web server” with an iPlanet UDS Web access server, which is an iPlanet UDS “web-aware” service object that provides an iPlanet UDS application with access to the Web. The following section provides information about the relationship between the Web server and the iPlanet UDS Web access server.

Web site A set of interlinked Web pages. A Web site can provide static information for end user viewing, or it can be used as a user interface, for example, to an iPlanet UDS application.

fortecgi Program and the iPlanet UDS NSAPI Plug-in

The Web browser uses a URL to request a Web page from the Web server. The sole function of the fortecgi program or the iPlanet UDS NSAPI plug-in is to direct the URL from the Web server to the appropriate iPlanet UDS Web access service object running in your iPlanet UDS environment.

The fortecgi program and the iPlanet UDS NSAPI plug-in provide equivalent functionality using different APIs. The NSAPI interface was developed by Netscape and is supported by their current generation of Web servers; it is generally a higher performing alternative. See [“Choosing between fortecgi and the iPlanet UDS NSAPI Plug-In” on page 199](#) for detailed information about when to use fortecgi and when to use the iPlanet UDS NSAPI plug-in.

Using URLs to access an iPlanet UDS service object When you create a URL to request a Web page from your iPlanet UDS application, the URL must contain a reference to the fortecgi program or the iPlanet UDS NSAPI plug-in, followed by a service name that identifies an iPlanet UDS Web access service object and a Web page name with optional parameter list.

The following example shows the first portion of a sample URL, including the reference to the fortecgi program followed by the iPlanet UDS service name:

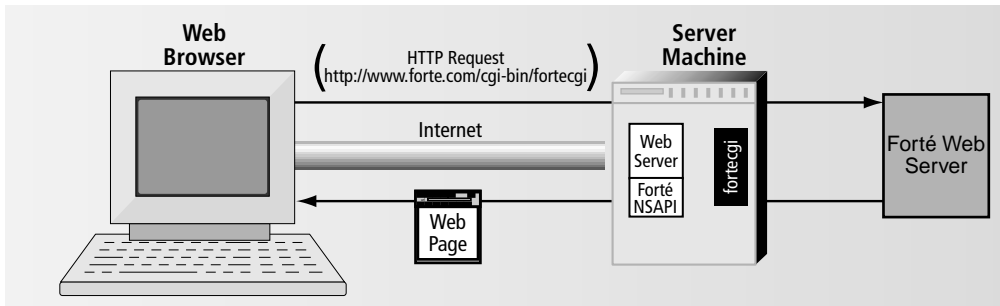
```
http://www.forte.com/cgi-bin/fortecgi?serviceName=fortedemo&...
```

The next example shows the same URL fragment with a reference to the iPlanet UDS NSAPI plug-in instead of the fortecgi program:

```
http://www.forte.com/web.forte?serviceName=ShopCartService&...
```

When the Web server receives a URL request for a Web page, it forwards the request to the appropriate program. **Figure 2-2** illustrates how the Web server forwards the URL request that contains a fortecgi reference to the fortecgi program:

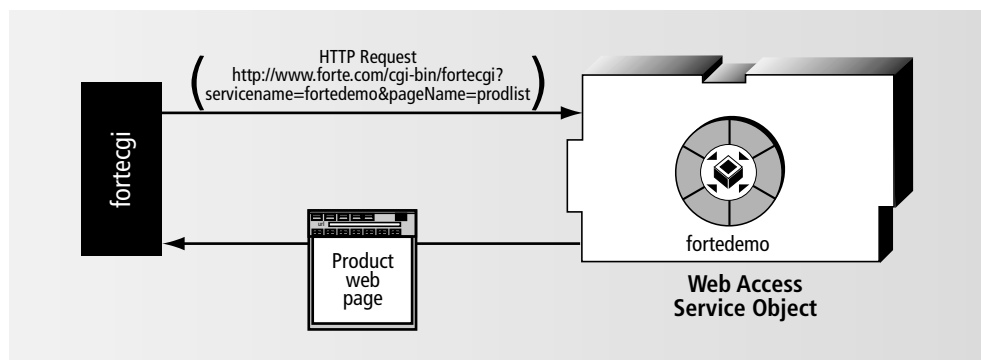
Figure 2-2 The Web Server and the fortecgi Program



The process is the same for the iPlanet UDS NSAPI plug-in. If the URL shown in **Figure 2-2** contains a reference to the iPlanet UDS NSAPI plug-in (rather than fortecgi), the page request is forwarded to the fortensapi plug-in rather than the fortecgi program.

The fortecgi program or the iPlanet UDS NSAPI plug-in uses the service name in the URL to locate the appropriate iPlanet UDS Web access service object and forwards the URL to the service object, which then obtains the appropriate page and returns it to the fortecgi program or iPlanet UDS NSAPI plug-in. **Figure 2-3** illustrates how the fortecgi program directs URL requests to the iPlanet UDS Web access service object using the service name:

Figure 2-3 The fortecgi program and the Web Access Service Object



The process is the same for the iPlanet UDS NSAPI plug-in. If the URL shown in [Figure 2-3](#) contains a reference to the iPlanet UDS NSAPI plug-in (web.forte), the page request is forwarded from the iPlanet UDS NSAPI plug-in to the Web access service object.

Both the fortectgi program and the iPlanet UDS NSAPI plug-in are provided as part of WebEnterprise. See the *WebEnterprise Installation Guide* for information about installing fortectgi and the iPlanet UDS NSAPI plug-in.

Web Access Service Object

The purpose of the Web access service object is to obtain the Web pages requested by the fortectgi program or iPlanet UDS NSAPI plug-in. The Web pages that the Web access service object obtains are constructed by a page factory service object.

There are two kinds of page factory service objects: page builder and scanner. A *page builder service object* creates a Web page by using an iPlanet UDS method that contains HTML or by converting an iPlanet UDS window into a Web page. A *scanner service object* creates a Web page by using an HTML template that was created by an HTML editor. See [“Page Factory Service Objects” on page 38](#) for information on page factory service objects.

The URL request sent to the Web access service object by fortectgi or iPlanet UDS NSAPI plug-in contains either:

- a Web page name and an optional parameter list
- an HTML template name and an optional parameter list

If the URL contains a Web page name, the Web access service object directs the request to the appropriate page builder service object. If the URL contains a template name, the Web access service object directs the request to the appropriate scanner service object.

Requests for pages When the URL contains a page name, the Web access service object uses the page name to determine which Web page needs to be constructed.

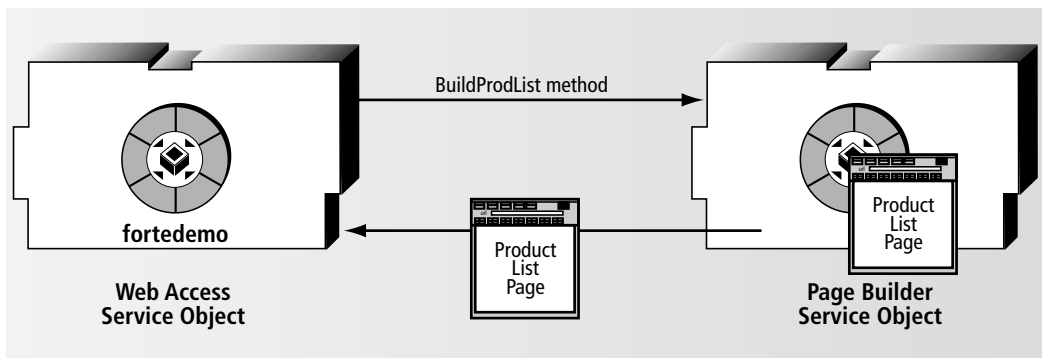
For example, the following URL contains a request for the “prodlist” page:

```
http://www.forte.com/cgi-bin/fortectgi?serviceName=fortedemo
&pageName=prodlist
```

When the Web access service object receives a request for a Web page, it invokes a method on the page builder service object, requesting the appropriate page. If there are page parameters, the Web access service object passes the page parameter values to the page builder service.

Figure 2-4 illustrates how the Web access service object obtains the dynamically created Web page from the page builder service object.

Figure 2-4 Using the Page Builder Service



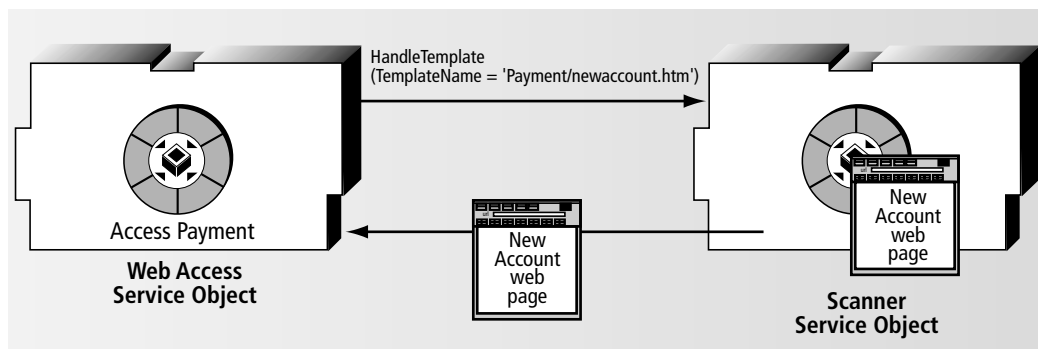
Requests for templates When the URL contains a template name, the Web access service object uses the template name to determine which template needs to be used by the scanner service object to construct the Web page.

For example, the following URL contains a request for the “shopmain” template:

```
http://www.forte.com/cgi-bin/fortecgi?serviceName=ShopCartService
&templateName=shopping/shopmain.htm
```

When the Web access service object receives a request for a template, it invokes a method on the scanner service object, requesting the appropriate template be used to create a Web page. If there are parameters, the Web access service object passes the parameter values to the scanner service object.

Figure 2-5 illustrates how the Web access service object obtains the dynamically created Web page from the scanner service object.

Figure 2-5 Accessing Dynamically Created Web Page

HTTPAccess class To create your Web access service object, you create a subclass of the iPlanet UDS HTTPAccess class, provided in the iPlanet UDS HTTP library. The HTTPAccess class provides two separate methods for handling requests for a Web page:

HandleRequest This method is automatically invoked when the Web access service object receives a URL with a page name. The HandleRequest method allows the Web access service object to request the appropriate Web page from the page builder service object. If your Web access service object is going to handle page requests that contain page names, you must override the HandleRequest method.

HandleTemplateRequest This method is automatically invoked when the Web access service object receives a URL with a template name. The HandleTemplateRequest method allows the Web access service object to request the appropriate Web page from the scanner service object. If your Web access service object is going to handle page requests that contain template names, you must override the HandleTemplateRequest method.

A single Web access service object can handle both kinds of requests, those that contain a page name and those that contain a template name.

Registering the Web access service object The HTTPAccess class also provides methods for registering and de-registering your Web access service object with the fortcgi program or iPlanet UDS NSAPI plug-in. You can enable HTTP access for your Web access service object in the Init method for the HTTPAccess subclass, so that HTTP access is automatically turned on when the service object starts up. Or,

you can create a separate iPlanet UDS client window that provides commands you can use to enable and disable access to the World Wide Web. See [“Web Access Service Object” on page 35](#) for information on registering your Web access service object.

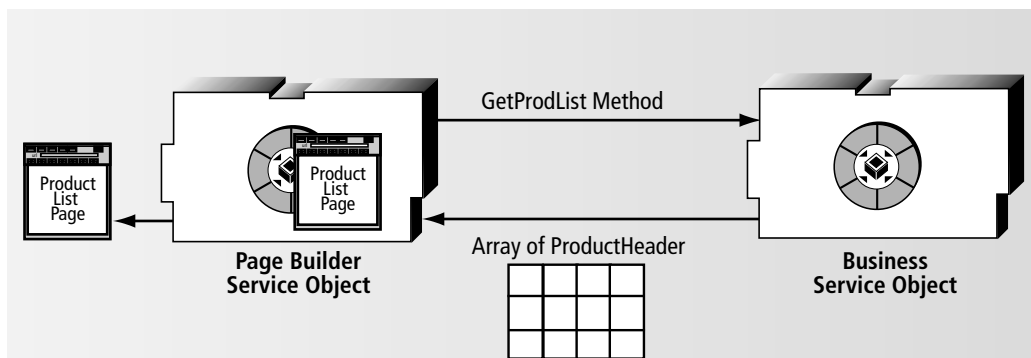
Page Factory Service Objects

The purpose of a page factory service object is to create the Web pages requested by the Web access server. When the page factory service object receives a request for a Web page from the Web access service object, it obtains the appropriate data from the iPlanet UDS business server and then creates the Web page dynamically. For example, to build the product list page, the SoftWear application’s page builder service object gets the product list information from the iPlanet UDS business service object and then creates the HTML page.

WebEnterprise provides two different kinds of page factory service objects: the page builder service object (which creates Web pages by using iPlanet UDS methods that generate HTML or by converting an existing iPlanet UDS window into a Web page) and the scanner service object (which creates Web pages by using an HTML template produced by an HTML editor).

[Figure 2-6](#) illustrates how a page builder service object dynamically creates the Web page using data provided by the iPlanet UDS server. The process is essentially the same for a scanner service object:

Figure 2-6 Constructing Web Pages Dynamically



Page builder service object A page builder service object creates Web pages by using iPlanet UDS methods that generate HTML or by converting an existing iPlanet UDS window into a Web page. When you create Web pages dynamically in your page builder service object, you have two basic options: enter HTML tags directly into your method code or use the iPlanet UDS HTML projects.

Using iPlanet UDS HTML projects iPlanet UDS provides the following three projects for constructing Web pages in your TOOL code:

HTML Project	Description
HTML	Provides classes that correspond directly to HTML elements. You set the attribute values to specify the element's "tags."
HTMLWindow	Provides a class that allows you to convert an iPlanet UDS window into an HTML document (that is, a Web page).
HTMLSQL	Provides a class that allows you to create a table on the Web page from a SQL query.

The advantages of using the HTML projects are that iPlanet UDS provides syntax checking for the TOOL code, and the HTML classes create fewer lines of code than HTML. In addition, you can use the iPlanet UDS Window Workshop to design your Web page, taking advantage of its visual layout tools, and then convert the window into a Web page. You can also convert an existing window (used by the iPlanet UDS client) into a Web page.

For complete information on creating a page builder service object, see [“Defining a Page Builder Service Object” on page 72](#).

Scanner service object The scanner service object creates Web pages by using an HTML template produced by an HTML editor, like Sausage Software’s HotDogPro or Allaire’s Homesite. Using the scanner service object to dynamically create your Web pages enables you to use a commercial HTML editor to design the basic format of the Web page, rather than entering HTML directly into your method code.

Using templates An iPlanet UDS *HTML template* is a text file that contains standard HTML and iPlanet UDS HTML tags. It differs from a regular HTML file only in that it contains additional tags representing dynamic data added to the HTML page by an iPlanet UDS scanner service. A page author creates an HTML

template using a text editor or an HTML editor, inserting special iPlanet UDS variables in the template to stand in for data that will be provided by an iPlanet UDS programmer. The iPlanet UDS programmer then adds in the iPlanet UDS tags and variables required to generate the desired data.

For complete information on creating a scanner service object, see [“Defining a Scanner Service Object” on page 69](#).

When the Web access service receives a request from a browser, it determines from the URL itself whether the request is for a page or a template. The Web access service directs all requests for pages to the page builder service object and all requests for templates to the scanner service object.

Application Design Considerations

There are several effective ways to design your iPlanet UDS Web application. The following sections describe two of the basic issues you should consider when you design your Web application and recommend some design alternatives. Topics covered include:

- structuring your Web application
- scaling your Web application

Structuring Your Web Application

There are three basic strategies for structuring your Web application. You can:

- create a Web interface for a new iPlanet UDS server
- add a Web interface to an existing iPlanet UDS client application
- share an existing iPlanet UDS server between a Web-based application and a window-based application

In general, we recommend a design in which you define each of the service objects in your Web application in separate projects. This is the most effective design because, when you partition the application, the project code for each service object will be on separate partitions. However, the scanner service object (unlike the page builder service object) must be defined in the same project as the Web access service object. Therefore, all three strategies tell you to define the scanner service object (if you have one) in the same project that defines the Web access service object.

Creating a Web Interface for a New iPlanet UDS Server

To create a new application with a new iPlanet UDS server and a Web interface, you can simply create the following projects:

- a project that defines the Web access service object and the scanner service object (if you have one)

This is the main project for the application. If you are using a scanner service object to create the Web pages, the scanner service object must be defined in the same project as the Web access service object.
- a project that defines the page builder service object (if you have one)

This project must be a supplier project to the project that defines the Web access service object.
- a project that defines the iPlanet UDS business server to provide the processing for the Web site

This project must be a supplier project for both the project that defines the Web access service object and for the project that defines the page builder service object.

When you are ready to partition the Web application, configure the main project as a server application, rather than as a client. To run the application, you can simply start up the iPlanet UDS server and then allow end users to access it by using a Web browser with the appropriate URL to get to the application's entry point Web page.

Adding a Web Interface to an iPlanet UDS Client Application

If you have an existing iPlanet UDS application with a standard iPlanet UDS client, and you wish to add Web access to that application, you can extend your existing application to include the iPlanet UDS Web access service object. In this design, the same application contains both the Web access server and the standard iPlanet UDS client.

You can implement the Web interface version of the existing client either in a page builder service object (for example, by using the WindowConverter class to convert the existing windows into HTML pages) or in a scanner service object (that is, by using an HTML editor to create an HTML "version" of the window-based interface).

The SoftWear sample application, described under **"Components of the SoftWear Example" on page 227**, uses this design. SoftWear illustrates use of the WindowConverter class to convert existing iPlanet UDS windows into Web pages.

To extend your client application, create the following new projects:

- a project that defines the Web access service object and the scanner service object (if you have one)

This project must be a supplier project for the application's main project. If you are using a scanner service object to create the Web pages, the scanner service object must be defined in the same project as the Web access service object.

- a project that defines the page builder service object (if you have one)

This project must be a supplier project to project that defines the Web access service object.

You then must make the project that defines the business server a supplier project for both the project that defines the Web access service object and the project that defines the page factory service object.

When you are ready to partition the new version of the application, configure the main project as a client application. When the client application is running, Web users can access it by using a Web browser with the appropriate URL to get the application's entry point Web page.

See [“Summary of Steps for Creating an iPlanet UDS Web Application”](#) on page 57 for further information about implementing this design.

Sharing an iPlanet UDS Server

If you have an existing iPlanet UDS application with a standard iPlanet UDS client, you might wish to create a completely separate Web version of your application, which still accesses the same iPlanet UDS business service, but does not include the standard iPlanet UDS client. The advantage of creating a separate Web version of the application is that you do not have to modify your existing iPlanet UDS application. You can create a completely separate Web version of your application by using a *reference partition*.

Reference partitions To define a Web interface for an existing iPlanet UDS server, create the following projects:

- a project that defines the Web access service object and the scanner service object (if you have one)

This is the main project for the application. If you are using a scanner service object to create the Web pages, the scanner service object must be defined in the same project as the Web access service object.

- a project that defines the page builder service object (if you have one)

This project must be a supplier project to project that defines the Web access service object.

You must then import the project that defines the iPlanet UDS business service into your repository. After the project that defines the iPlanet UDS business service is in your repository, you must make it a supplier project to both the project that defines the Web access service object and the project that defines the page factory service object.

When you are ready to partition the Web application, configure the main project as a server application, rather than as a client. You can then create a reference partition for the iPlanet UDS business service in your new application. Instead of containing a new service object, the reference partition points to the existing service object that was originally deployed as part of the first application.

See *A Guide to the iPlanet UDS Workshops* for complete information about creating reference partitions.

To run the application, you can simply start up the iPlanet UDS server and then allow end users to access it by using a Web browser with the appropriate URL to get to the application's entry point Web page.

Scaling Your Web Application

One of the important advantages of using iPlanet UDS to provide the server processing for a Web site is that you can take advantage of the scaling features of iPlanet UDS: load balancing and failover.

Replicating the iPlanet UDS business server A single Web access server can interact with any number of replicates of the iPlanet UDS business server. Thus, the data processing for multiple requests from the Web can be distributed across multiple partitions or multiple nodes.

However, the Web access server itself cannot be replicated. When the `fortecgi` program or iPlanet UDS NSAPI plug-in forwards an HTTP request to the Web access server, it uses a port number to locate the particular server. Therefore, there cannot be multiple copies of this server.

Replicating the page factory service object Because you cannot replicate the Web access server, it is possible that the Web access server could create a performance bottleneck in your application. Obviously, the processing required to construct Web pages is more significant than checking for the page type and returning the page. Therefore, the design we recommend divides the Web page

processing between two service objects. The Web access service object checks only for the page type and returns the appropriate page. The page factory service object does the bulk of the work, interacting with the iPlanet UDS business service and constructing the actual HTML Web pages. Whether you use the page builder or the HTML scanner service object to generate your Web page (see [“Determining which Services will Provide Pages” on page 80](#)), the page factory service object is a standard iPlanet UDS service object and you can replicate it as you would any standard iPlanet UDS service object.

However, if performance is not a major concern, you can streamline your architecture by eliminating the page factory service object and having the Web access service object construct the HTML pages. See [“Quick Tutorial: EasyWeb” on page 50](#) for an example of using this design.

See [Chapter 8, “Partitioning and Deployment”](#) for additional information about partitioning iPlanet UDS Web applications and replicating service objects.

Differences between Window- and Web-Based Applications

Whether you are creating a Web site to serve as a user interface for an existing iPlanet UDS application or you are creating a new iPlanet UDS application to provide processing for a Web site, there are three important characteristics about the Web that you must keep in mind:

- Web pages do not post events.

The only time your iPlanet UDS application can detect changes to the Web page is after the end user clicks the Submit button.

- The Web is stateless.

Because requests for Web pages are separate messages, you must use special WebEnterprise features to maintain state information about a single user’s interaction with the application.

- Web users can move between the Web pages arbitrarily.

Because Web browsers allow users to navigate through Web pages by using the browser’s Back, Forward, and Home buttons, you cannot control the structure of the application the same way you can in a window-based interface.

An application designed for a window-based user interface typically uses events to provide processing based on the end user's actions, and state information to track a single user's interactions with the application. And, with a window-based user interface, the application controls the exact order in which the windows are presented to the end user.

Unfortunately, the differences between window- and Web-based applications make it impossible for you to use exactly the same code for your window-based user interface as for your Web-based user interface. For example, a Display method designed for a window that performs data validation within an event loop cannot be used with a Web page.

However, by designing your application appropriately, you can share most of your user interface code between the two user interfaces. The following sections provide background information about the differences between the window- and Web-based user interfaces, and describe how to modify your iPlanet UDS application to share code between them.

Performing Data Validation and Other Window Processing

An iPlanet UDS window uses events to trigger data validation. As the end user moves from one field to another on a window, entering and changing data, the Display method is active, and each event can trigger data validation. By using an event loop to handle the window events, the application can perform data validation on a field-by-field basis.

When an error is detected in a window, the application displays an error message to the end user and prevents the user from leaving the field that has the error.

Submit button for Web pages A Web page, on the other hand, does not interact with the application until the end user clicks the Submit button. As the end user moves from one field to another on the Web form, entering and changing data, there is no way for the application to detect movement or changes. When the end user clicks the Submit button, the completed Web form is sent to the application, which must then perform data validation on the form as a whole.

When an error is detected on a Web form (after the Submit button is clicked), the application displays an error Web page to the end user. The end user can then go "back" to the Web form, correct the error, and then click the Submit button again.

The `WindowConverter` class (described in the iPlanet UDS online Help) allows you to convert the visual appearance of an existing iPlanet UDS window into a Web page. However, because the `Display` method for a window is based on events, you cannot use the existing window's `Display` method to provide processing for a Web page. Instead, the page builder method for your Web page needs to provide the equivalent processing.

Sharing code between window and Web page Although you cannot use the same method to handle processing for both the window and the Web page, you can share your window initialization, data validation, and operation code between them.

To use the window initialization, data validation, and operation code in your `Display` method for the corresponding Web page, you must modify your original `Display` method. Rather than including all initialization, data validation, and operation code within the `Display` method itself, you must create independent initialization, validation, and operation methods that both the `Display` method and the page builder service can call.

After you create your independent methods, you can streamline your original `Display` method. Your new version of the `Display` method can start by invoking the initialization methods, and then provide an event loop that invokes data validation methods in response to events on the window.

After rewriting the `Display` method for the window, you can write the page builder method for creating the corresponding Web page. The page builder method can start by invoking the same initialization methods on the window object to load the data into it. The method can then convert the window object into a Web page, which will be displayed to the Web user.

After the Web user enters data onto the Web page, the page builder service can use the `LoadParameters` method of the `WindowConverter` class to load the Web page parameters (the data) back into the original iPlanet UDS window object. The page builder service can then invoke the original data validation and processing methods directly on the iPlanet UDS window object itself.

See [“Sharing Window Code with the Web Page” on page 151](#) for an example of sharing window code.

Keeping State Information

State information is information that must be maintained for a *single user* of the application. For example, when the user of the SoftWear Catalog application places his first order, a shopper ID is stored as state information for the duration of the shopping session. All subsequent orders placed by the user are added to a single shopping basket, which the user can view at any time.

In standard iPlanet UDS window-based applications, you can store state information on both the client and the server. To store state information on the client, you can use local objects. For example, in its standard iPlanet UDS window version of the user interface, the SoftWear Catalog application uses the ShopperID attribute of the ClientBroker class to store the ID of the shopper. To store state information on the server, you can use transaction or session dialog duration service objects.

When you are interacting with the Web, however, you must handle state information much differently.

State information on the client Because there are no local iPlanet UDS variables on a Web page, if you wish to maintain state information on the Web client, you must store it in one of the following formats:

- a cookie

Using WebEnterprise's cookie feature, you can construct text strings containing session-specific information, and store the cookies on the Web client when you return a Web page. You can then retrieve the cookie information for use or updating during subsequent interactions with that user. Note, however, that users and/or system administrators can disable or limit the use of cookies on their machines.

- a hidden form element on the Web page
- a page parameter in the URLs included as links on the Web page

See [“Using Cookies” on page 186](#) for further information on cookies and on handling state on the client.

State information on the server Because requests for Web pages are separate messages, the Web access service object cannot have transaction or session dialog duration. When the Web access service object receives a request for a Web page from the fortectgi program, it returns the Web page to fortectgi and the connection is ended. Therefore, the Web access service object has message dialog duration.

To enable you to handle state information on the server, WebEnterprise provides session management features for managing state, identity, and security information. The WebEnterprise session management features allow you to build continuity into the application as users navigate, however randomly, through a Web site. Session management also gives users the feeling that the application is tailored to them individually.

Some examples of session management are:

- automatic user validation
- customized responses (returning different pages or data) based on a user's recent actions
- tracking a user's progress through an application
- automatic timing out of sessions that have expired
- avoiding duplicate sessions

See [“Session Management Features” on page 156](#) for complete information on session management and on handling state on the server.

When the application requires an ongoing connection between the Web page and the iPlanet UDS application, you can use a Java applet in your Web page that communicates with an iPlanet UDS service object. However, doing so requires using the iPlanet UDS Java interoperability feature. For more information, see *iPlanet UDS Java Interoperability Guide*.

Structuring the User Interface

When you create a window-based user interface, you provide the commands and buttons that let the end user move from one window to another. When a window is closed, the end user cannot open it again unless you provide the mechanism for him to do so.

Buttons for browsing downloaded pages With a Web-based user interface, you have much less control over the order in which the end user views the Web pages. Of course, you provide the hypertext links that allow the user to navigate forward from one window to another. However, once a page has been downloaded, the end user can go “back” to it at any time. The Back, Forward, and Home buttons in the browser allow the user to move between the pages in an arbitrary manner. In addition, bookmarks allow the end user to access the application from an arbitrary page rather than starting with the designated entry point page. Bookmarks can create a security problem if the end user bypasses the only page that requires a password.

Unlike a window-based user interface, the Web displays a single page at a time. So, although the end user has access to all pages that have been downloaded, he can only display one window at a time. A window-based user interface, on the other hand, can display any number of windows concurrently.

These differences between the window- and Web-based user interfaces make it impossible for you to create a Web interface that is structured identically to your window-based interface. When you are designing your Web pages, you should keep these differences in mind. You might, for example, want to repeat information on Web pages that would be unnecessary to repeat in any application that displays multiple windows concurrently. Since you can use Enterprise's session management features to track the user's progress through the application, you can also provide special programming to handle some of the navigation problems.

Another difference to consider is that with a window-based user interface, your application has continual access to the windows. iPlanet UDS applications use events to keep concurrent windows synchronized. For example, if, using the product detail page, the user purchased the last item in stock, you could post an event to the product list page to remove that item from the list of available products.

With a Web-based user interface, however, your application only has access to a page at the time it is created or submitted. If the user makes a change to one page that has an effect on another page that is already downloaded, you will not be able to update the affected page. For example, once the product list page has been downloaded, there is no way for you to update it. This difference might have an effect on what kind of information you display on each Web page.

Again, all these differences between window-based and Web-based user interfaces are simply due to the nature of the Web, and you should keep them in mind while planning your Web pages.

NOTE If your application requires an ongoing connection between the Web page and the iPlanet UDS application, you can include on the Web page a Java applet that communicates with an iPlanet UDS service object. A Java applet that is communicating with iPlanet UDS would allow you to update the Web page dynamically. However, to add a Java applet to an iPlanet UDS Web page, you must use the iPlanet UDS Java interoperability features as described in the *iPlanet UDS Java Interoperability Guide*.

Quick Tutorial: EasyWeb

The EasyWeb tutorial demonstrates the fewest possible steps required to build an iPlanet UDS Web application. It requires only a Web access service object (no page factory service object), and uses only the `HandleRequest` method to return pages to the end user (no page builder methods).

Before you begin this tutorial, make sure that:

- The HTTP library is installed in your development environment and has been imported into your development repository.
- The `fortecgi` program is installed on the Web server (and you know its URL). Refer to the section in the *WebEnterprise Installation Guide* that describes how to install WebEnterprise on your web server.

➤ Create an entry point Web page for the EasyWeb application using HTML

The EasyWeb application's entry point Web page provides the end user with access to the iPlanet UDS application and is the first Web page that the user will see. The entry point Web page for EasyWeb is a simple, standard HTML file, which includes starting and ending HTML tags, text, and options. The page has one fill-in field requesting "Your Name," and it prompts the user to indicate which of three pages to return.

1. Create and edit a file containing the following HTML. Alternatively, you can import the text from the file `$YOUR_ROOT/forte/examples/easyweb/easyweb.htm` (where `YOUR_ROOT` is the directory on the Web server where you copied the WebEnterprise directory `/cdrom/web/platform/forte`).

```

<HTML>
<HEAD>
<TITLE> Easy Page </TITLE>
</HEAD>
<BODY>
<FORM action="http://www.forte.com/cgi-forte/fortecgi">
<INPUT type=hidden Name=serviceName Value=EasyWeb>
Your Name: <INPUT type=text Name=userName></INPUT><BR>
Which page?<BR>
<INPUT type=radio Name=pageName value=hello>Hello</INPUT><BR>
<INPUT type=radio Name=pageName
value=goodbye>Goodbye</INPUT><BR>
<INPUT type=radio Name=pageName value=good luck>Good
Luck</INPUT><BR>
<INPUT type=submit>
</FORM>
</BODY>
</HTML>

```

2. In your copy of the file, change the line that begins with “<FORM action...” to reflect the URL of the Web server on which fortectgi is installed. This line is required in every Web page (HTML document) in your iPlanet UDS Web application to identify the fortectgi program that returns the page. The portion of the URL shown above as “www.forte.com” often takes the form *your_machine.your_domain.com*. Note that when your Web server is running on NT, you must replace “fortectgi” with “fortectgi.exe” in the URL.
 3. Save the application’s entry point Web page file to the same location on your Web server so that you will be able to access it from your Web browser. Save the file as *\$YOUR_ROOT/forte/examples/easyweb/easyweb.htm*.
- **In iPlanet UDS, create a new Web project to define the Web access service object**
1. In the Repository Workshop, choose Plan > New Project.
 2. In the New Project dialog, name your new project “EasyWeb” and turn off the Include Display and Include Database toggles. EasyWeb does not require the Display and GenericDatabase libraries.
 3. Click OK. The Project Workshop for the new project, EasyWeb, opens.

➤ **Include the iPlanet UDS HTTP library as a supplier plan for the new Web project**

1. In the Project Workshop for EasyWeb, choose File > Supplier Plans and include the HTTP library.

The WebEnterprise HTML projects are not required for EasyWeb.

➤ **In the EasyWeb project, create a new subclass of HTTPAccess**

1. From the Project Workshop for EasyWeb, either click on the New Nonwindow Class tool or choose Component > New > Nonwindow Class.
2. Name the new class "EasyAccess" and use HTTPAccess as its superclass.

➤ **Create a Web access service object with the new EasyAccess class as its type**

1. From the Project Workshop for EasyWeb, either click on the New Service Object tool or choose Component > New > Service Object.
2. In the New Service Object Dialog, name this new service object "EasyAccessService" and choose "TOOL class" as its base class.
3. In the Service Object Properties dialog, specify the EasyAccess class as the service object's class.

➤ **Override the HandleRequest method in the EasyAccess class**

In this step, you override the HandleRequest method inherited from the HTTPAccess class to provide the code that handles each request for a Web page, returning the requested page to the browser.

1. In the Repository Workshop, double-click the HTTP library. In the Project Workshop for the HTTP library, open the HTTPAccess class and select the HandleRequest method. Drag the HandleRequest method from the HTTPAccess class and drop it on the EasyAccess class.

2. Open the HandleRequest method in the EasyAccess class.

The Method Workshop displays no text for this method because it was copied from a library rather than a project. Do *not* change the method's parameters or return value.

3. Either type in the following method code or use the Import Text command in the Method Workshop to import the text from the file
`$FORTE_ROOT/install/examples/web/easyweb/easyhr.txt`.

```

response : HTTPResponse = new;

//Find the page name.
pageName : TextData = new;
pageName.SetValue(request.PageName);
returnData : TextData = new;

//Generate response pages.
if pageName.IsEqual('hello', IgnoreCase = TRUE) then
    returnData.Concat('<HTML>');
    returnData.Concat('<HEAD><TITLE>Hello
Page</TITLE></HEAD>');
    returnData.Concat('<BODY>');
    returnData.Concat('Hello, ').Concat(request.
        FindNameValue('userName'));
    returnData.Concat('</BODY>');
    returnData.Concat('</HTML>');

elseif pageName.IsEqual('goodbye', IgnoreCase = TRUE) then
    returnData.Concat('<HTML>');
    returnData.Concat('<HEAD><TITLE>Goodbye
Page</TITLE></HEAD>');
    returnData.Concat('<BODY>');
    returnData.Concat('Goodbye, ').Concat(request.
        FindNameValue('userName'));
    returnData.Concat('</BODY>');
    returnData.Concat('</HTML>');

elseif pageName.IsEqual('good luck', IgnoreCase = TRUE) then
    returnData.Concat('<HTML>');
    returnData.Concat('<HEAD><TITLE>Good Luck
Page</TITLE></HEAD>');
    returnData.Concat('<BODY>');
    returnData.Concat('Good Luck, ').Concat(request.
        FindNameValue('userName'));
    returnData.Concat('</BODY>');
    returnData.Concat('</HTML>');
else
    returnData.Value = 'Unknown page ";
    returnData.Concat(pageName).Concat('".');
end if;

response.AssignResponse(returnData);
return response;

```

NOTE This method simply uses HTML tag syntax (starting and ending tags for each HTML element). It creates three simple pages (titled “Hello,” “Goodbye,” or “Good Luck”), and, based on the input from the Web user, it returns one of the pages or an error.

➤ **Create a start-up class and method to start the iPlanet UDS application**

The purpose of the start-up class that you create in this step is to start the iPlanet UDS application. When the iPlanet UDS application starts, it will register the Web access service object with the fortectgi program, enabling Web users to access the iPlanet UDS application.

1. In the Project Workshop for EasyWeb, create a new class called `StartWeb`. You can either click the New Class icon or choose Component > New... > Nonwindow Class. Name this new class “`StartWeb`” and use `Object` as the superclass.
2. In the `StartWeb` class, create a method call “`StartUp`” with no parameters and no return value. The `Start-up` method should contain the following code:

```
event loop
  when task.Shutdown do
    exit;
  end event;
```

3. In the Project Workshop, choose File > Start Class Method and set the start class for your project to “`StartWeb`” and the start method to “`Start-up`”.

➤ **Register the Web access service object with the fortectgi program**

In the Method Workshop, add code to the `Start-up` method for the `StartWeb` class to register the Web access service object with the fortectgi program using the `EnableAccess` method. Place the following code before the event loop:

```
EasyAccessService.EnableAccess('EasyWeb', 7777,
  'http://www.forte.com/cgi-forte/fortectgi');
```

Use the following values for the `EnableAccess` method's parameters:

- "EasyWeb" for the first parameter
- the port number for the node where the `EasyAccessService` service object is going to be running for the second parameter

The port number is used by the socket—be sure to pick a number that is not already in use on your machine.

- the URL for your `fortecgi` executable for the third parameter

Replace the "`www.forte.com`" portion of the URL shown above with your Web server's name, typically in the form `your_machine.your_domain.com`. If you are running on NT or VMS, replace "`fortecgi`" with "`fortecgi.exe`".

► Test the application

1. In the Project Workshop, choose Run > Test Run to run the iPlanet UDS application. Note that it may take about a minute for the Web access service object to start.
2. Use your Web browser to open the application's entry point Web page. Then, enter your name, choose a salutation, and click on the Submit button.
3. You will see a new Web page, with your personalized greeting on it.

If you encounter any problems while trying to run this tutorial, see ["Troubleshooting Web Client Errors" on page 218](#) or ["Troubleshooting Web Administrator Errors" on page 222](#). Also, refer to the *WebEnterprise Installation Guide* if you suspect the installation was not complete.

► Exit the application

1. Quit the Web browser.
2. In iPlanet UDS, choose Run > Cancel Run to exit the iPlanet UDS application.

Setting Up a Web Application

This chapter describes the tasks involved and the alternatives you consider when you start to design an iPlanet UDS Web application or user interface. The topics in this chapter include:

- using the predefined iPlanet UDS Web projects and classes
- creating the iPlanet UDS services appropriate for your application
- starting and stopping Web access to an iPlanet UDS Web application
- making initial decisions about pages, such as session management requirements

Summary of Steps for Creating an iPlanet UDS Web Application

The following table summarizes the steps to build an iPlanet UDS Web application. This list is the comprehensive set of steps required to build an initial iPlanet UDS Web application.

The order of steps is only a suggested order. In addition, some steps are optional depending upon the requirements of your application. As you become familiar with WebEnterprise features, you will understand how you can deviate from the order and which steps are optional under different circumstances.

This manual describes these steps in detail, with several examples.

Summary of Steps for Building an iPlanet UDS Web Application

Responsible Person	Step Description	See
Author and Programmer	Design the application's page content and navigation.	page 78
	Determine whether to use the scanner service, page builder service, or both.	page 80
	Identify dynamic data and session management requirements.	
Programmer	Define the Web access service object.	page 64
Programmer	If you will use a scanner service:	
	Define the scanner service object.	page 72
	Layout HTML templates using an HTML editor.	page 108
	Create the tag handler class(es) that implements the TagHandlerInterface interface. Use either: a subclass of HTMLScanner a custom class.	page 112 page 113 page 114
	Define the HandleTag method.	page 117
	Define the HandleCondition method.	page 123
	Define the HandleTemplateRequest method.	page 111
	If you used custom classes, choose either: Static registration: Invoke RegisterTagHandler. Dynamic library loading: Make TagHandlers into libraries and create handler file.	page 123
	Programmer	If you will use a page builder service:
	Define the page builder service object.	page 74
	Write page builder methods.	page 138
	Use WindowConverter class to take advantage of existing iPlanet UDS windows.	page 140 page 143
	Define the HandleRequest method.	

Summary of Steps for Building an iPlanet UDS Web Application

Responsible Person	Step Description	See
Programmer	If you require session management:	
	Initialize Session Management attributes.	page 171
	Assign session properties.	
	Subclass HttpSession class.	
	Modify URL links for session management.	page 182
Programmer	Design Administration Window and write initialization code.	page 212
Programmer and Author	Test and modify as desired.	page 94
Programmer and Author	Partition and deploy the application.	page 187

Suggested Project Hierarchy

When you build a new iPlanet UDS Web application, you use projects supplied by WebEnterprise in addition to projects you have defined for your application. Every iPlanet UDS Web application requires the HTTP library as a supplier plan. The other three predefined WebEnterprise projects, HTML, HTMLWindow, and HTMLSQL (the HTML projects) are optional supplier plans.

Because you can design an iPlanet UDS Web application using a variety of approaches, there is no one required project hierarchy. Rather, some projects are required supplier plans and some are required under certain circumstances.

All iPlanet UDS Web applications require a Web access project. Either a tag handler project or a page builder project is also required, to generate Web pages.

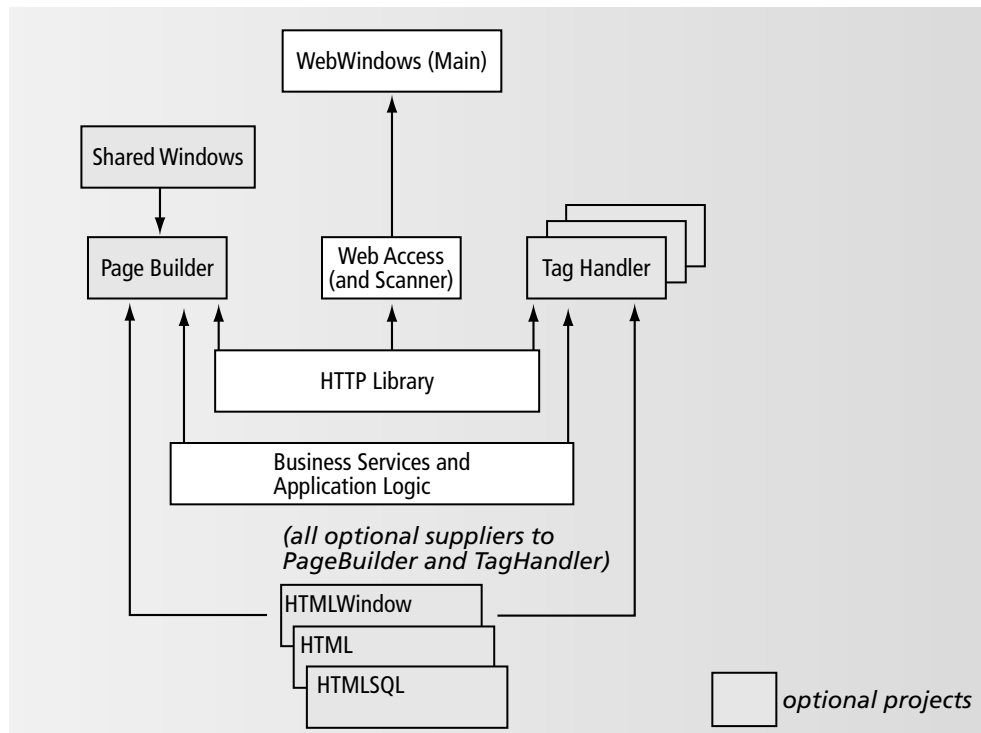
Required supplier plans While the projects that an application uses vary depending upon the application requirements, the following rules apply to *all* iPlanet UDS Web applications:

- The Web access project (a custom subclass of HTTPAccess) must be a supplier project to the main project of the iPlanet UDS application.

- The HTTP library must be a supplier plan to the following projects, if the projects are used: the page builder project, the Web access project, and the project containing tag handler classes.
- The business services project(s) must be a supplier plan to the following projects, if the projects are used: the page builder project and the tag handler project.

These relationships are shown in **Figure 3-1**, which shows how the user-defined projects and iPlanet UDS projects relate.

Figure 3-1 Suggested Project Hierarchy for an iPlanet UDS Web Application



➤ **To include a project as a supplier plan**

1. In the Project Workshop for the project requiring the supplier plan, choose File > Supplier Plans.
2. Double-click the desired project to add it as a supplier plan.

An iPlanet UDS Web application must have at least one page builder project or tag handler project, based on the following rules:

- If an application constructs *all its pages* using the scanner service, the page builder project is not needed.
- If an application constructs *all its pages* using the page builder service, the tag handler project is not needed.

While more applications may use the scanner service to generate pages, it is not unreasonable for an application to use both services. In fact, either service can invoke the other service to obtain a page or a portion of a page. So it is not uncommon for an iPlanet UDS Web application to use both services, and therefore require both projects.

When to Include the Optional HTML Projects

iPlanet UDS WebEnterprise provides three utility projects, called HTML, HTMLWindow, and HTMLSQL, that provide a number of different classes for you to use when generating pages. These HTML projects can be supplier plans to either the page builder project or the tag handler project.

The HTML Project

Include the HTML project if you wish to use HTML classes, rather than typing HTML directly in your code, to create Web pages. The HTML classes are directly based on HTML format elements, so it is easy to learn the class names and attributes, particularly if you are already familiar with HTML.

The benefits of using the HTML classes are described in the iPlanet UDS online Help. For more information and a list of the classes, see the iPlanet UDS online Help.

The HTMLWindow Project

Include the HTMLWindow project if you have existing iPlanet UDS windows that you wish to convert into Web pages, or if you wish to use the Window Workshop to create new windows to be converted to Web pages. Using the HTMLWindow project allows you to:

- build Web pages by defining and converting iPlanet UDS windows
- share the underlying program logic that loads and manipulates the data in the Web pages and iPlanet UDS windows

You use the `WindowConverter` class in the `HTMLWindow` project to convert the layout of an iPlanet UDS window to a Web page. Because both the window layout and program logic can be shared by the Web and the iPlanet UDS applications, these windows can be thought of as “shared windows.”

NOTE Due to fundamental differences in the interfaces, you cannot directly convert all the layout features and logic of an iPlanet UDS window to a Web page—some additional coding is required.

Only include the `HTMLWindow` project if you will use it. Because `HTMLWindow` includes the `Display` library as a supplier plan, you incur additional overhead when you include the `HTMLWindow` project.

For more information on using the `HTMLWindow` project, refer to the iPlanet UDS online Help. For more information on using the `WindowConverter` class, refer to [“Adapting iPlanet UDS Windows with WindowConverter” on page 146](#).

The HTMLSQL Project

Include the `HTMLSQL` project if you would like to embed in a Web page the results of queries to a relational database. The `SQLConverter` class in the `HTMLSQL` project allows you to display results of stored or ad hoc queries on a Web page.

Only include the `HTMLSQL` project if you will use it. Because `HTMLSQL` includes the `GenericDBMS` library as a supplier plan, you incur additional overhead when you include `HTMLSQL`. You need not necessarily include `HTMLSQL` in order to interact with a database, because your existing iPlanet UDS application may already adequately define all database interactions.

See the iPlanet UDS online Help for an explanation of using the `SQLConverter` class to include database data on a Web page.

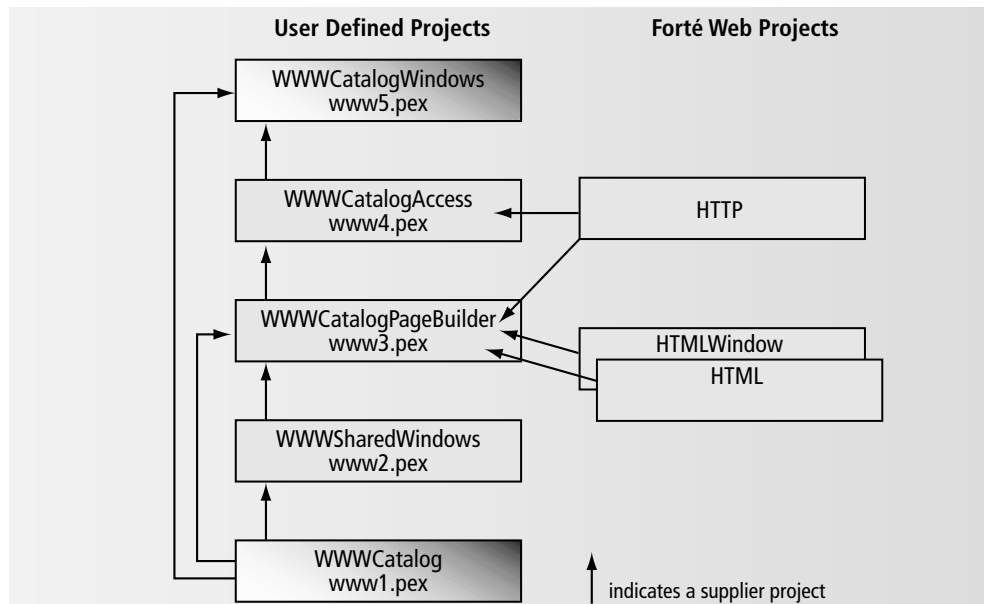
Project Structure for the SoftWear Application

If you are adding Web access to an existing iPlanet UDS application, you will create some new projects and modify the existing client application somewhat. The `SoftWear` application demonstrates this; the two projects `WWWCatalogWindows` and `WWWCatalog` represent the “existing” iPlanet UDS application with no Web

interface (the WWW prefixes of the names are simply to differentiate the projects when you import them for your use). The WWWCatalog project contains the CatalogService service object, with which the WWWAccessService service object communicates on behalf of Web clients.

Figure 3-2 shows the project names used by the SoftWear example (along with the .pex files used to import the projects).

Figure 3-2 Project Hierarchy for the SoftWear Application



Also note that two user-defined projects are suppliers to the page builder project: WWWSharedWindows supplies windows to be used by WindowConverter and WWWCatalog supplies the business service object, which supplies data to the page builder service object.

Defining a Web Access Project and Service

The Web access project is used to define the Web access service object, which is required to add Web access to an iPlanet UDS application.

► **To create a Web access project**

1. Open a workspace that contains your existing iPlanet UDS Web application.
2. Create a new project, your Web access project, using the Repository Workshop.
3. Include the HTTP library as a supplier plan.

The HTTP library is a required supplier plan for every iPlanet UDS Web application.

4. Create a subclass of HTTPAccess.

Do not change any of the class runtime properties.

Creating a Web Access Service Object

The role of the Web access service object is to accept and delegate Web requests and to return Web responses. Whenever a Web access service object is enabled by the EnableAccess method, it registers the following information with the fortectgi program or the iPlanet UDS Web server plug-in:

- the service name by which the Web access service object is known
- the port number where the service object will listen for Web requests sent by fortectgi or the plug-in

The fortectgi program maintains a file called fortectgi.dat with one row for each currently registered Web access service object. The fortectgi and iPlanet UDS Web server plug-ins use the fortectgi.dat file to route incoming browser requests to the appropriate Web access service object.

► **To define a Web access service object**

1. Using the Project Workshop for the Web access project you created above, create a new service object.
2. Check the TOOL class radio button and enter the new subclass of HTTPAccess as the base class for the service object.

You can leave the dialog duration set to “Session.” In fact, due to the nature of the HTTP protocol, all communications between the Web access service object and the Web server are equivalent to message dialog duration, so it does not matter what you set the dialog duration to.

The Web access service object cannot be replicated, so do not check either Failover or Load Balancing.

In the SoftWear example the Web access service object is called WWWAccessService.

3. Register the Service.

Every Web access service must register with the fortectgi program or the iPlanet UDS Web Server Plug-in so that the Web access service can receive requests from fortectgi or the plug-in. Registration can be automatic or manual (as described in the section titled *“Setup Options for fortectgi and iPlanet UDS Plug-ins”* on page 200).

NOTE In WebEnterprise, autoregistration requires the use of the fortectgi program, even if your application will take advantage of the new iPlanet UDS NSAPI plug-in. For more information, refer to *“Choosing between fortectgi and the iPlanet UDS NSAPI Plug-In”* on page 199.

Enabling the Web Access Service Object

The Web access service object must register with the fortectgi program to allow Web clients to access the iPlanet UDS Web application. After the service object is registered, fortectgi can send it Web requests.

EnableAccess method The Web access service object uses the EnableAccess method of the HTTPAccess class to register with the fortectgi program, passing fortectgi the following parameters:

- the service name: the name by which the Web access service object is known
- the port number: the port on which the service object will listen. You must enter a port number; there is no default port
- the URL for the fortectgi with which the service object is registering
- an optional plug-in parameter, used if you are using an iPlanet UDS plug-in rather than fortectgi

If this optional parameter is specified, then the fortectgi program is used *only* for registration, and the specified iPlanet UDS plug-in is used for all other Web server-to-iPlanet UDS communications.

When EnableAccess is invoked, the Web access service becomes available immediately.

In the following example, a service object known as My Service will listen at port 1234, and is registering with a fortectgi at the URL `http://www.forte.com/`:

```
self.EnableAccess(serviceName = 'MyService',  
                 servicePort = 1234, URLForFortecGI = 'http://www.forte.com/');
```

This information is entered into the fortectgi data file (by default called fortectgi.dat) for use by fortectgi. For more information on the EnableAccess method, refer to the iPlanet UDS online Help.

Autoregistration and manual registration Autoregistration using fortectgi is the simplest way to register a Web access service. You have other registration options, including using manual registration with an iPlanet UDS Web Server Plug-in. For more information about these registration options, refer to [“Choosing between fortectgi and the iPlanet UDS NSAPI Plug-In” on page 199](#).

You can invoke EnableAccess in two general ways, as described below.

Using a Start Method to Enable Access

One way to enable access is to include the `EnableAccess` method in the designated start method for your application. Then, when your iPlanet UDS application starts, the service object is automatically registered and Web access enabled. You should invoke `EnableAccess` very early in the start method, before the event loop.

Advantages and disadvantages The advantage to this approach is that it requires essentially no extra coding and you know that Web access is enabled whenever the iPlanet UDS application is running. This may be a good approach for when you are initially building and testing a new iPlanet UDS Web application. The disadvantage is that you cannot control Web client access independently from iPlanet UDS client access—if the application is running, clients of both types can use it.

If you use the start method to enable access, you can use either method of registration. That is, you can invoke `EnableAccess` with two parameters (for manual registration) or with three parameters (for autoregistration).

CAUTION Invoking `EnableAccess` in the `Init` method can be somewhat tricky; we recommend that you do not invoke `EnableAccess` in the `Init` method for the `HTTPAccess` subclass, unless you are certain you understand the implications of doing so.

Using an Administration Window to Enable Access

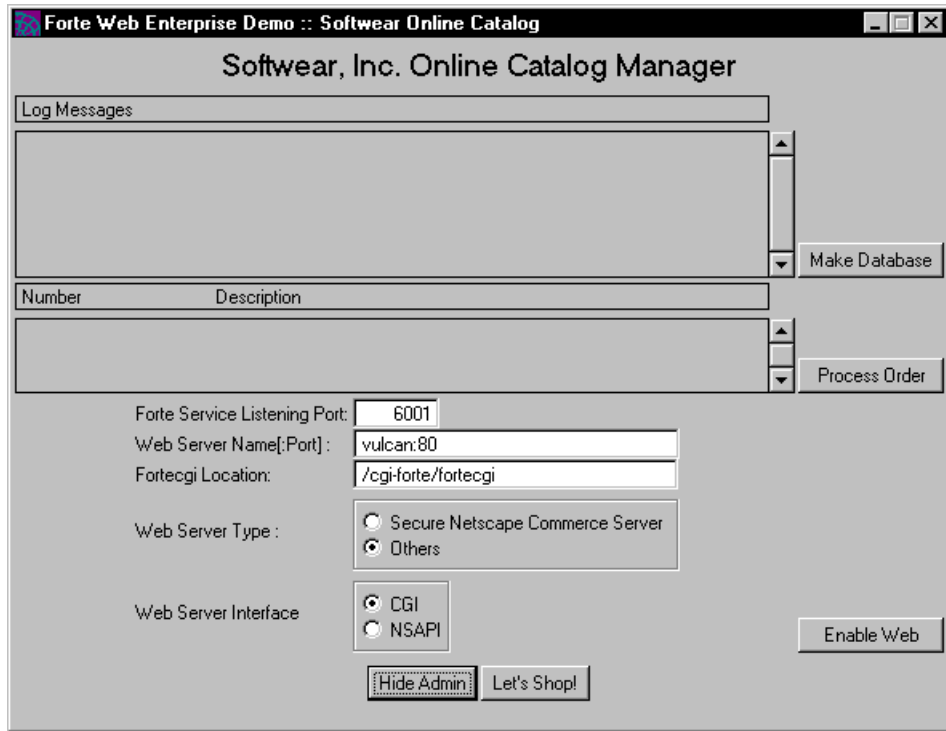
You can use an administration window to explicitly enable and disable Web access. This approach gives you more flexibility than using a Start method. In your iPlanet UDS application, create a window class and define a window to use to enable and disable access. This window is strictly an iPlanet UDS window—it is not a Web page and Web clients never see it.

Over time you can modify the window to add other initialization tasks. For additional information on other initialization tasks that might be performed in this window, see [“Initialization Tasks” on page 212](#).

Advantages and disadvantages This approach gives you more control over when you enable and disable access; in particular, it allows you to enable Web client access independently from iPlanet UDS client access.

The SoftWear application uses an administration window called Main Window to perform a few simple tasks in addition to enabling and disabling access. Main Window is shown in [Figure 3-3](#).

Figure 3-3 Example Administration Window



In fact, the `Display` method of the `MainWindow` class provides a number of administrative functions. Since `SoftWear` is a demonstration application, the `Display` method includes provisions for using different Web servers. If the Web server is a non-secure server, it invokes `EnableAccess` using autoregistration; if the Web server is secure, then it invokes `EnableAccess` assuming manual registration. Note that in the latter case, instead of using the HTTP protocol, the secure version of HTTP is invoked (HTTPS). The `Display` method also provides support for either the `fortecgi` or `iPlanet UDS NSAPI` plug-in, as indicated by the radio buttons in the administration window.

Also note that the label and function of one button changes from `Enable Web` to `Disable Web`, depending on which method was last invoked.

Disabling Web Access

To stop Web access to your iPlanet UDS application, you must de-register the Web access service object by invoking the `DisableAccess` method on `HTTPAccess` subclass.

For service objects that use autoregistration, the `DisableAccess` method de-registers the Web access service object with the `fortecgi` program and removes the service object from the `fortecgi` program's list of active Web access service objects.

For service objects that use manual registration, the `DisableAccess` method should be invoked *after* the file `fortecgi.dat` is edited to delete the row for that service object (see [“Manual Registration” on page 202](#)).

You typically disable the Web access service object just before the application or partition shuts down.

```
self.DisableAccess();
```

NOTE Depending upon your network configuration, the de-registration process may take a minute or two.

Defining a Scanner Service Object

The scanner service object is an optional service object whose purpose is to generate Web pages.

When the Web access service receives a page request (a URL), it first determines whether the request is for a page or a template. It then directs all requests that use the `templateName` parameter to the scanner service object (*scanner* for short) and all requests that use the `pageName` parameter to the page builder service object. When the request is for a template, the scanner uses the named template to generate the requested page, and returns it to the Web access service object to forward to the browser.

See [“About the HTML Scanner Service” on page 102](#) for more information on the scanner service object.

The scanner service object should always be created in the same project as your Web access service object (see [“Defining a Web Access Project and Service” on page 64](#)).

- **To create the scanner service object**
 1. Open the Project Workshop for your Web access project.
 2. Create a subclass of the `HTMLScanner` class.
 3. Create a service object based on this subclass.

Typically you will make this service object environment visible and message dialog duration. If you wish, you can mark this service object as replicated for load-balancing or failover.

4. Set default values for attributes of this service object.

You should set a value for the `DocumentRoot` attribute on your subclass of `HTMLScanner`. This attribute specifies a base directory where the scanner service object expects to find template files. For more information on this attribute, see [“The iPlanet UDS Document Root Directory” on page 210](#).

You only need to set the `HandlerConfigFile` attribute if you are using dynamic loading to load one or more libraries that contain tag handler class(es). For more information, see [“The Handler File” on page 209](#).

Defining a Web Page Builder Project

The Web page builder project contains the page builder service object and the page builder class, which defines one or more page building methods used to construct Web pages for an iPlanet UDS Web application. The use of this project and the page builder service is optional; you do not need it if all your pages are returned by one or more scanner services.

If you use this project and the page builder service object, you can replicate the page builder service object to improve performance for applications with many concurrent users. Web access service objects cannot be replicated, but their primary purpose is simply to pass Web requests and responses. Page builder service objects do more intensive processing and can be replicated (see [“Replicating Partitions” on page 192](#)).

For each Web page to be returned by the page builder service, you will write a page building method.

► **To create a Web page builder project**

1. Choose the New Project... command in the Repository Workshop.

The New Project dialog appears.

2. Enter the name of the new project. A name that includes PageBuilder is suggested.

The SoftWear example uses the name WWWCatalogPageBuilder for this project.

3. Include optional Web projects as suppliers to page builder project.

See *“When to Include the Optional HTML Projects”* on page 61 for a description of when you should include these projects.

4. Create a page builder class.

This class should inherit from Object. Make sure that the Distributed property is set to allowed, as you will use this class to create the page builder service object. A suggested name is PageBuilder or a name that ends with PageBuilder.

In the SoftWear example, this class is called `CatalogPageBuilder`.

Alternative ways to use the page builder class When used, the page builder class is typically the base class for a Web page builder service object that generates Web pages for the Web access service object. This approach has two benefits: your code is more modular and you can load balance the page builder service object for improved performance.

Strictly speaking, the page builder class is not required. One alternative is to write a “long” `handleRequest` method for the `HTTPAccess` subclass that returns all possible Web pages. In this case, you need not even create a page builder class.

Another alternative is to use a page builder class but no page builder service object. This page builder class would include several methods that construct individual Web pages. While this approach is more modular than the previous alternative, it does not offer potential performance gains that you can get by using a service object, because you cannot load balance a class.

Defining a Page Builder Service Object

The Web page builder service object builds specified Web pages on request from the Web access service object and passes the pages back to that service object.

You need only define this service if you require it to assemble one or more pages for your application. If your iPlanet UDS Web application will construct all its pages using templates and scanner services, you do not need to define a page builder service.

► To create a page builder service object

1. In the Project Workshop for your PageBuilder project, create a new Service Object, whose base `TOOL` class is the new page builder class.
2. If you want to replicate this service object for load balancing or failover, set the dialog duration to Message, and check the appropriate field under Replication Options in the Service Object Properties dialog.

Unlike the Web access service object, the page builder service can be replicated. You can replicate the page builder service for either load balancing or failover. Load balancing is desirable if you anticipate a high number of concurrent Web users. Failover is desirable if you are particularly concerned about availability and reliability of your iPlanet UDS Web application. For more information see [“Replicating Partitions” on page 192](#).

In the SoftWear example this service object is called `CatalogPageBuilderService`.

You must define methods for each page that you want to be built by this service object. The recommended approach is to write one page builder method to generate each Web page. For information on defining pages using a page builder, see [Chapter 6, “Creating Pages Using Page Builder Methods.”](#)

Defining a Shared Windows Project

You can add Web access to an existing iPlanet UDS application in which you have already defined a number of user windows. The benefit of doing so is that you can make these windows “dual-purpose”—that is, viewable by Web browser clients as well as iPlanet UDS window clients. To achieve this, you create a shared windows project, as described below.

The shared windows project contains windows defined in an iPlanet UDS application that you also want to use as Web pages. The shared windows project allows the iPlanet UDS and Web applications to share the appearance (design) of the windows, as well as some of the underlying processing of the data in the windows.

You do not need a shared windows project if you have no user windows defined in your iPlanet UDS application that you would like to use in Web pages. However, if you have any user windows (window classes) and you would like your Web application pages to “mirror” these windows, then you should create this project.

► To create the shared windows project

1. Choose the New Project... command in the Repository Workshop.

The New Project dialog appears.

2. Enter the name of the shared windows project.

The SoftWear example names this project WWWSharedWindows.

3. Identify the iPlanet UDS user windows that you want to also appear in Web pages and move these window classes to the shared windows project:

- a. Open the Project Workshop for the main project and the new Shared windows project.
- b. Drag and drop each class that you wish to share to the new project.
- c. Delete the classes from the main project.

4. Include the shared windows project in the main and page builder projects.

If you use a shared windows project, then you must include it as a supplier plan to both the main project and to the Web page builder project.

5. Modify or write methods on the windows.

Writing Methods for Shared Windows

For each of the shared windows you must determine how much modification is required to convert the iPlanet UDS window to a working Web page. Although the iPlanet UDS windows and Web pages can share much of the layout and logic, you will need to make some modifications to account for differences in functionality, behavior, and user expectations typical for each environment.

Use WindowConverter to convert layout You use methods on the WindowConverter class to convert iPlanet UDS windows or window elements to Web pages and page elements. These methods, however, only convert the layout. Then you must modify the underlying iPlanet UDS logic (using the iPlanet UDS application's business service object) to work for the Web interface. To see an example of a converted window, see [Figure 6-1 on page 141](#) and [Figure 6-2 on page 142](#).

For more information, refer to [“Differences between Window- and Web-Based Applications” on page 44](#), and [“Sharing Window Code with the Web Page” on page 151](#).

Planning Web Pages

This chapter describes general issues regarding the construction of Web pages in an iPlanet UDS Web application and contains the following topics:

- initial page design considerations
- choosing between the scanner service and the page builder service to generate each page
- special types of pages
- embedding graphics and generating URLs
- error handling
- testing

Once you have decided which service to use to generate pages, refer to either [Chapter 5, “Creating Pages Using Templates”](#) or to [Chapter 6, “Creating Pages Using Page Builder Methods.”](#) Your application can use both types of services.

Roles of the Web Author and Web Programmer

Designing an attractive and useful Web application often requires the talents of multiple persons— for example, a *content provider* to write HTML text, a layout/graphic arts *designer*, and possibly a *programmer* to provide content from external sources such as a database, stock feeds, and so on. In fact, to provide and maintain Web pages that are both visually appealing and up-to-date usually requires the ongoing contribution and cooperation of several persons.

The iPlanet UDS approach to building Web pages assumes that pages will include content that is derived programmatically. To distinguish who performs which steps in the following process, this manual uses the following terms and definitions:

author The person responsible for the *static content* of a page. The static content includes the layout and design of a page, any text that does not change, along with any embedded graphics.

programmer The person responsible for generating the *dynamic content* of a Web page—that is, content that is generated using a query that may contain user-entered criteria.

Designing an iPlanet UDS Web user interface that integrates with an existing iPlanet UDS application can entail a fair amount of planning. This chapter describes some tasks and considerations as you begin to design the Web interface. Some of these tasks require coordination between the Web interface page author and iPlanet UDS programmer.

Figure 4-1 Steps Involved in Building Web Pages

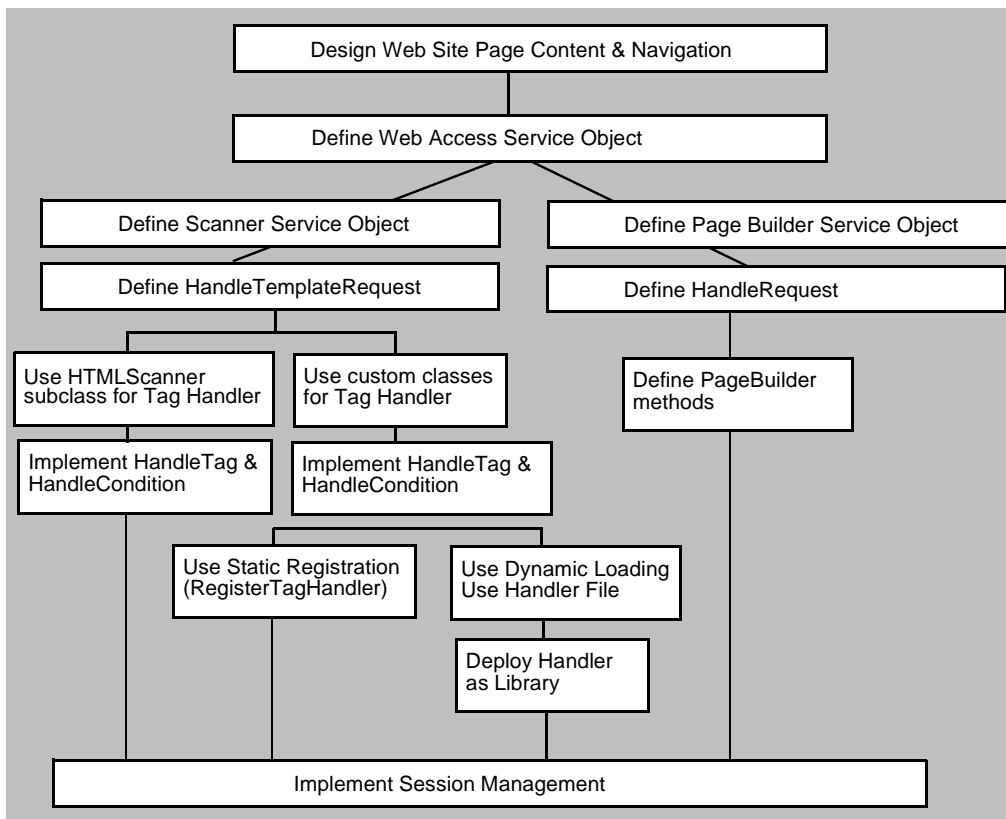


Figure 4-1 shows a chart of specific steps involved and choices you must make when planning and building an iPlanet UDS Web application. The following tasks are just some that you should consider:

- design a rough layout of the page hierarchy (for example, from the entry page to the pages where users query data)
- determine whether any portions of pages can be reused
- for each page, identify which dynamic data it should display
- for each page, identify which other pages it should link to
- identify which iPlanet UDS classes will provide the generated data
- identify which pages will be generated and returned by which service (a HTML scanner service or a page builder service)
- if using a scanner service:
 - choose whether to use custom classes or a subclass of HTMLScanner for the tag handler
 - identify variables to embed in the HTML pages to display the generated data
- identify what state information may be required for a user of the interface
- determine the optimal location, directory structure, and naming standard to use for the pages
- identify which pages require restricted access (a session)

Also see [“Summary of Steps for Creating an iPlanet UDS Web Application”](#) on [page 57](#) for a more comprehensive list of steps.

Initial Decisions About Pages

As you start to plan an iPlanet UDS Web application, whether it is an entirely new application or one based on an existing iPlanet UDS application, you face a number of planning tasks and decisions.

A fundamental issue is how an end user navigates through an application. Navigation design strategies for Web interfaces differ substantially from those for iPlanet UDS client interfaces. From a user's perspective, a Web interface is often easier and more flexible to use; from a developer's perspective, it differs in some notable ways from an iPlanet UDS client window interface. If your application has both types of interfaces, you must consider when both interfaces can use the same approach, and when they will diverge due to the differences in the technology.

Designing iPlanet UDS client windows is described in other iPlanet UDS documentation. This chapter describes some basic considerations for implementing iPlanet UDS Web interfaces. However, it does not attempt to provide a comprehensive discussion of designing a Web application. For example, should you design short, simple Web pages, or longer multi-part Web pages, and what links should you embed on a given page to suggest navigation paths.

Static and Dynamic Web Pages

In an iPlanet UDS Web application, every Web page can be categorized as either a static or a dynamic Web page. Many applications consist of both types of pages, and how you design your application depends upon what types of pages you must include.

Static Web pages *Static* pages have a fixed content that does not change at runtime. The page's exact and complete content is known when the page is defined by its author. An example is a press release or announcement. Another example is the entry point page for the SoftWear application, shown in [Figure 4-2 on page 84](#); it is the starting point for Web clients of the SoftWear application.

Dynamic Web pages *Dynamic* pages are pages whose entire or partial content is generated at runtime, when the page is constructed. In addition, the exact content may be a result of the Web user's particular criteria. A typical example is a Web page that is returned as a result of a search: for example, for all items out of stock or for all surnames beginning with "ST." Often dynamic pages can provide users the exact information desired more directly than static pages.

Both the page builder service and the scanner service can generate dynamic Web pages. Both services use the parameter values in the `HttpRequest` object to construct the requested Web page. For example, in the `SoftWear` application, both the `BuildProdPage` and `AddToBag` methods use the product code to identify which product to display or add:

```
detail : ProductDetail = CatalogService.GetProductDetail
    (request.FindNameValue('Code').value);
```

Your iPlanet UDS Web interface can build URLs to use as links that pass parameters in the `HttpRequest` object and specify which pages, with which values, the user is requesting. In turn, the scanner or page builder service uses the parameter values to construct the actual page requested.

Using the iPlanet UDS HTML Projects to Create Pages

iPlanet UDS provides three optional projects, called the HTML projects, to help you construct Web pages in your `TOOL` code. You can use these projects with templates or page builder methods, although you are more likely to use them in page builder methods.

HTML Project	Description	See
HTML	Provides classes that correspond directly to HTML elements. You set the class's attribute values to specify the element's attribute values.	iPlanet UDS online Help
HTMLWindow	Provides a class that allows you to convert an iPlanet UDS window into an HTML document (that is, a Web page). NOTE: Not all widgets are converted. See the iPlanet UDS online Help.	iPlanet UDS online Help
HTMLSQL	Provides a class that allows you to create a table on the Web page from a SQL query.	iPlanet UDS online Help

If you use the HTML projects, iPlanet UDS provides syntax checking for the TOOL code. You can use the iPlanet UDS Window Workshop to design a new Web page, take advantage of its visual layout tools, and then convert the window into a Web page. Or you can convert an existing iPlanet UDS client window into a Web page.

Determining which Services will Provide Pages

All pages returned by an iPlanet UDS Web application are generated by either a scanner service or a page builder service. An iPlanet UDS Web application must have at least one scanner or page builder service; a more complex application might have multiple services or a combination of both.

In many ways these two services are parallel: both services generate Web pages (HTML documents). The scanner service uses documents called *templates* to generate Web pages, and is called by the `HandleTemplateRequest` method. The page builder uses TOOL code (page builder methods) to generate Web pages, and is called by the `HandleRequest` method. Which service and method are invoked depends upon whether the URL contains the string “pageName” or “templateName.”

When you design and build an iPlanet UDS Web application, you must decide whether to use a scanner service, a page builder service, or some combination of both, to return pages to the Web browser. You should consider a number of factors, described in this section.

Whether you use a page builder service or a scanner service to return pages, you can replicate either type of service for purposes of load balancing or failover.

Using a scanner If you are designing a new application, you may find it easier to design pages using iPlanet UDS templates along with one or more scanner services. Some of the advantages of using templates follow:

- Web page designers can use familiar tools to create Web browser user interfaces.
- You can use the WYSIWYG (what you see is what you get) features of an HTML editor to design the page (layout, font, tables, and so on).
- You can use other features of the HTML editor package to manage your HTML files.
- You can view the final appearance of your Web page without running iPlanet UDS.

- You can change the text of an underlying HTML template and the changes are picked up in *real time*—you need not shut down or restart your application. You simply edit the template and save the newer template file in the location where the scanner service expects to find it.

Note that you do need to write iPlanet UDS code when using templates, but you do not use code to generate the *entire* page—just the dynamic portion of the page.

Using a page builder When you use a page builder service (or page builder methods without a page builder service) then you actually construct each Web page using TOOL code based on the HTML classes. Specifically, you use the method workshop to write a page builder method to generate each distinct page. You do not use an HTML editor, so you do not get the WYSIWYG advantages. And, you use TOOL code to generate the entire page, including static text portions as well as dynamic data.

If you are adding Web access to an existing iPlanet UDS application, you can convert existing windows (defined as iPlanet UDS window classes) to Web pages. You can embed these converted windows into templates to be used by a scanner service, or you can use the converted windows with a page builder service.

When to define which method Depending upon how the pages in your iPlanet UDS Web interface are generated, you will need to define the `handleRequest` method, the `handleTemplateRequest` method, or both. You define these methods to provide the appropriate handling for each of the unique pages in your application.

A URL that requests a page from a scanner service identifies the page using a “`templateName`” parameter. Then the Web access service automatically forwards the browser’s request to the appropriate scanner service. A URL that requests a page from a page builder service identifies the page using a “`pageName`” parameter; the Web access service automatically forwards the browser’s request to the appropriate page builder service.

- You must define a `handleRequest` method in your subclass of `HTTPAccess` if you have any pages that are generated by a page builder service object.

If the URL has a `PageName` parameter, it is a page builder request, and the Web access service object invokes the `handleRequest` method. **Code Example 4-1** shows a URL that requests the `AddUser` page from the page builder service:

Code Example 4-1 URL that Requests a Page from a Page Builder Service

```
http://www.forte.com/cgi-forte/fortecgi?serviceName=fortedemo
&pageName=AddUser&userName=Ann+Jones&userID=1
```

- You must define a `HandleTemplateRequest` in your subclass of `HTTPAccess` if you have any pages that are generated from iPlanet UDS HTML templates.

If the URL has a `TemplateName` parameter, it is a template request, and the Web access service object invokes the `HandleTemplateRequest` method. [Code Example 4-2](#) shows a URL that requests the `AddUsr` template from the scanner service:

Code Example 4-2 URL that Requests a Page from a Scanner Service

```
http://www.forte.com/cgi-forte/fortecgi?serviceName=fordedemo
&templateName=AccountMaint/Addusr.htm&userName=Ann+Jones&userID=1
```

In this URL, the template name `Addusr.htm` is preceded by its path, which is relative to the `DocumentRoot` specified for the scanner service object.

For more information on overriding these two methods, refer to [“Defining the HandleTemplateRequest Method” on page 111](#) and [“Defining the HandleRequest Method” on page 143](#).

Identifying Session Management Requirements

The WebEnterprise session management features provide mechanisms for controlling access to individual pages or groups of pages in a Web application, for creating sessions for individual Web browser interactions, and for maintaining state information for individual sessions. If your application contains any sensitive information or should be restricted in access in any way, then you will probably want to use the session management features, which are described in detail in [Chapter 7, “Using Session Management.”](#)

Using session management with static pages If you use session management and your application has any static pages, those pages must be located along with the dynamic pages—this location is determined by the attribute `DocumentRoot` (on your subclass of HTML Scanner). This location allows the scanner service object to return the static pages with session management implemented. This is the one circumstance in which you would use an iPlanet UDS service object to return a static page.

If a given static Web page does not require session management, you do not need to use an iPlanet UDS service to build or return the page, and you can simply store the static page on the Web server. However, this applies only to static Web pages that you want to be accessible to any Web user.

Page to create sessions If you use session management you should plan a page that gathers information required to create a session. The location for this page is automatically used by an application— users are automatically returned that page if and when they need to create a new session. For more information about how to use this page automatically when necessary, see [“Setting the SessionCreationURL” on page 172](#).

Special Purpose Pages

This section describes two types of pages that you might find useful in your iPlanet UDS Web applications. These pages are:

- entry point page
- session creation page

Entry Point Page

Your application should include an attractive page for a starting point for users to begin the program, and you should provide an easy way for Web users to access this page. You might designate this page as a “main page” or a “home page,” depending upon your Web application’s hierarchy of information. We use the term *entry point* Web page to describe the first page your users will see when accessing your application.

The entry point page for the SoftWear application is shown in [Figure 4-2](#).

Figure 4-2 Entry Point Web Page for the SoftWear Application

Getting to the entry point page To reach the entry point Web page, Web users typically do one of the following:

- Enter a URL for the entry point page. This URL must be made known to your users, so it should be relatively simple to type in, ideally without any parameters.
- Use a hypertext link on another Web page to jump to the entry point page.

The link can appear on any Web page and can be a graphic (such as a push button) or text (such highlighted text with a link containing the URL). See “Using Links” on page 85 for how to define a URL.

Contents of the entry point page The entry point Web page contains one or more links that the Web user selects to begin to use the iPlanet UDS Web application. For example, the entry point page for the SoftWear example contains one button that says “Let’s Shop!” A user who prefers to quit the application rather than shop can simply enter a new URL or exit the Web browser.

Entry point pages can be static Often a static page is sufficient for the entry point page, because the structure of your iPlanet UDS Web application is less likely to change than the data in it, and the first page might simply offer a few choices to the Web user. If you use a static page, you need not use a service object to provide it; instead, locate it with your application's other static Web pages. The EasyWeb and SoftWear examples both use static pages named `homepage.htm` that are stored on the Web server.

Session Creation Page

If you use session management in your application, and you require any input data from users in order to create a valid session (such as a user ID and password), then you should define a page where users can enter this information. This page is called a *session creation page* (or *login page*); it may be the same page as your entry point page, it may be the next (expected) page that users would go to, or it might be any page to which you can code a link. You can use a template for this page.

The links in your application should help the user get to the login page easily. You can also set the `SessionCreationURL` attribute to the URL for this page. By setting this attribute, users are automatically redirected to the login page if they attempt to access a page for which they have inadequate permission (specifically, the page has the property `SESSION_REQUIRED` and the user does not have a valid session). The user then has the option of entering the information required to start a new session.

For an example of setting this attribute, see [“Setting the SessionCreationURL” on page 172](#). For more information on the session management features, and how to use them, see [Chapter 7, “Using Session Management.”](#)

Using Links

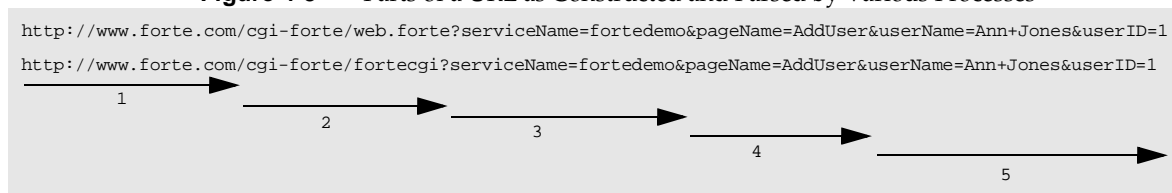
Web pages are characterized by the use of links that allow Web users to jump quickly from page to page. To add links to a Web page, you embed the appropriate URLs in the appropriate places in the Web page. You can add links to straight HTML paragraphs or text using the HTML anchor markup tag `<A>`. You also add links to HTML forms or pages that are converted from iPlanet UDS windows, as a part of preparing the window for conversion (see [“Using the WindowConverter Class” on page 140](#)).

Format of URLs used by WebEnterprise

A URL (universal resource locator) is a text string that can be considered an address for a Web page. Typically Web users enter relatively simple URLs in order to access a site or application; then the application generates more complicated URLs that contain more specific information used to return subsequent pages to the user.

A URL, therefore, can consist of several parts, each part used by a “player” in the Web application (such as a Web server process or an iPlanet UDS service object), which in turn passes on any remaining information to the next player. This sequence and use of parts is shown in [Figure 4-3](#).

Figure 4-3 Parts of a URL as Constructed and Parsed by Various Processes



Reading from left to right, the “numbered” parts of the URL are used as follows:

1 The Web protocol and Web Server Handled by DNS, identifies the protocol (such as HTTP, HTTPS, or FTP) and a unique Web server on the Internet.

2 The interface on the specified Web server Handled by the Web server, directs the request to the selected CGI or plug-in interface. These values are case-sensitive on UNIX systems.

(The following parameters can appear in any order, but typically appear in this order.)

3 The serviceName parameter Identifies the name of the Web access service. This part is used by the interface (whether fortecgi or the iPlanet UDS NSAPI plug-in). The fortecgi program or plug-in looks up the specified service name in the fortecgi.dat file.

If the service name appears in the fortecgi.dat file, the Web access service object is enabled and registered; fortecgi or fortensapi forwards the Web request (the URL) to the named service.

If the service name does not appear in the file, then it is either unknown or not currently enabled. In this case, fortecgi or fortensapi returns a Request Failure error to the Web user with the message “Service not found.”

4 The `pageName` or `templateName` parameter Used by the Web access service object, this parameter identifies which page or template is requested. This value is required and, on UNIX systems, is case-sensitive.

Depending upon the value of part 4, the Web access service object passes the current request to either the page builder service object or the scanner service object.

5 Remaining parameters Used by the page builder or scanner service object in order to build the exact Web page requested by the user.

The `$$FORTE.ExecURL` Variable

The WebEnterprise global variable `$$FORTE.ExecURL` specifies the path to your Web server. You must use the `$$FORTE.ExecURL` variable in all links in your Web application; see [“Using the `\$\$FORTE.ExecURL` Variable in URLs” on page 183](#).

How `$$FORTE.ExecURL` is used WebEnterprise uses the value of `$$FORTE.ExecURL` to expand all generated URLs (including links in generated pages) to include the correct domain and host name for the web server, and the location of either the `fortecgi` program or the Forte plug-in.

An example of this expansion follows. A URL embedded in a template might appear as follows:

```
<a href="$$FORTE.ExecURL?
ServiceName=ShopCartService&TemplateName=/Shopping/products.htm&
category=$$currCategory.ID" target="main">$$currCategory.Name</a>
```

HTML file: `frcontent.htm`

And the actual URL link would be expanded to the following:

```
<a href="http://www.forte.com/cgi-forte?
ServiceName=ShopCartService&TemplateName=/Shopping/products.htm&
category=$$currCategory.ID" target="main">$$currCategory.Name</a>
```

How \$\$FORTE.ExecURL is set The value for \$\$FORTE.ExecURL is set in one of the following ways:

- If you use automatic registration for the Web access service object, the value is derived from the EnableAccess method of HTTPAccess.
 - If *only* the URLForForteCGI input parameter is specified, that value is taken for \$\$FORTE.ExecURL.
 - If the PluginURL input parameter is *also* specified, that value is used.
- If you use manual registration for the Web access service object, then you must explicitly set the value using one of the following two methods:
 - If you are using an iPlanet UDS plug-in, invoke SetPlugInURL method on your HTMLScanner subclass.
 - If you are not using an iPlanet UDS plug-in, invoke SetExecURL method on your HTMLScanner subclass.

Constructing Links

► To add a link to a Web page

1. Choose or create the page on which you will include the link.
2. Include an HTML anchor element, with which you associate a URL.
3. In the URL, include the parameters that identify the service, page name, and any page parameters to fully specify the page. You must also use (or substitute if appropriate) the variable \$\$FORTE.ExecURL in the first portion of the URL.

```
<a href="$$FORTE.ExecURL?
ServiceName=ShopCartService;TemplateName=/Shopping/products.htm;
category=$$currCategory.ID" target="main">$$currCategory.Name<
/a></
```

HTML file: frcontent.htm

(To see how this link, which is embedded in an ITERATE tag, appears on the Web page, refer to [Figure 5-2 on page 99](#); the left-hand frame contains three product categories, each linking to its own page.)

Link example The following code shows the construction of dynamic links. It constructs a definition list showing product names and short descriptions of each product. The product name has an anchor, or link, to a more detailed page of information for the product. The code shows how to build that link dynamically.

Because they are the same for all products, the `serviceName`, `pageName`, and `shopperID` variables are appended to the anchor outside the “product” loop. Then for each product, the anchor is completed by appending the unique code for each product.

This code could also use the `Encode` method of the `HTTPResponse` class to ensure that the URL does not contain any spaces or control characters. See [“Constructing a Result Set” on page 118](#) for an example of `Encode`.

```

linktd : TextData = self.CGIURL.Clone(deep=TRUE);
linktd.concat('?serviceName=').concat(SERVICE_NAME);
linktd.concat('&pageName=prodDetail');
linktd.concat('&shopperid=').concat(shopperID);
-- Build the basic page structure.
html : HTML = new;
head : HTMLHead = new;
body : HTMLBody = new;
head.Add(HTMLTitle(Text='SoftWear Inc. :: Product List'));
html.Add(head);
. . .[continues]
-- Next, build a list of products. This code uses the
-- HTML list features (<DL>, <DT>, and <DD>).
dl : HTMLDl = new;
dt : HTMLDt;
dd : HTMLDd;
for p in CatalogService.GetProductList() do
  -- For each product, use its name as <DT>
  -- and its short description as <DD>.
  -- Also, put in the rest of the link needed
  -- to go to the ProdDetail page.
  dt = new;
  prodLink : TextData = new(value=linktd.value);
  prodLink.concat('&Code=').concat(p.Code);
  dt.Add(HTMLDt(Text=p.Name, Href=prodLink.value));
  dl.Add(dt);
  dd = new;
  dd.Add(p.ShortDescription);
  dl.Add(dd);
end for;
body.Add(dl);

```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder
 • **Method:** BuildProdList

Using Images and Graphics

Most Web pages include a variety of graphics to make them more useful as well as appealing. You add graphics to a Web page just like any other HTML element. The following code adds the company's logo.

```
html : HTHtml = new;
head : HTHead = new;
body : HTBody = new;

head.Add(HTTitle(Text='Softwear Inc. :: Product List'));
html.Add(head);

-- Build the content of page.
-- First, a image for the company logo.
body.Add(self.CompanyLogo());
```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder
• **Method:** BuildProdList

The entire CompanyLogo method follows:

```
-- Create a IMG element with its SRC set to
-- %LocImage%tinylogo.gif.

center : HTCenter = new;
img : HTImg = new;
img.Src = self.GetImageFile('tinylogo.gif');
img.Alt = 'SoftWear Inc.';
center.Add(img);

-- Create a line separator to make it look better.
center.Add(HTHr());

return center;
```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder • **Method:** CompanyLogo

In this example, the text "SoftWear Inc." is specified as text to use when printing the document or for display on browsers that disable graphics.

If you are returning binary data such as an image, you should use the `AssignBinaryResponse` method instead of the `AssignResponse` method. The `AssignBinaryResponse` method automatically sets the value of the `ContentType` attribute to the value of the `MIMETYPE` parameter of the `AssignBinaryResponse` method. Some typical settings for binary data include “image/gif,” “image/jpeg,” and “image/tiff.”

Location of the source graphics For a discussion of where to store graphic images and files that you use in Web pages, refer to [“Graphic, Image, and Binary Data Files” on page 211](#).

Error Handling

Web clients of iPlanet UDS applications are as likely to generate errors and exceptions as are iPlanet UDS clients. As the application processes page requests using the Web client’s input, it should also return helpful feedback to Web clients when they enter inadequate information.

Your existing iPlanet UDS application most likely already includes exception handling code that can be used or modified for your Web pages. For example, in the `SoftWear` application a shopper must enter an address before checking out. The method below checks that the address is complete. If information is missing, then an informative error message is returned to the shopper.

```
-- Check required information.
-- Raise exception if name or address is missing.

if self.Name=NIL or self.Name='' or
self.Address=NIL or self.Address='' or
self.City=NIL or self.City='' or
self.State=NIL or self.State='' or
self.Zip=NIL or self.Zip='' then
  ex : GenericException = new;
  ex.SetWithParams(SP_ER_USER, '%1 %2',
    TextData(value='Your personal information is
incomplete.'),
    TextData(value='Please fill in name and address.'));
  ex.DetectingMethod = 'Order::CheckOrder';
  ex.MethodLocation = 14;
  task.ErrorMgr.AddError(ex);
  raise ex;
end if;
```

Project: WWWCatalog • **Class:** Order • **Method:** ValidateOrder

The Default Web Error Page

The predefined methods `HandleRequest` and `HandleTemplateRequest` include exception handling for various user errors. If an error occurs, they return to the Web user a page with the iPlanet UDS error and an icon that reflects the severity of the error. An example of this page is shown in [Figure 4-4](#).

Figure 4-4 Example of the Default Error Page










You can substitute your own default error page by adding code similar to the following to the exception handler in your `HandleRequest` method:

```
. . . HandleRequest method . . .
exception
  when e: GenericException do
    errorResponse : HTTPResponse = new;
    -- Generate the error message page
    -- Assign the Web page to errorResponse
    return errorResponse;
```

Error Handling Icons

WebEnterprise contains standard icons that automatically appear on error pages for certain types of iPlanet UDS errors. If an iPlanet UDS exception of a particular type is raised, an HTML page is automatically generated that embeds the icon and the iPlanet UDS error. See the example in [Figure 4-4 on page 92](#).

The following table shows the icons and the iPlanet UDS exceptions to which they map, in increasing levels of severity.

Icon	Icon File	iPlanet UDS Exception Reason Code	Description
	warning.gif	SP_ER_WARNING SP_ER_INFORMATION	A warning or a minor warning. In either case the operation completed.
	user.gif	SP_ER_USER	An error in TOOL code.
	forte.gif	SP_ER_ERROR	An error in iPlanet UDS code.
	fatal.gif	SP_ER_FATAL	An error fatal to the program, causing iPlanet UDS to exit the partition in which the exception was raised.
	usage.gif	(none)	Generated by fortectgi and fortensapi, usually in response to an incomplete URL.
	request.gif	(none)	Generated by fortectgi and fortensapi, indicates a problem with the request, such as an unknown service name or a service that is unknown or currently disabled.
	runtime.gif	(none)	Generated by fortectgi and fortensapi, indicates a runtime error. Often due to a socket error.

Testing a Web Application

The design of a Web application is typically an iterative process. You make modifications and improvements to obtain your vision of the application. Like the design, testing is iterative, particularly because thorough testing requires testing at a variety of levels, from individual page layout and logic, to the coordination of pages, testing session and state management, and load testing.

As you develop individual pages, you can view and test some changes. However, you cannot see or test all pages and logic completely, until you run the page with the iPlanet UDS Web access service object, scanner or page builder service objects, and business services that are part of a deployed application. Only during such testing can you actually see how the actual data will appear incorporated into the page.

Initially you need not partition an application to test it. If you have fortectgi and a Web server installed on your machine, you can test from either the Repository or Project workshops.

► To test a Web application locally

1. In the Repository Workshop, select the project for your iPlanet UDS Web application that contains the start class, and click on the Run icon.

or:

In the Project Workshop for the project that contains the start class, click on the Run icon or select the Run command on the menu.

If you are using an administration window to enable access, then at this point iPlanet UDS clients can access the application, but Web clients cannot.

If you are invoking EnableAccess in your start method, then Web clients can also access the application.

2. If necessary, use the administration window to enable Web access.

NOTE Depending upon your network configuration and traffic, it may take a minute to enable access (and disable access).

Now Web clients can also access the iPlanet UDS application.

3. Use the Web browser to access the entry point page for the application.
4. Use the hypertext links to jump from page to page to test the flow of pages and data.

If you are unable to exit the iPlanet UDS application, you can use the Cancel Run command at any time to cancel execution. This command cancels the client partition for the application. The remote partitions continue to run; use the Stop Remote Partitions command to stop remote partitions.

You may also find additional helpful information in *A Guide to the iPlanet UDS Workshops*.

Creating Pages Using Templates

In an iPlanet UDS Web application, all Web pages that are returned to a Web client are generated either by a scanner service or by a page builder service. This chapter describes how to use the scanner service to return Web pages to Web clients.

The first part of this chapter is an overview of the concepts related to using templates, including iPlanet UDS tags, the scanner service object, tag handlers, and registering tag handlers.

The remainder of the chapter describes how to use these features and contains the following topics:

- designing templates and using iPlanet UDS tags
- writing tag handlers that call iPlanet UDS services to access the data required by a given template
- registering or dynamically loading the tag handlers with the scanner service
- reference for the FORTE HTML tags

For information about using the page builder service to return Web pages, refer to [Chapter 6, “Creating Pages Using Page Builder Methods.”](#)

About iPlanet UDS Templates

An iPlanet UDS *HTML template* is a particular type of HTML source file, a text file that contains standard HTML and iPlanet UDS HTML tags. It differs from a regular HTML file only in that it contains additional tags that must be handled by an iPlanet UDS scanner service. An iPlanet UDS Web application might consist almost entirely of server-parsed HTML templates. A new service, the scanner service, described in the next section, performs the parsing.

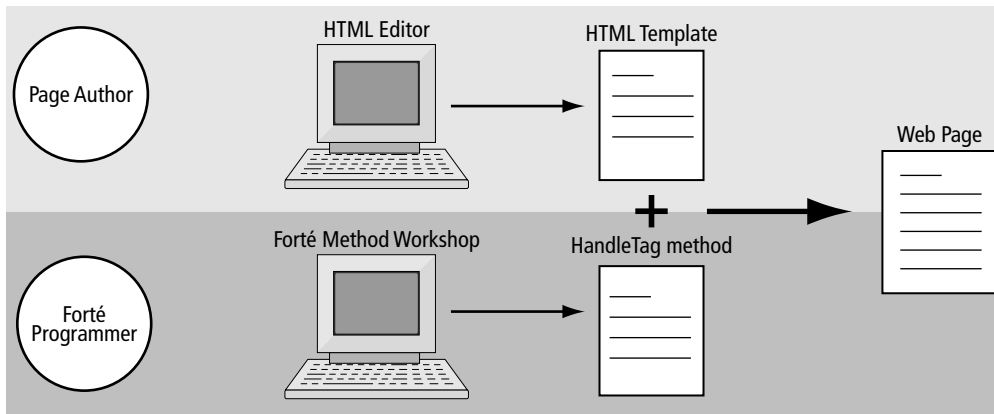
One benefit of HTML templates is that they allow different people to contribute to the Web page process, each using their own tools in their area of expertise. Specifically, page authors and iPlanet UDS programmers can coordinate to build Web pages that draw on data that is obtained and managed by the iPlanet UDS runtime system.

The author creates and modifies a template using a text editor or an HTML layout editor such as Microsoft FrontPage. When designing the template, the author includes special iPlanet UDS variables to represent content that will be provided by the programmer. A template can result in an entire page that is returned to a Web user, or a portion of a page, such as a frame or a running footer that is called by multiple pages.

For example, the author might write and design general text about corporate training classes, allowing space for a set of dates and locations. Then the programmer would add in the iPlanet UDS tags and variables required to generate the desired set of dates and locations, whenever the Web page is requested. Thus, the page is made up of both static and dynamic contents. But the design of the page remains under the control of the author.

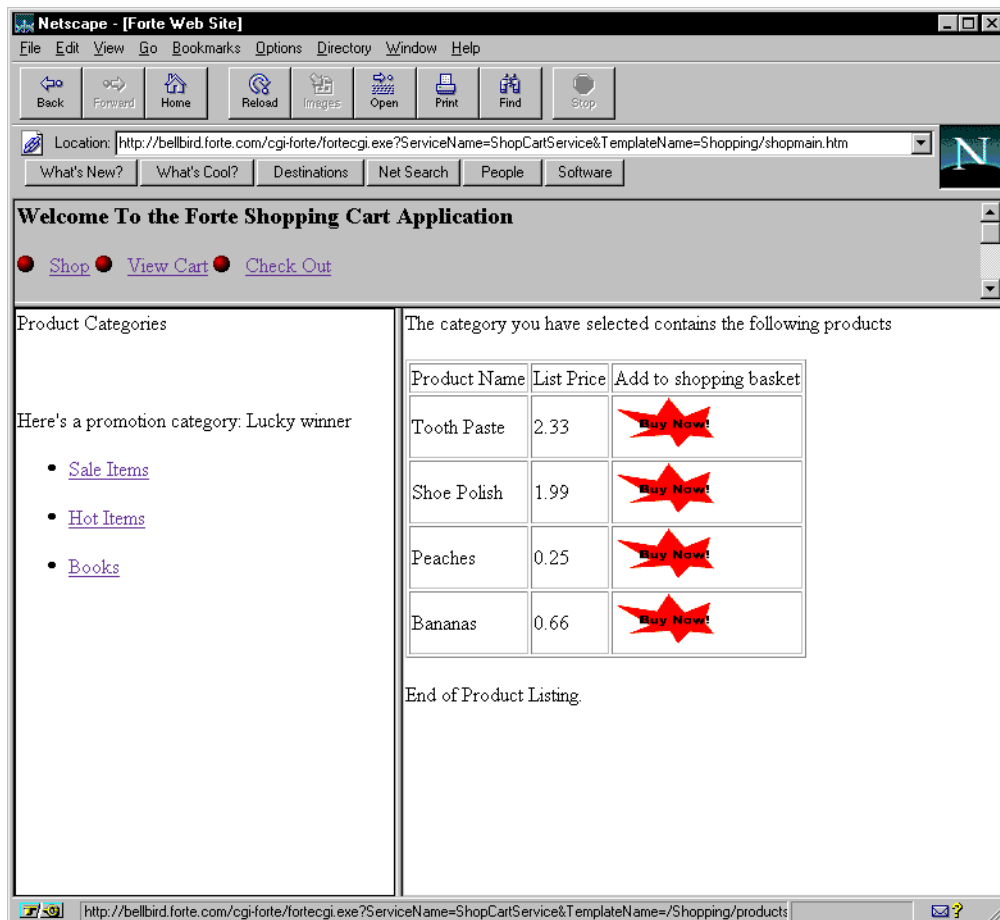
Figure 5-1 shows how the author and programmer coordinate to produce the template and iPlanet UDS code required to generate a dynamic Web page. First the author uses the HTML editor to create an HTML template. The programmer must add corresponding code in the iPlanet UDS application. At runtime, the template and the iPlanet UDS application produce a Web page with generated data.

Figure 5-1 Relationship of Template and TOOL Method to Create a Web Page



The next pages contain a simple example, taken from the iPlanet UDS sample application ShopCart, that demonstrates the use of a template. In this example, the template is for one of three frames in a Web page of the ShopCart application. The final page is shown in [Figure 5-2](#).

Figure 5-2 Final Web Page Showing Generated Data (List of Products)



The next pages show the following:

- the template (HTML file) for the lower right-hand frame, including the FORTE tags ([Figure 5-3](#))
- the TOOL code that handles the FORTE tags (in the text)

Figure 5-3 shows the template used to generate one frame in Figure 5-2. This template file, products.htm in the ShopCart example, at runtime generates a table of products.

Figure 5-3 Example HTML Template Showing FORTE Tags

```

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 2.0">
<title>Product Description</title>
<base target="main">
</head>

<body bgcolor="#FFFFFF">

<p>The category you have selected contains the following products</p>
<?Forte Execute CatalogHandler.GetProducts Resultset="ProductListRS">
<table border="1">
  <tr>
    <td>Product Name</td>
    <td>List Price< /td>
    <td>Add to shopping basket</td>
  </tr>
  <?forte iterate prodrow ProductListRS.productList>
  <tr>
    <td>$$prodrow.Name</td>
    <td>$$prodrow.Price</td>
    <td><a
href="$$FORTE.ExecURL?ServiceName=ShopCartService&TemplateName=/Shopping
/dispcart.htm&price=$$prodrow.URLPrice&product=$$prodrow.URLName">
      </a></td>
  </tr>
<?/forte iterate prodrow></table>
<p>End of Product Listing. </p>

</body>
</html>

```

The template in [Figure 5-3](#) uses two iPlanet UDS tags:

- FORTE EXECUTE, with a tag named `CatalogHandler.GetProducts`
The values for the variables `$$prodrow.Name` and `$$prodrow.Price` are generated by the EXECUTE tag.
- FORTE ITERATE, with an iterator named `prodrow`

To respond to these tags, the iPlanet UDS programmer must write one or more Tag Handlers in which he codes the `HandleTag` method to respond to these tags and, if desired, generate the product data. A portion of a `HandleTag` method follows.

```

if Tag.Compare('PlaceDirectedAd',ignorecase=TRUE) = 0 then
. . .
elseif Tag.Compare('GetCategories',ignorecase = true) = 0 then
. . .
-- Handle 'GetProducts' to display available products
-- for the Requested category.
elseif Tag.Compare('GetProducts',ignorecase=true) = 0 then
. . .
    category : TextData = Request.FindNameValue('category');
    products : Array of Product =
        ShoppingService.GetProducts(category);

    for rowNum in 1 to products.Items do
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'Name',
            value = products[rowNum].Name);
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'ID',
            value = products[rowNum].ID);
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'Price',
            value = products[rowNum].Price);
. . .
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'URLPrice',
            value =
Response.Encode(products[rowNum].Price.TextValue));
        rset.Add(listname='productList',
            row = rowNum,
            attributeName = 'URLName',
            value = Response.Encode(products[rowNum].Name));
    end for;
end if;

```

Project: ShopCartClasses • **Class:** CatalogHandler • **Method:** HandleTag

The Web page that is generated by this HTML template and the HandleTag method appears in [Figure 5-2](#). This page is actually defined by three templates (we have followed the right-hand frame only).

Note that every time the page is generated, it will pick up the current list of products and prices (although in the ShopCart example, these do not change).

About the HTML Scanner Service

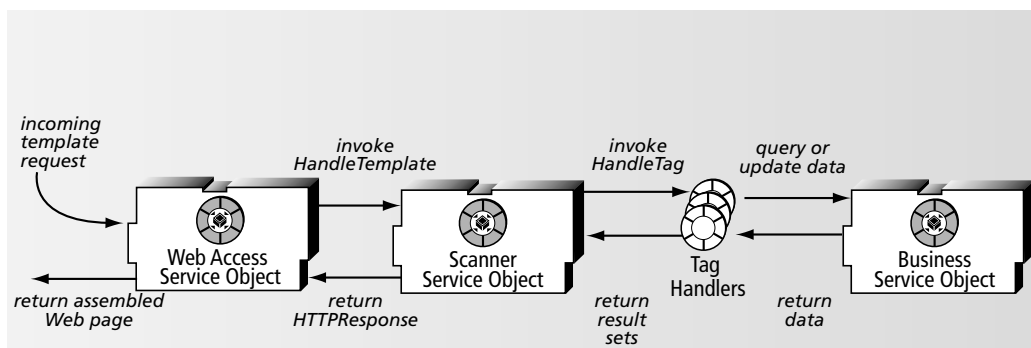
When the Web access service receives a request from a browser, it determines from the URL itself whether the request is for a page or a template. The Web access service directs all requests that use the "templateName" parameter to the scanner service object (*scanner* for short) and all requests that use the "pageName" parameter to the page builder service object. Thus, the scanner service and the page builder service are comparable services, both called by the Web access service.

When the request is for a template, the scanner uses templates to generate the appropriate Web page and returns the page to the Web access service object to return to the originating Web browser. (The page builder uses page builder methods to generate the appropriate Web pages for the Web access service to return to the browser.)

The purpose of the scanner is to coordinate the assembly of the actual Web page that is returned to the user. The scanner processes the named template, builds an HTML stream, translates the iPlanet UDS HTML tags into runtime data (called result sets) by executing each iPlanet UDS tag as it is encountered in the template, and adds the result set data to the template's HTML stream (as specified using iPlanet UDS variables) for runtime results.

[Figure 5-4](#) shows the role of the scanner service object in the Web application.

Figure 5-4 Role of the Scanner Service Object



When the scanner processes an iPlanet UDS HTML template, it encounters one or more FORTE tags, such as FORTE EXECUTE *getproducts* or FORTE IF *checksession*. The scanner processes each tag it encounters in the template, to assemble a Web page. The scanner then produces an HTTPResponse object that contains the complete Web page, and forwards the HTTPResponse object to the Web Access service object to return to the Web browser.

To optimize performance, the scanner maintains compiled versions of the templates in memory. It compiles the template the first time it is executed and recompiles the template only if the scanner detects a change in the template. You can revise a template and have the changes picked up immediately.

See [“Defining a Scanner Service Object” on page 69](#) for instructions for creating a scanner service object.

The iPlanet UDS HTML Tags

The iPlanet UDS HTML tags are a set of markup tags that can be embedded in iPlanet UDS HTML templates. The iPlanet UDS programmer provides code for each occurrence of an iPlanet UDS tag, to specify what processing and data are required.

The iPlanet UDS HTML tags are summarized in the table below. Note that in this manual the tags are referred to in abbreviated form (for example, the FORTE EXECUTE tag) while the actual syntax requires a preceding question mark (as in ?FORTE EXECUTE *tag_name*).

For a full description of iPlanet UDS HTML tag syntax, see [“Reference for iPlanet UDS HTML Tags” on page 127](#).

iPlanet UDS Tag	Tag Purpose
?FORTE EXECUTE	Executes the specified tag. You must write the code for each unique tag name in a <code>HandleTag</code> method.
?FORTE ITERATE ... ?/FORTE ITERATE	Performs a loop. Used to dynamically generate tables, lists, and so on.
?FORTE IF ... ?/FORTE IF	Executed in a TRUE condition. You must write the code for each unique condition tag name in a <code>HandleCondition</code> method. Condition tags may be nested.
?FORTE ELSE	Executed in a FALSE condition. This tag can only be used in an IF condition block.

iPlanet UDS Tag	Tag Purpose
?FORTE INCLUDE	Invokes a different template inline.
?FORTE REDIRECT	Redirects to a different template.

Tag Handlers

One responsibility of the iPlanet UDS programmer for an iPlanet UDS Web application is to define and maintain one or more *tag handler classes* (or *tag handlers*). These classes contain the methods that are invoked when a template encounters an iPlanet UDS tag.

Tag handler classes use two methods to respond to iPlanet UDS tags: the `HandleTag` method and the `HandleCondition` method. The method signatures for these methods are predefined in the `TagHandlerIface` interface in the HTTP Project. To create a tag handler, first you create a class to implement the `TagHandlerIface` interface and then you define these methods to add code for each unique tag name. For example, if a template contains the line `<?FORTE EXECUTE getLastNames>`, then the programmer must ensure that the `HandleTag` method responds when invoked with the tag name `getLastNames`.

You have a choice of which class(es) to use for tag handler(s). You may create one or more tag handlers, depending upon the requirements of your application. You can use any combination of the following:

- a subclass of the `HTMLScanner` class (that is, the base class of the scanner service object)
- a custom class

While both approaches are possible, using one or more custom classes is generally more practical for an application that has any number of developers, because it reduces contention for checking out methods for revision.

Purpose of the HandleTag and HandleCondition Methods

Regardless of the class you use as a tag handler, you must write code for each unique tag name used in the iPlanet UDS templates. Specifically, you must define the following methods if you use the following FORTE tags in a template:

iPlanet UDS Tag	Define This Method
FORTE EXECUTE <i>tag_name</i>	HandleTag
FORTE ITERATE <i>tag_name</i>	
FORTE IF <i>tag_name</i>	HandleCondition
FORTE ELSE	

Note that one `HandleTag` or `HandleCondition` method can contain code for a number of different tag names—you need not write a separate method for each unique tag name.

A `HandleTag` method can perform any or all of the following:

- process data input from the browser, map data to business objects, and invoke business services
- invoke business services and generate a *result set*
- update session specific information for the current browser session
- generate an HTML stream and append it to the result page currently being assembled

A `HandleCondition` method can perform any or all of the following:

- perform some sort of test and return TRUE or FALSE
- process input data (as above)
- generate an HTML stream and append it to the result page currently being assembled

Tag parameters The FORTE EXECUTE tag has optional parameters that allow you to pass additional information to the `HandleTag` method. You can specify one or more parameters, each with a name and a value. See [“FORTE EXECUTE Tag” on page 128](#) for more information.

Result Sets and iPlanet UDS Variables

You use the FORTE EXECUTE tag to generate data for display in a Web page, in the form of a result set. A *result set* is a temporary collection of data elements; each member of the result set is available for inclusion in a Web page. The value of using result sets (instead of static data) is that the result set always uses the most current data, keeping your Web page up to date. For example, a page showing training classes would automatically display new classes as they are scheduled, no matter whether the total list is 5 dates or 15.

To use a result set in a template, first you generate the result set in a `HandleTag` method. As you generate each member of the result set you assign the member a name. Then, you can refer to any member in a template, using an *iPlanet UDS variable*.

Format of an iPlanet UDS variable An iPlanet UDS variable requires two names: the name of the *result set* followed by a period and the name of the *result set member*, preceded by the string “\$\$”. The resulting string is called an iPlanet UDS variable:

`$$result-set-name.result-set-member-name`

The following examples of iPlanet UDS variables are both from the result set named `currCategory`; one uses the result set member ID and the other uses the member Surname:

`$$currCategory.ID`
`$$currCategory.Surname`

A result set can take any “shape” and return any data, from a single value to a complex set of values, such as a mix of scalars and arrays. See [“Constructing a Result Set” on page 118](#) for an example of naming and generating a result set.

The scope of a result set is the template in which it is created—the result set is destroyed after the template is done executing. While you can use a result set in multiple templates, in fact a new result set is created for each execution of the template. There is one exception to this: a template that is included in another template (using the FORTE INCLUDE tag) can refer to result sets that were generated in the calling template.

Example

In the following example of an HTML template, a result set named `currShopping` is produced by a FORTE EXECUTE tag named `GetCartContents`. The `currShopping` result set contains several members (`loginURL`, `numRows`, and `iterateList`). The member `IterateList` is actually a table whose rows are put in a display table by the FORTE ITERATE tag.

```

<p>Here's a list of all the products in your shopping cart.
<?forte execute GetCartContents resultset="currShopping"></p>

<form action="$$currShopping.loginURL" method="POST"
target="_top">
  <input type="hidden" name="numItemRows"
value="$$currShopping.numRows"><input type="hidden"
name="ServiceName" value="PaymentService"><input
type="hidden" name="TemplateName"
value="Payment/secinfo.htm"><p>&nbsp;</p>
  <table border="1" cellpadding="2">
    <tr>
      <td>Item Name</td>
      <td>Quantity</td>
      <td>Price</td>
    </tr>
    <?forte iterate curritem currShopping.iterateList><tr>
      <td>$$currItem.Name</td>
      <td><input type="text" size="20"
name="$$currItem.RowNum"
value="$$currItem.quantity"></td>
      <td>$$currItem.Price</td>
    </tr>
    <?/forte iterate curritem><tr>
      <td>&nbsp;</td>
      <td><table border="0">
        <tr>
          <td>Total</td>
        </tr>
      </table>
      </td>
      <td><table border="0">
        <tr>
          <td>$$currShopping.Total</td>
        </tr>
      </table>
      </td>
    </tr>
  </table>

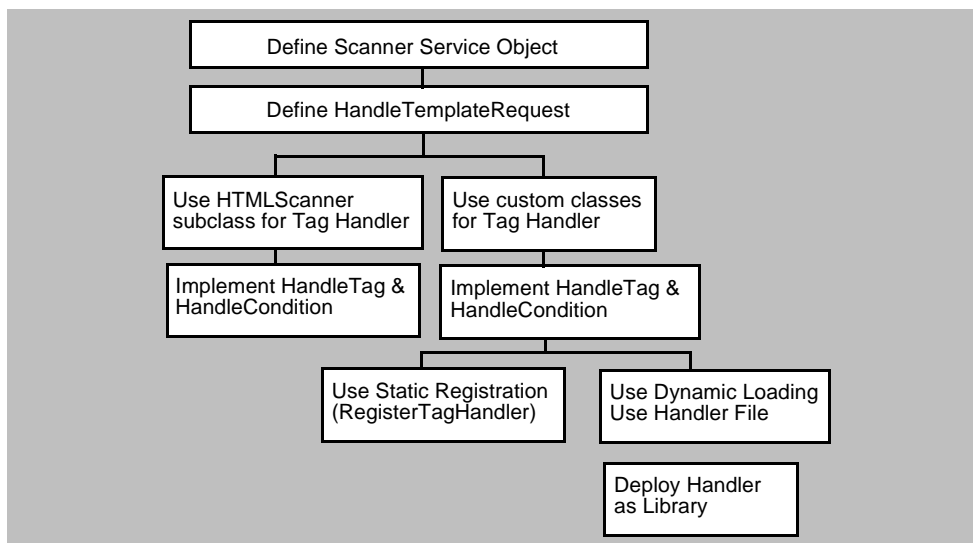
```

HTML File: shopping/dispcart.htm

Summary of Steps for Using Templates

The remainder of this chapter describes how to use templates and scanner service objects to return Web pages. These steps are summarized in [Figure 5-5](#).

Figure 5-5 Implementing Templates for Use with Scanner Service Objects



Designing a Template with iPlanet UDS Tags

This section describes how to use a third party HTML editor software package to define and layout Web pages. Each page corresponds to one HTML file. An HTML file that includes iPlanet UDS HTML tags is called an iPlanet UDS *template*.

► **To design a dynamic page using an iPlanet UDS HTML template**

1. Open an HTML or file editor and create a new Web page (HTML file).

You can use a commercial HTML editor (such as Sausage Software's HotDog, SoftQuad's HotMetal, and others) to edit and format a Web page that uses FORTE tags. You can also use a simple file editor, such as VI or emacs. If you use a commercial HTML editor, see ["Using HTML Editors"](#) below.

2. Add and format all the static text that you want on the page.
3. Add iPlanet UDS HTML tags as needed. (See ["Putting iPlanet UDS Tags in a Template"](#) on page 110.)

4. Add iPlanet UDS variables that will cause iPlanet UDS to substitute live data when the page is generated. (Before you can refer to an iPlanet UDS variable, you must use a FORTE EXECUTE tag that will generate the result set on which the variable is based.)
5. Add the graphics for the page.

You can include graphics of many types on a Web page that is generated by the page builder service or the scanner service. These graphics can be stored and maintained on either the Web server or the iPlanet UDS server. See *“Graphic, Image, and Binary Data Files” on page 211* for a discussion of how to reference the files in templates and pages, and the advantages of using different locations.
6. For future reference, make a note of which scanner object (if you have more than one) will be responsible for returning this template.

Using HTML Editors

Many HTML editors support two modes of editing and viewing an HTML file. You can usually switch quickly between these two modes in any HTML editor, although the menu or button name may vary by editor.

- In WYSIWYG mode (“what you see is what you get”) the page you are working on appears essentially the same as the page the end user will see. You see few, if any, HTML tags or markup.
- In HTML source mode the page you are working on appears very different from how it will appear when viewed by a user. You can see all the HTML tags (such as `<p>`, `<h1>`, and so on).

Figure 5-3 on page 100 shows the HTML source file for the template in *Figure 5-2*; you would edit this file if you were working in HTML source mode.

Using source mode to enter tags You should generally use source mode to enter or modify FORTE tags. (This is essentially the same as editing the HTML file using a file or text editor, which is also fine.)

Note on WYSIWYG mode If you prefer to use WYSIWYG mode to enter FORTE tags, you will encounter various techniques for handling the FORTE tags, depending on the editor. Describing the idiosyncrasies of these editors is beyond the scope of this manual. While you can use WYSIWYG mode in many cases, you should refer to iPlanet UDS Tech Note 11313 to obtain additional information regarding your HTML editor.

Putting iPlanet UDS Tags in a Template

Adding and modifying tags To add FORTE tags to a template, you edit the underlying HTML source (just as you would edit any text file). If you are using an HTML editor, you may need to switch between WYSIWYG and source mode. You can switch back to WYSIWYG mode to continue editing or to view the page in progress.

You can place the FORTE EXECUTE and REDIRECT tags in HTML comments. These two tags will not appear in the HTML template in WYSIWYG view, but will invoke the tag handler. However, if you place any other iPlanet UDS tag in an HTML comment it is treated as a comment, and will not invoke the tag handler.

You can add any tags to a template, as long as the tag handler for each tag is either registered by the scanner that will handle the template, or the tag handler can be dynamically loaded by the scanner. That is, the tags in a single template can be handled by multiple tag handlers.

Order and placement of tags No iPlanet UDS tags are required, nor is the order predetermined, with a few exceptions. The guidelines are described under each individual tag description; see [“Reference for iPlanet UDS HTML Tags” on page 127](#).

If you want to add any data or HTML text to the current Web page, you must use a FORTE EXECUTE tag to generate a result set.

The ITERATE tag must follow an EXECUTE tag, which generates a list result set. You can nest EXECUTE tags within an ITERATE loop; you might do this if you are generating a master-detail report. In this case, a FORTE EXECUTE tag would appear between the starting ITERATE and ending ITERATE tags.

You might place all the EXECUTE tags at the beginning of the template for easy reference, unless some EXECUTE tags are conditional, in which case you do not want to incur the overhead of processing them when they are not required.

One-part or two-part tag names The format of the tag name depends upon whether the tag handler class is a subclass of HTMLScanner or a custom class.

- If tag names are in two-part format, then a custom class is the tag handler. (See [“Using Custom Classes” on page 114](#) for more information.)
- If tag names are a simple name, then the tag handler is (must be) the subclass of HTMLScanner that is the base class for the scanner service object. (See [“Using Subclasses of HTMLScanner” on page 113](#) for more information.)

Note that you can always use a two-part tag name, as in *tag_handler_class.tag_name*.

ITERATE tag The iPlanet UDS ITERATE tag is the most complex tag to use. For specific instructions on using the iPlanet UDS ITERATE tag, refer to [“Using ITERATE to Add Tables and Lists” on page 120](#).

Tag reference Reference information for each iPlanet UDS tag appears in [“Reference for iPlanet UDS HTML Tags” on page 127](#).

Defining the HandleTemplateRequest Method

The purpose of the `HandleTemplateRequest` method is to pass a request for a Web page based on a template to the appropriate scanner service object.

In this method, you essentially “assign” the handling of a uniquely named template to a particular scanner service. When you make this “assignment,” you must assure that:

- the template HTML file is available to the delegated scanner in its document root directory (see [“The iPlanet UDS Document Root Directory” on page 210](#))
- any tag handlers invoked from the template are registered with the scanner service (see [“Register or Load Tag Handlers” on page 123](#))

It is possible to forward all template requests to the same scanner service. For example, if your application uses only one scanner service, then this method should simply pass all requests to that scanner. Or if each Web access service object only has one scanner service, then similarly, all requests for a given Web access service are forwarded to its scanner (this is the case for ShopCart, shown below).

In more complex applications, the `HandleTemplateRequest` method should reflect the design and number of scanner services. If one Web access service has multiple scanner services, then the `HandleTemplateRequest` method should only forward requests to a scanner service that expects that particular template.

► To define the `HandleTemplateRequest` method

1. Open the Class Workshop for your subclass of `HTTPAccess`.
2. Open the Method Workshop for the `HandleTemplateRequest` method.
3. Add code to define the `HandleTemplateRequest` method.

The code should invoke the `HandleTemplate` method, and delegate the request processing to the appropriate scanner service object. The scanner that receives the request (is called) must be able to process that template.

Your `HandleTemplateRequest` method may be very short (as shown in the following example), or it may contain a `case` statement if your application uses multiple scanners.

Example

The following example passes a template request (contained in the request parameter) to a specific scanner service for processing.

```
method CartAccess.HandleTemplateRequest
(input request: HTTP.HTTPRequest): HTTP.HTTPResponse
begin
return CartScannerSO.HandleTemplate(request);
end method;
```

Project: Access • **Class:** CartAccess • **Method:** HandleTemplateRequest

Creating TagHandlers

When you use templates to generate pages for a Web application, you must implement the interface called `TagHandlerIface` (predefined by the HTTP project). The `TagHandlerIface` interface allows you to define Web-enabling code and reuse the code for multiple pages. When you implement this interface, you provide the code for each unique tag name used in a template. The code for any specific tag name may call one or more iPlanet UDS business services and return data to be displayed on a Web page.

The `TagHandlerIface` interface defines signatures for two methods: `HandleTag` and `HandleCondition`.

Any class in which you implement the `TagHandlerIface` is called a *tag handler* class. You have a choice regarding which class(es) to use as tag handlers:

- You can implement the interface in one or more subclasses of `HTMLScanner`.
- You can create one or more custom classes to implement the interface.

Why you might choose a subclass or custom class is described in the following sections. The `ShopCart` example demonstrates both approaches; the `CatalogHandler` tag handler is a custom class, and the `CartScanner` tag handler is a subclass of `HTMLScanner`.

In the scanner, a single instance of the registered handler could be invoked from multiple threads simultaneously. One implication of this is that if any state information is stored in the handler, you should be aware that the state information may be accessed by multiple threads simultaneously. This is similar to having attributes of service objects when service objects may get called simultaneously by multiple clients.

Using Subclasses of HTMLScanner

You can use a subclass of `HTMLScanner` for a tag handler. You should use the same class that is the base class for the scanner service object that will encounter the tags in templates. Because the `HTMLScanner` class is predefined by iPlanet UDS to implement the `TagHandlerIface` interface, its subclasses also implement the interface.

You might use a subclass of `HTMLScanner` for a tag handler for the following reasons

- Your Web application is relatively small. In a large Web application, the `HandleTag` method would get too large.
- Your application team is small. With a large team, there would be contention among developers to check out the `HTMLScanner` subclass

If you implement the interface in a subclass of `HTMLScanner`, you do not need to use static registration or dynamic loading.

If you use a subclass of `HTMLScanner` for a tag handler, then the tag names that you use will have a simple format in HTML templates: *tag_name*. The tag name need not be preceded by the name of the tag handler class (although it can be); the tag handler class is assumed to be a subclass of `HTMLScanner`.

For an example, see the file `shopping/dispcart.htm` in the ShopCart example. This template refers to a tag named `GetCartContents` defined in the method `CartScanner.HandleTag`.

► To use a subclass of HTMLScanner for a tag handler class

1. Open the Project Workshop for the project that contains your scanner service (this is the same project as your Web access project).
2. Create a new class that has `HTMLScanner` as its superclass.
3. Drag or copy the methods `HandleTag` and `HandleCondition` from the `TagHandlerIface` interface (or from the `HTMLScanner` class) to the new class.

4. Provide code to define both methods. Make sure to provide code for each unique iPlanet UDS tag name and condition.

See the following sections for information about coding the `HandleTag` and `HandleCondition` methods.

Using Custom Classes

You can define one or more custom classes to implement the `HandleTagIface` interface. If you use a custom class, the class can be a subclass of `Object` or any other class. You might use a custom class for a tag handler for the following reasons:

- to reuse tag handlers among multiple templates without building a single monolithic method (this facilitates making the template and tag handlers reusable across Web applications)
- to divide the Web application development among a team of developers
- to allow modifications to a Web application without bringing it down (using dynamic library loading)

You can use more than one custom class as a tag handler class. For example, the ShopCart example uses two custom classes: `CatalogHandler` and `LoginHandler`.

If you use a custom class for a tag handler, then the tag names that you use will have a two-part format in HTML templates: *handler_class_tag_name*.

For an example, see the file `shopping/frcontent.htm` in the ShopCart example. This template refers to a tag named `CatalogHandler.GetCategories`, defined in the method `CatalogHandler.HandleTag`.

► To use a custom class for a tag handler class

1. Create a tag handler project specifically to contain your tag handler class(es).
2. Make the HTTP library a supplier plan to your new project.
3. Make any desired business services supplier plans to your new project.
4. In this project, create one or more custom classes to implement the `TagHandlerIface` interface.

These custom class can use any class as a superclass.

For each class, use the Interfaces tab page of the Class Properties dialog, and select the `TagHandlerIface` as the interface to implement.

5. Drag or copy the methods `HandleTag` and `HandleCondition` from the `TagHandlerInterface` interface to the new class(es).
6. Add code for both methods, making sure that you provide code for each unique iPlanet UDS tag name and condition.

See the next sections for more specific information and examples of defining the `HandleTag` and `HandleCondition` methods.

Now you must choose to use either static registration or dynamic loading for each tag handler class. For more information on making this decision, refer to [“Choosing Static Registration or Dynamic Loading” on page 124](#).

7. (To use static registration) Invoke the `RegisterTagHandler` method in an application initialization method.

See [“Using Static Registration” on page 124](#).

8. (To use dynamic library loading) Configure this project as a library.

Note that the supplier classes must also be configured as libraries. Often the easiest way to do this is to add all the supplier libraries to the same library distribution that contains this library.

You must deploy the library in the same environment where the Web application is deployed.

Refer to *A Guide to the iPlanet UDS Workshops* for information about creating a library, partitioning a library, and making the library distribution. Also refer to the *iPlanet UDS Programming Guide* for more information about using interfaces.

9. (For dynamic library loading only) Create a handler file for each scanner, or determine which handler file can be shared by the current scanner. Set the value of the `HandlerFile` attribute of the scanner service object to the name of the file.

See [“The Handler File” on page 209](#) for more information about this file.

In the ShopCart example, two classes in the ShopCartClasses project implement the `TagHandlerInterface` interface: `CatalogHandler` and `LoginHandler`.

Writing Tag Code

For each unique tag name associated with an iPlanet UDS tag that you embed in an iPlanet UDS HTML template, you must provide corresponding tag code, in `TOOL`, in a tag handler class. For example, if you use the tag `FORTE EXECUTE getdepts`, then your `HandleTag` method must include code for the name "getdepts."

You define tag code in either a `HandleTag` or a `HandleCondition` method. A typical `HandleTag` method might contain code for several different `FORTE EXECUTE` tags. You do not necessarily write one method for each unique tag name. For an example, see the `ShopCart` example method `CatalogHandler.HandleTag`.

Using input parameters from the EXECUTE tag The syntax of the `EXECUTE` tag allows for an arbitrary number of parameters (in the form of name-value pairs) that can be used to pass information to the tag handler. These parameters can take information from the Web page to be passed to the `HandleTag` method that handles that particular tag name.

Each parameter must be in the form of a name-value pair. The value must be enclosed in a double-quoted string.

Example

For example, assume that the HTML tag looks like the following:

```
<?FORTE Execute ParamTestFn(firstParam="fooString",  
    secondParam = "100")>
```

The corresponding code in the `HandleTag` method uses the `GetParameter` method on the `ParameterList` class to retrieve the individual parameters. That code might look like the following:

```

if Tag.Compare(source='ParamTestFn') = 0 then
-- Must cast the parameter to either IntegerData or TextData,
-- since GetParameter returns Object.

    -- Use the TextData
    myTextData : TextData =
      (TextData)Parameters.GetParameter(ParameterName='firstParam');
    -- Use myIntegerData
    myIntegerData : IntegerData =
      (IntegerData)Parameters.GetParameter(ParameterName='secondParam');
    ....
end if;

```

Defining output for a tag Often a tag is used, and defined, specifically to return data that is generated based upon criteria entered by the Web user. In this case, you write the tag code to return the data as a *result set*. A result set can return anything from one single value to a complex set of values that might be thought of as a mix of scalars and arrays. This process is described in the next section.

However, tag code can simply append pure text or a graphic (perhaps a logo) to an HTML page under construction. You can append “raw HTML” or use any of the HTML projects to directly append an HTML stream to the template.

Defining the HandleTag Method

You must add code to the `HandleTag` method to handle each unique tag name used for a FORTE EXECUTE or FORTE ITERATE tag.

► To define a HandleTag method in a tag handler class

1. In your `HandleTag` or `HandleCondition` method, test for the name of the tag, as in:

```

if Tag.Compare('PlaceDirectedAd',ignorecase=true) = 0 then
. . . -- processing for the PlaceDirectedAd tag
elseif Tag.Compare('GetCategories',ignorecase=true) = 0 then
. . . -- processing for the GetCategories tag
elseif Tag.Compare('GetProducts',ignorecase=true) = 0 then
. . . -- processing for the GetProducts tag

```

Project: ShopCartClasses • **Class:** CatalogHandler • **Method:** HandleTag

2. Add the appropriate processing code for each unique tag.

If you are coding a FORTE EXECUTE that returns a result set, then you must build a result set. (A FORTE EXECUTE tag need not return a result set.)

Constructing a Result Set

For most FORTE EXECUTE tags you will construct a result set that contains data that can be embedded in a Web page. (Note though, that a FORTE EXECUTE tag need not return a result set.)

Each result set is represented by an instance of the `ResultSet` class. For more information on this class, refer to iPlanet UDS.

► To build a result set in a `HandleTag` method

1. As needed, add “simple” members to the result set.

Use the variation of the `Add` method with the `rsMember` parameter, to specify each member’s name and value. (See the iPlanet UDS online Help.)

2. As needed, add “list” members to the result set.

Use the variation of the `Add` method with the `listName` parameter, to specify each member’s name and value. (See the iPlanet UDS online Help.)

You can add any number of either type of member in any order; a result set can take any “shape.”

The following guidelines may help you create result sets:

- You do not need to instantiate a result set object; simply use the `ResultSet` object passed as an input parameter to the `HandleTag` method.
- Although the iPlanet UDS tag in the HTML template specifies a result set name, your TOOL code *never specifies the actual result set name*. Rather, your TOOL code simply adds members to the result set by member name, as shown below, but the result set name itself does not appear anywhere in your TOOL code (except, perhaps, in comments).

Assume that the following iPlanet UDS tag is added to an HTML template file. It calls the tag named `CatalogHandler.GetProducts` with the result set named `ProductListRS`:

```
<?Forte Execute CatalogHandler.GetProducts Resultset="ProductListRS">
```

To respond to this tag, the `HandleTag` method defined for `CatalogHandler` constructs the `ProductListRS` result set, by adding the members “productList” (an array):

```

. . .
-- Handle 'GetProducts' to display available products
-- for the Requested category.
. . .
elseif Tag.Compare('GetProducts',ignorecase=true) = 0 then

    -- Retrieve the category id, which should have been
    -- a parameter riding in with incoming Request
    category : TextData = Request.FindNameValue('category');

    products : Array of Product =
        ShoppingService.GetProducts(category);

    for rowNum in 1 to products.Items do
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'Name',
            value = products[rowNum].Name);
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'ID',
            value = products[rowNum].ID);
        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'Price',
            value = products[rowNum].Price);

        -- Note the following usage of the Encode() method:
        -- any resultset members that
        -- are intended to be used as URL parameters must
        -- be encoded appropriately in case
        -- they contain spaces, control chars, etc.
        -- In the following 2 lines, Price contains
        -- a period (.) and the product name may contain
        -- spaces, so we encode them.

        rset.Add(listname = 'productList',
            row = rowNum,
            attributeName = 'URLPrice',
            value =
Response.Encode(products[rowNum].Price.TextValue));
        rset.Add(listname='productList',
            row = rowNum,
            attributeName = 'URLName',
            value = Response.Encode(products[rowNum].Name));
    end for;
end if;

```

Project: ShopCartClasses • **Class:** CatalogHandler • **Method:** HandleTag

This example uses the `Encode` method of the `HttpResponse` class to ensure that all members of the result set that might be used as URL parameters are encoded appropriately. For example, because the URL cannot accept spaces or control characters in the page parameters, we use `encode` on the result set member `Price` (which contains periods) and on `Name` (which may contain spaces).

Using ITERATE to Add Tables and Lists

You use the FORTE ITERATE tag in HTML templates to display “repeating” data generated dynamically. The FORTE ITERATE tag generates tables, menus, lists, drop-down lists, list boxes, and so on, for any HTML construct—plain HTML, or HTML lists, rows or tables.

The basic syntax for the ITERATE tag in an HTML template is:

```
<?FORTE ITERATE iterator_name result_set_list_member
```

The ITERATE tag uses an intermediate data storage construct, called an *iterator*, that is demonstrated below.

A running example of this tag appears earlier in this chapter; see [Figure 5-2 on page 99](#), [Figure 5-3 on page 100](#), and the sample `HandleTag` method following [Figure 5-3](#).

Instructional example The following example describes in detail how to use the ITERATE tag.

► To place a dynamically generated table in a Web page

1. Place a FORTE EXECUTE tag in your template.

The FORTE EXECUTE tag names the result set that will contain a list type member:

```
<?FORTE EXECUTE ATagHandler.AnyTagName RESULTSET="SetwithList">
```

You will use the tag name (`AnyTagName`) in the `HandleTag` method that generates the result set. (See [Step 3](#)).

You will use the result set name (`SetwithList`) again in the template in the FORTE ITERATE tag. (See [Step 2](#).)

2. Place a FORTE ITERATE tag in your template.
 - a. Lay out two rows of the table (a header row and the first data row). Set up the column headers, formatting, text flow, borders and shading.
 - b. Enter a FORTE ITERATE tag before the second table row, and the /FORTE ITERATE (end) tag after the end of the second row and before the end of the table.
 - c. Enter variables in the format *iterator_name.list_member_name* to place data from the iterator into the cells of the second row. The iterator then “clones” the second row for each row in the result set.

```

<?FORTE ITERATE MyIteratorName SetwithList.ListXYZName>
Print: $$MyIteratorName.ScalarElementName
</FORTE ITERATE MyIteratorName>
```

The iterator name is only used within the ITERATE tag, as the prefix for each element of the list.

3. Code the `HandleTag` method to generate a result set with a member that is a list:

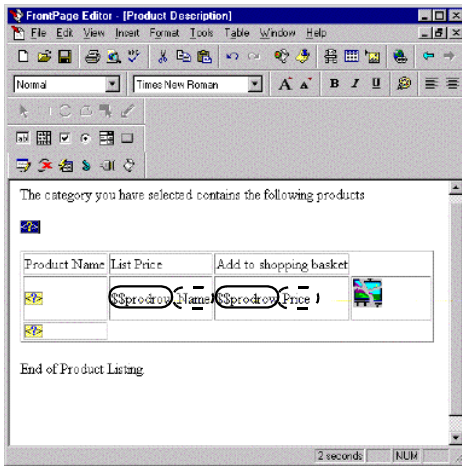
```

if Tag.Compare(source=AnyTagName) = 0 then
  someForteArray = someForteSO.GetArray();
  for rowNum in 1 to someForteArray.Items do
    rset.Add(listName='ListXYZName' ,
      -- actual name of list in ITERATE tag
      row=rowNum,
      -- row number of item in ITERATE loop
      attributeName='ScalarElementName',
      -- actual $$ member that ITERATE will print
      value=someForteArray[rowNum].someForteTextData);
      -- datavalue that corresponds to ScalarElementName
  end for;
```

You use the variation of the `Add` method for List result set members (see the iPlanet UDS online Help) to add each individual member to the result set.

Then, from this code, you use the values for two parameters (`listName` and `attributeName`) in the FORTE ITERATE tag in the template. (See [Step 2](#).)

HTML Template



HTML Source

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<html>

<head>
<meta http-equiv="Content-Type"
content="text/html; charset=iso-8859-1">
<meta name="GENERATOR" content="Microsoft FrontPage 2.0">
<title>Product Description</title>
<base target="main">
</head >

<body bgcolor="#FFFFFF">

<p>The category you have selected contains the following products</p>
<?Forte Execute CatalogHandler.SetProducts Resultset="ProductListRS">
<table border="1">
  <tr>
    <td>Product Name</td>
    <td>List Price< /td>
    <td>Add to shopping basket</td>
  </tr>
  <?forte iterate prodrow productListRS.productList>tr>
    <td>${prodrow.Name}</td>
    <td>${prodrow.Price}</td>
    <td><a
href="${FORTE.ExecURL?ServiceName=ShopCartService&TemplateName=/Shopping
/dispcart.htm&price=${prodrow.URLPrice}&product=${prodrow.URLName}"
  </a></td>
  </tr>
<?/forte iterate prodrow /table>
<p>End of Product Listing. </p>

</body>
</html>
```

HandleTag Method

```
elseif Tag.Compare( GetProducts, ignorecase=true) = 0 then
...
category : TextData = Request.FindNameValue('category');
products : Array of Product = ShoppingService.GetProducts(category);

for rowNum in 1 to products.Items do
rset.Add(listname = 'productList',
  row = rowNum,
  attributeName = 'Name',
  value = products[rowNum].Name);
rset.Add(listname = 'productList',
  row = rowNum,
  attributeName = 'ID',
  value = products[rowNum].ID);
rset.Add(listname = 'productList',
  row = rowNum,
  attributeName = 'Price',
  value = products[rowNum].Price);
...
rset.Add(listname = 'productList',
  row = rowNum,
  attributeName = 'URLPrice',
  value = Response.Encode(products[rowNum].Price.TextValue));
rset.Add(listname='productList',
  row = rowNum,
  attributeName = 'URLName',
  value = Response.Encode(products[rowNum].Name));
end for;
end if;
```

KEY

- = iterator name
- = tag name
- = member name

Defining the HandleCondition Method

For each uniquely named FORTE IF tag used in your HTML templates, you must handle the tag in a HandleCondition method.

The HandleCondition method returns a boolean result. This result determines whether the HTML block following the FORTE IF tag is used (if the result evaluates to TRUE) or if the HTML block following the optional FORTE ELSE tag is used (if the result evaluates to FALSE).

Example

The following example handles a condition named Validate. This particular condition returns TRUE if the login name and password associated with the current request are valid, otherwise this condition returns FALSE.

```

if CondName.Compare(source='Validate',ignorecase=TRUE) = 0 then
  loginName : TextData = Request.FindNameValue('login');
  passWord : TextData = Request.FindNameValue('password');

  if loginName <> NIL and passWord <> NIL then
    if loginName.ActualSize <> 0 then
      (ShopSession)(Request.CurrentSession).UserName = loginName;
      (ShopSession)(Request.CurrentSession).Password = passWord;
      result = TRUE;
    end if;
  else return FALSE;
  end if;
end if

```

Project: ShopCartClasses • **Class:** LoginHandler • **Method:** HandleCondition

Register or Load Tag Handlers

If you use custom classes for tag handlers, then you must either register those classes or load them dynamically. Stated a different way, if any of your iPlanet UDS tags is of the form FORTE EXECUTE *tag_handler_class.tag_name*, then you must register that class.

```
MyHTMLScannerSO.RegisterHandler (taghandlerclass)
```

You can register the tag handler statically or dynamically load it.

Choosing Static Registration or Dynamic Loading

Static registration is somewhat simpler to use, and is suitable if you know, before deploying your application, exactly which tag handler classes to register. However, if you use static registration, you cannot register a new tag handler class nor modify existing tag handlers while your Web application is running.

For critical applications requiring 24 x 7 up-time, tag handlers can be configured into a library. Then, if the scanner does not find a requested class to be registered, it attempts to dynamically load the class from a user-specified library. Dynamic loading does not require you to temporarily shut down a Web application in order to modify or add a new tag handler.

Using Static Registration

To register a tag handler statically, you invoke the `RegisterTagHandler` method of your `HTMLScanner` subclass in initialization code for your application. This method registers an instance of a tag handler class (that is, it registers an object of that class) with the scanner.

Example

Assume that the following tag is used in a template:

```
<?FORTE EXECUTE CatalogHandler.dosomething>
```

Then the following code would register an instance of a `CatalogHandler` tag handler:

```
-- first instantiate an object of the implementing class
shoppingCatInst : CatalogHandler = new;
-- then register the object with the scanner service object
CartScannerSO.RegisterTagHandler( shoppingCatInst );
```

Project: ShopCartApp • **Class:** AppControlWin • **Method:** InitializeWebServices

The scanner always searches for a tag handler in the static registered list first.

Using Dynamic Loading

If the scanner cannot find the class in the static registered list, it tries to load the implementing class dynamically from a user-specified library. In this case, the scanner requires a special text file (the *handler file*) to locate the library.

If you use dynamic loading, you must create a handler file (described in the next section) to contain class names and runtime class loading information. When the scanner sees a tag name or condition name of the form *class_name.tag_name*, and *class_name* is not already registered, it searches the handler file to load the class from a library. The class is then used to process tags and conditions.

For example, the following FORTE EXECUTE statement uses a two-part tag name, `orderservice.getorders`. If the `orderservice` class does not appear on the static registered list, the scanner service searches the handler file for an entry that begins with the class name `OrderService`.

```
<?FORTE EXECUTE orderservice.getorders>
```

If the `OrderService` class has not been statically registered, the scanner searches the handler file to locate the `OrderService` tag handler. The scanner then calls `HandleTag` on an `OrderService` object to process the tags and conditions.

The Handler File

The handler file is required only if you use dynamic loading for a tag handler.

The scanner service reads the handler file whenever a template request must be handled by a tag handler that is not statically registered.

The format of the handler file is one row for each tag handler class. Each row should contain the following data items, delimited by a comma:

- class name
- project name
- library name
- distribution ID
- compatibility level

Code Example 5-1 shows a sample handler file given the two libraries (LibA contains the tag handler classes TestClass1 and TestClass2, and LibB contains the tag handler classes BetaClass1 and BetaClass2):

Code Example 5-1 Sample Handler File

```
TestClass1, DynClasses, LibA, dynclass, 0
TestClass2, DynClasses, LibA, dynclass, 0
BetaClass1, DynClasses, LibB, dynclass, 0
BetaClass2, DynClasses, LibB, dynclass, 0
```

Identifying the handler file You specify the name and location of the handler file using two attributes of HTMLScanner:

- The DocumentRoot attribute specifies the base directory for the handler file (as well as template files). (See [“The iPlanet UDS Document Root Directory” on page 210.](#))
- The HandlerConfigFile attribute contains the path or file name or both for the handler file. Its value is appended to the value for DocumentRoot.

The value should use portable name format (see the Framework Library online Help).

You can override this value with an environment variable or a command line argument. The priority of these settings is as follows, listed from highest priority to lowest:

Command line argument Overrides all settings. Use the command line argument `-configfile` when starting the partition that contains the scanner service object.

Environment variable Overrides HTMLScanner’s HandlerConfigFile setting. Use the FORTE_WW_HANDLER_CONFIG_FILE environment variable.

Attribute setting Overridden by the other two techniques. Use HandlerConfigFile as described previously in this section.

Scanners can share files If a scanner service object requires dynamic access to *any* tag handler, then the scanner service object must have one handler file associated with it. A scanner that has no dynamic tag handlers does not need the file. Multiple scanner service objects can share the same handler file.

Testing a Template

Web page editors allow you to switch to a Web browser to view a page that you are currently working on. However, this viewing will not catch any errors in the iPlanet UDS tag syntax. Errors of that type cannot be detected until you can actually request the template from a running Web access service and scanner service that can handle that template.

Reference for iPlanet UDS HTML Tags

This reference section contains descriptions, syntax, and examples for the iPlanet UDS tags that you can use in iPlanet UDS HTML templates.

General syntax notes As you embed iPlanet UDS tags, keep the following general syntax rules in mind:

- iPlanet UDS HTML tags use a syntax similar to HTML syntax, with the addition of the prefix ?FORTE (or ?/FORTE to end a statement).
- Ending tags must start with ?/.
- Tag reserved words are case-insensitive (this manual shows them in upper case).
- Some tag options are case-sensitive on some platforms (for example: a file/path name).
- Spaces or white space within iPlanet UDS tags can take the form of: empty, space, tab, newline, or carriage return. All are equivalent and non-significant.
- Names for tags, result sets, and iterators and result set members can be of any length.
- In the EXECUTE and IF tags, enclose the name-value pairs in parentheses. Enclose the value in double-quotes.
- You can place either the FORTE EXECUTE or REDIRECT tag in an HTML comment—the tag will not appear in the HTML template in WYSIWYG view, but it will invoke the tag handler.

You should not place any of the other tags in an HTML comment, as they will not execute.

The tags are summarized in the following table and described below.

iPlanet UDS Markup Tags		
Tag Name	Tag Purpose	OK to embed in Comment?
?FORTE EXECUTE	Executes the specified tag. You must write the code for each unique tag name in a HandleTag method.	yes
?FORTE ITERATE ... ?/FORTE ITERATE	Performs a loop. Used to dynamically generate tables, lists, and so on.	no
?FORTE IF ... ?/FORTE IF	Executed in a TRUE condition. You must write the code for each unique condition tag name in a HandleCondition method. Condition tags may be nested.	no
?FORTE ELSE	Executed in a FALSE condition. This tag can only be used in an IF condition block.	no
?FORTE INCLUDE	Invokes a different template inline.	no
?FORTE REDIRECT	Redirects to a different template.	yes

Note on HTML editors Editing, modifying, and saving iPlanet UDS tags in a template can vary depending on the exact HTML editor that you use. Refer to iPlanet UDS Tech Note 11313 for detailed usage information for specific third party HTML editors.

FORTE EXECUTE Tag

The FORTE EXECUTE tag specifies the name of a tag to be executed.

Syntax

```
<?FORTE EXECUTE [tag_handler_class.]tag_name
  [(parameter_name = "parameter_value" [, . . .])]
  [RESULTSET = "rsname"]>
```

You can specify the parameter list and result set name in any order.

The parameter list is optional. You can specify an unlimited number of name-value pairs of parameters, each pair separated by a comma. Every value (including numeric constants) must be enclosed as a double-quoted string. See example c below. These parameters are passed to the `TagHandler` method.

The `RESULTSET` parameter is optional. If used, the corresponding tag code must construct a result set with one or more members. A member can be a list that will be processed by the `ITERATE` tag.

You can place an `EXECUTE` tag within an HTML markup comment, thereby making it invisible in the WYSIWYG view, but still invoking the corresponding tag handler. For example, the following usage is valid: `<!-- <?FORTE EXECUTE myFunc> -->`

Examples

Following are examples of several possible forms of the FORTE EXECUTE tag:

- a. `<?FORTE EXECUTE tag_name>`

```
<?FORTE EXECUTE AddLogo >
```

- b. `<?FORTE EXECUTE tag_name RESULTSET="resultset_name">`

```
<?FORTE EXECUTE SetupOrderSummary
RESULTSET="OrderSummary">
```

- c. `<?FORTE EXECUTE tag_name (parm-list) RESULTSET="resultset_name">`

```
<?FORTE EXECUTE class.tagName (param1="value",
param2="2352")
RESULTSET="rsname">
```

- d. `<?FORTE EXECUTE tag_handler_class.tag_name>`

```
<?FORTE EXECUTE WestReg.SetupOrderSummary>
```

FORTE IF ... ELSE Tags

The FORTE IF tag invokes the specified condition code that returns either TRUE or FALSE, and branches to subsequent code accordingly. The tag may have an optional named parameter list in parentheses. Conditions may be nested. Each IF tag must have a corresponding ending tag.

The FORTE ELSE tag is optional within a FORTE IF tag, to specify code to be invoked in the case that the IF tag resolves to FALSE. The ELSE tag can be used with the FORTE REDIRECT tag to send users to another page when a FALSE is returned by the FORTE IF condition.

Syntax

```
<?FORTE IF [tag_handler_class.]condition_name>
    [(parameter_name = "parameter_value" [, ...])]
...
[<?FORTE ELSE>]
...
</?FORTE IF>
```

The *tag_handler_class* is the name of a custom tag handler class.

Do not use a *condition_name* with the */?FORTE IF* tag. The FORTE ELSE tag has no corresponding */?FORTE ELSE* tag.

Example

The following example of the FORTE IF...ELSE tags validates user information. (For the full example, refer to the ShopCart.html file.)

```
. . .
<?FORTE IF LoginHandler.Validate>
. . .
<p>Final Order Summary</p>
<p>Thank you ...
<?FORTE EXECUTE LoginHandler.DumpLoginInfo(Name="yes") >
. . .
<p>You have ordered the following items:</p>
. . .
<form action="$$FORTE.ExecURL" method="POST">
  <input type="hidden" name="PageName" value="ProcessOrder"><input
    type="hidden" name="ServiceName" value="PaymentService"><p><input
    type="submit" name="Final" value="Please process my order!"></p>
</form>
. . .
<?FORTE ELSE>
<?FORTE REDIRECT "Payment/existacct.htm">
```

```
<?/FORTE IF>
```

HTML Template: summary.htm

FORTE ITERATE Tag

The FORTE ITERATE tag is used to generate a table dynamically.

This tag creates a named iterator to process an existing result set member that takes the form of a list. The ITERATE tag must follow an EXECUTE tag that has created the result set member, and the result set member must be a list containing one or more rows. Each ITERATE tag requires a corresponding ending ITERATE tag. You can create master-detail tables by nesting an EXECUTE tag within an ITERATE loop and setting parameters for the EXECUTE tag to the master value to iterate on.

Syntax

```
<?FORTE ITERATE iterator_name result_set_list_member
    [START = "numeric_constant"]
    [MAX = "numeric_constant" ]>
...
<?/FORTE ITERATE iterator-name>
```

The *iterator_name* is used when you refer to members of the iteration result set.

The *result_set_list_member* is a member of a result set that is an array or list, and was added to the result set using the variation of the Add method with the `listName` parameter.

The optional `START` and `MAX` parameters can appear in any order. The `MAX` parameter is useful if you are generating a table of an unknown size, and you would like to limit the number of rows that are actually displayed.

You must edit the HTML source directly to embed an ITERATE tag “between” table rows. Place the FORTE ITERATE tag as shown the code sample that follows, after the table’s header row (if there is one) and before the first row of table data. For a complete example see [“Using ITERATE to Add Tables and Lists” on page 120](#).

The effect of the ITERATE tag is to replicate the HTML text present between the `?FORTE ITERATE` and the `?/FORTE ITERATE` tags. Therefore, the starting and ending ITERATE tags cannot be embedded within HTML comments.

Example

The ShopCart example uses the following HTML template to display a table showing one or more items:

```

<?forte execute GetCartContents resultset="currShopping">
. . .
    <tr>
        <td>Item Name</td>
        <td>Quantity</td>
        <td>Price</td>
    </tr>
<?forte iterate curritem currShopping.iterateList><tr>
    <td>$$currItem.Name</td>
    <td><input type="text" size="20"
        name="$$currItem.RowNum"
value="$$currItem.quantity"></td>
    <td>$$currItem.Price</td>
</tr>
<?/forte iterate curritem>          <tr>
. . .
                                <td>Total</td>
. . .
                                <td>$$currShopping.Total</td>

```

HTML Template: Dispcart.htm

This template calls the following code in the `HandleTag` method; it creates the result set member, called `IterateList`, that generates the table of items:

```

elseif Tag.Compare(source='GetCartContents',ignorecase=true)
    = 0 then
. . .
if orders = NIL then
    orders = new;
else// else if previous orders exist, add them to rs for display
    for rownum in 1 to orders.Items do
        rset.Add('iterateList',rownum,'Name',orders[rownum].Product.Name);
        rset.Add('iterateList',rownum,'RowNum',IntegerData(value=rownum));
        rset.Add('iterateList',rownum,'Quantity',IntegerData
            (value=orders[rownum].Quantity));
        rset.Add('iterateList',rownum,'Price',orders[rownum].Product.Price);
        -- previously existing orders do not get highlighted in display
        -- rset.Add('iterateList',rownum,'HighlightRow',TextData(Value=' '));
        orderTotal = orderTotal + orders[rownum].Product.Price.Value;
    end for;
end if;

```

Project: ShopCartAccess • **Class:** CartScanner • **Method:** HandleTag

Nested ITERATE tags You can nest ITERATE tags, for example, to cause an outer iterator to loop through table rows and an inner iterator to loop through table columns. If you do so, be sure to end the nested block before you end the enclosing block (just as you would with any iterator, like nested `for` statements), as shown in the following code fragment:

```
<?forte iterate mainitem ...>
    ...
    <?forte iterate dependitem ...>
        ...
    <?/forte iterate dependitem>
    ...
<?/forte iterate mainitem>
```

FORTE INCLUDE Tag

The FORTE INCLUDE tag adds inline the specified template to the current template. Use the FORTE REDIRECT tag if you wish to switch to a different template altogether.

A template that is included in another template (using the FORTE INCLUDE tag) can refer to result sets that were generated in the calling template.

If a template recursively includes a large number of other templates (for example, template A includes templates B and C, each of which includes other templates, and so on), the Scanner Service Object partition could exhaust its default stack space. You can use the FORTE_STACK_SIZE environment variable to adjust the partition's stack size. See the *iPlanet UDS System Management Guide* for a description of how to adjust the stack size.

Syntax

```
<?FORTE INCLUDE "template_name">
```

You must enclose the `template_name` in double-quotes.

Examples

```
<?/FORTE INCLUDE "todaysadvert.htm">
```

FORTE REDIRECT Tag

The FORTE REDIRECT tag ends the use of the current template, and switches to the named template instead. Use the FORTE INCLUDE tag if you want to include inline a different template.

You may find it useful to use the FORTE REDIRECT tag after the FORTE ELSE tag, to anticipate certain circumstances when the FORTE ELSE condition evaluates to FALSE.

Syntax

```
<?FORTE REDIRECT "template_name">
```

You must enclose the `template_name` in double-quotes.

Examples

```
<?FORTE REDIRECT "Payment/existacct.htm">
```

Creating Pages Using Page Builder Methods

In an iPlanet UDS Web user interface, all Web pages that are returned to a Web client are generated by either the page builder service or the scanner service. This chapter describes how to use the page builder service and iPlanet UDS HTML classes to return Web pages to Web clients.

This chapter includes the following topics:

- the purpose of the page builder service
- writing the `handleRequest` method to return any requested page
- writing page builder methods to construct single Web pages
- using the `WindowConverter` class with existing iPlanet UDS windows
- designing a window for use as a Web page
- sharing window code with a Web page

For information about using the scanner service and templates to return Web pages, refer to [Chapter 5, “Creating Pages Using Templates.”](#)

Using a Page Builder Service

The page builder service and the scanner service are analogous; they both generate Web pages on request from the Web access server. Based on whether the URL contains the keyword “`pageName`” or “`templateName`,” the Web access service forwards each request for a page to either the page builder service or the scanner service. Each service in turn creates Web pages by obtaining the appropriate data from the iPlanet UDS business server and generating the Web page dynamically.

For example, to build the product list page, the SoftWear application's page builder service object gets the product list information from the iPlanet UDS business service object and then creates the HTML page.

Typically, the page builder service object defines one page builder method for each page. When the Web access service object receives a request for a page, it invokes the appropriate page builder method, passing the page parameters to the page builder service object. The page builder method constructs the Web page dynamically and returns the completed page to the Web access service object, which in turn forwards the page to the Web client.

Interacting with the iPlanet UDS business server To obtain the data needed for constructing the Web page, the page builder service object accesses the iPlanet UDS business service object. (This process is similar to using an iPlanet UDS server to provide data displayed by an iPlanet UDS client.) For example, to get the product list for the product page, the page builder service invokes a `GetProdList` method on the iPlanet UDS business service object, which returns the product list as an Array of Product Header.

Because the page builder service interacts with the iPlanet UDS business server to get data for constructing the Web page, your page builder service must translate the iPlanet UDS data into HTML. (You cannot display an iPlanet UDS object directly on a Web page.) Typically you will write a set of page builder methods in your page builder class to obtain data from the iPlanet UDS business service and translate that iPlanet UDS data into HTML.

Page builder service is optional This manual recommends the use of a Web access service object and either a scanner service object or a page builder service object (or both). While the Web access service object itself can generate Web pages, the Web access service object cannot be replicated and page construction can cause a performance bottleneck.

Page Builder Methods

Page builder methods are used to create dynamic Web pages—pages that contain data based upon the Web user's particular request. Such a page might retrieve and display the availability for a particular item, based upon a size and color entered by the Web user.

Before writing the page builder methods, you should decide which pages are required. Usually you define one Web page to correspond to each iPlanet UDS user window. While you can offer a different set of options (windows) to Web users than to iPlanet UDS clients, a good way to begin is to define a page builder method for each UserWindow class that is to be used as both an iPlanet UDS window and a Web page.

Typically, each page builder method is a wrapper for a corresponding method on the iPlanet UDS business server. The page builder method constructs a Web page by translating the HTTP request into iPlanet UDS data, invoking the corresponding method on the iPlanet UDS server, and translating the returned data into HTML.

In the SoftWare application, the `BuildProdList` method produces the product list page to be displayed to Web user. To get the data for the product list page, the `BuildProdList` method invokes the `GetProductList` method on the business service object. It then uses the `Add` method to add the returned data to the outgoing product list Web page. An excerpt from this method follows:

```
-- Next, build a list of products.  This uses the HTML list
-- features (<DL>, <DT> and <DD>).

dl : HTDl = new;
dt : HTDT;
dd : HTDd;

//GetProductList method invoked on business service
for p in CatalogService.GetProductList() do
  -- For each product, use its name as <DT>
  -- and its short description as <DD>.
  -- Also, put in the rest of the link needed
  -- to go to the ProdDetail page.
  dt = new;
  prodLink : TextData = new(value=linktd.value);
  prodLink.concat('&Code=').concat(p.Code);
  dt.Add(HTA(Text=p.Name, Href=prodLink.value));
  dl.Add(dt); //Data added to Web page

  dd = new;
  dd.Add(p.ShortDescription);
  dl.Add(dd);
end for;
```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder
 • **Method:** BuildProdList

Techniques for Writing Page Builder Methods

Given a URL that contains a page name and optionally some parameters, each page builder method constructs a Web page, which it returns to the `HandleRequest` method. A page builder method may perform a variety of tasks. For example, if you use the `WindowConverter` class to construct a page, you might not use the `Add` method to add individual HTML elements to a page. The following list simply gives you an idea of some of the tasks involved in constructing Web pages:

- examine the incoming Web request (an object of type `HttpRequest`) to determine which page is requested, and if specific data is requested for the page (for example, which catalog item to display details about)
- create the Web page to be built and returned (create an object of type `HTHtml`)
- add links to other pages by referencing the appropriate URLs
- invoke the appropriate methods on the business service object to get the requested data
- convert text, data values, and widgets to HTML elements, using any combination of HTML markup, or classes from the projects `HTML`, `HTMLWindow`, and `HTMLSQL`
- use the `Add` method to append each HTML element to the page in progress
- convert the finished page (an object of type `HTHtml`) to a string, using the `ConvertToString` method

If you are getting started, you can use the following code as a very simple example for creating a page, assuming that you are using the `HTML` project and are not converting a window.

```
html : HTHtml = new;  
head : HTHead = new;  
body : HTBody = new;  
html.Add(head);  
html.Add(body);  
head.Add(HTTitle(Text='HereisaTitle'));  
body.add('Hello, world');  
return html.ConvertToString();
```

The following pages describe various techniques for building Web page elements in your page builder methods.

Using HTML Tag Markup Directly

Web pages are defined using the HTML language, which uses HTML start and end tags to specify a number of HTML elements such as major and minor headers, various lists and fonts, and so on. Many tags also have options, such as to align right or left. You can use HTML markup directly, without using the HTML classes in the HTML project.

To use HTML directly in a page builder method, you simply concatenate text with the desired HTML tags and tag options.

Example

The following example creates the same simple page that is created in the preceding section. Since it uses no HTML classes, it does not require the HTML project to be a supplier.

```
td : TextData = new;  
td.Concat( '<HTML><HEAD><TITLE>HereisaTitle' );  
tc.Concat( '</TITLE></HEAD>\n' );  
td.Concat( '<BODY>Hello, world </BODY>\n' );  
td.Concat( '</HTML>\n' );  
return td.value;
```

Using HTML Classes

You can use the iPlanet UDS HTML classes instead of HTML tags to write Web pages. If you use the HTML classes, you gain additional syntax checking. See the iPlanet UDS online Help to compare the same Web page generated using both techniques.

You embed text in various formats in a Web page using the `Add` method of the `HTElement` class. For example, to add a straightforward paragraph, you would use the `HTP` element:

```
html : HTHtml = new;
head : HTHead = new;
body : HTBody = new;
para : HTP    = new;
html.add(head);
html.add(body);
head.add(HTTitle(Text='Day Dreaming'));
body.add(para);
para.add('Sometimes I sits and thinks,');
para.add('and sometimes I just sits.');
```

Using the WindowConverter Class

If you have user windows that are already defined in an iPlanet UDS application, then you can convert them to Web pages using the `WindowToForm` or `WindowToDocument` methods on the `WindowConverter` class.

WindowToForm Use the `WindowToForm` method if you have a complete user window that you want to convert, as is, to a Web form. The `SoftWear` application contains examples of this.

FieldToElement Use the `FieldToElement` method if you want to convert only part of a window.

The `WindowConverter` class creates only the layout or appearance of the equivalent Web form. You must also provide for the underlying logic that loads, validates, and manipulates data. See the iPlanet UDS online Help for instructions on working with data in a Web page.

NOTE Not all iPlanet UDS widgets are converted. For a list of how the widgets are mapped, see the iPlanet UDS online Help.

The `SoftWear` example uses `WindowConverter` to adapt several iPlanet UDS windows to Web pages. Below you see the window as defined in the `CatalogPageWindow` class in the `WWWSHaredWindows` project:

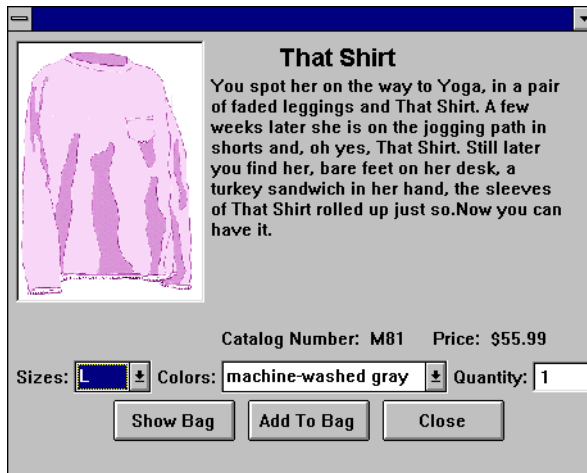
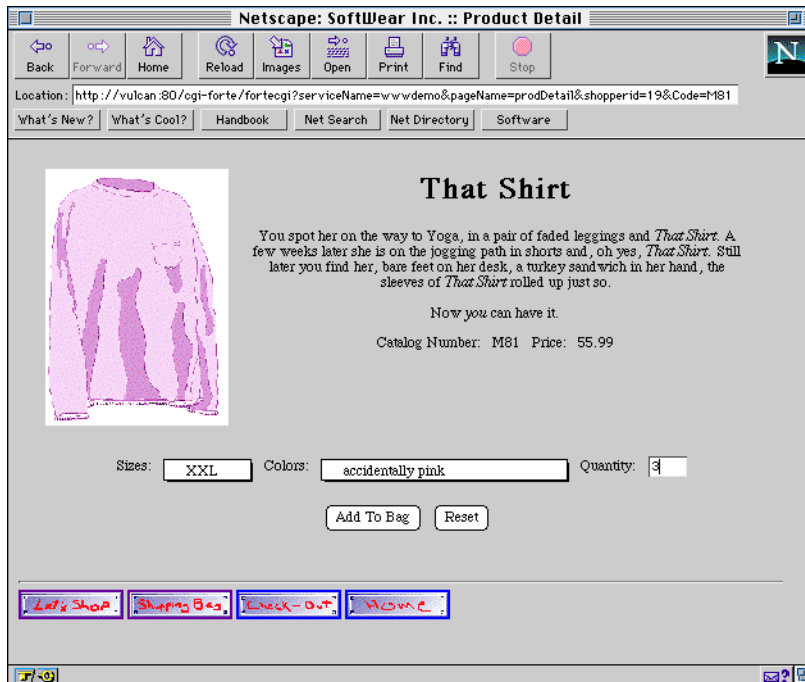
Figure 6-1 iPlanet UDS Client Window

Figure 6-2 shows the equivalent Web page, constructed using the `BuildProdPage` method on the `WWWCatalogPageBuilder` class. This method uses the `AssignFormat` method of the `WindowConverter` class to convert a `CatalogPageWindow` object:

```
w : CatalogPageWindow = new;
w.Setup(product = detail.Header, sourceURL = self.SourceURL);
. . .
Converter : WindowConverter = new(sourceWindow = w);
Converter.AssignFormat(w.<Name>, 'H1');
```

Project: `WWWCatalogPageBuilder` • **Class:** `CatalogPageBuilder`
 • **Method:** `BuildProdPage`

The layout and contents of the page, while not identical, are essentially equivalent.

Figure 6-2 Web Page Converted from iPlanet UDS Window

Using the SQLConverter Class

The `SQLConverter` class in the `HTMLSQL` project provides a quick and easy way for you to display tabular data retrieved from a database. Use the `SQLToTable` method on the `SQLConverter` class to perform a query on a database and return the results in an object of type `HTTable`.

See the iPlanet UDS online Help for a description of `SQLConverter` and some examples.

Saving Generated Pages to HTML Files

After the page builder method constructs a Web page, it converts the page's contents to a string to be passed back to the fortectgi program (or the iPlanet UDS plug-in). The string contains a series of HTML elements that form that page on the Web browser. You can view the HTML code that is generated, by writing a file using the `WriteToFile` method of the `HTElement` class. Sometimes these files are useful while debugging or sharing text (cutting and pasting) between pages.

Use the `WriteToFile` method after the page (or desired portion of the page) is generated. Simply specify a filename to which the generated HTML code is written. For example:

```
htmlPage : HTElement = new;
-- construct the page ...
htmlPage.WriteToFile('c:\\tmp\\pdetail.html');
```

Generated code can be too dense to read. If you set either of the attributes `HasEOL` or `HasReturnAfterStart` to `TRUE` for one or more of the HTML classes, the generated code is more readable (due to more line breaks).

Defining the HandleRequest Method

The Web access service automatically invokes the `HandleRequest` method whenever it receives a new request for a page (as opposed to a request for a template). You must define the `HandleRequest` method if you have any pages that are generated and returned by a page builder service. If all your pages are returned by a scanner service (that is, all your pages are generated by templates) then the `HandleRequest` method is never invoked.

The `HandleRequest` method invokes the appropriate method on the page builder service object to create the requested Web page. After the page builder service object returns the requested Web page, the `HandleRequest` method passes the page to the fortectgi program (or iPlanet UDS plug-in) to return to the Web user.

For a sample `HandleRequest` method, see [“A Sample HandleRequest Method” on page 145](#).

► **To define the HandleRequest method**

1. Drag the method signature for the `HandleRequest` method from the `HTTPAccess` class to your subclass of `HTTPAccess`.
2. Open the method in the Method Workshop.

CAUTION Do not change the parameters or return value of this method.

3. Check the input parameter (an object of type `HttpRequest`) for a valid page name.
4. Code a number of sections that resemble the following:

```

if pageName.IsEqual ('pageA', IgnoreCase=TRUE) then
    response.AssignResponse(Builder.BuildPageA(request));
else if pageName.IsEqual ('pageB' IgnoreCase=TRUE) then
    response.AssignResponse(Builder.BuildPageB(request));
...
end if;

```

For each valid page, invoke the corresponding page builder service method, passing through the input parameter (request, an object of `HttpRequest`). The page builder service object requires the information in the `HttpRequest` object in order to construct the appropriate page that contains the desired data. Then, for each page, the page builder service object returns a string (of HTML elements for that particular page).

5. Invoke the `AssignResponse` method on the string returned from the page builder service object. The `AssignResponse` method uses the string of HTML elements to set attributes in the `HttpResponse` object, in particular the `EntityBody` attribute.
6. Return the Web page as an `HttpResponse` object (already created in the standard `HandleRequest` method).

```
return response;
```

7. If desired, you can override the error handling contained in the predefined `HandleRequest` method. If you do not alter `HandleRequest`, then any unhandled exceptions raised in your code will result in a default Web error page, described [“The Default Web Error Page” on page 92](#).

A Sample HandleRequest Method

A sample `HandleRequest` method follows. You can use it as a sample when defining your `HandleRequest` method.

```
-- This method is automatically called when a Web request is for a
-- page rather than a template.

response : HTTPResponse = new;

-- Find the page name.
pageName : TextData = new;
pageName.SetValue(request.PageName);

-- Generate response pages.
if pageName.IsEqual('EnterNewPageNameHere', IgnoreCase=TRUE) then
    --
    -- Depending on the page name, create different response
    -- pages in string format. Then assign the page to response.
    -- Here you can invoke the methods you defined
    -- in the page builder service object.
    -- elseif statements may be needed for other pages.
    --
else
    -- Generate an exception for unknown page.
    ex : HTTPACCESSException = new;
    ex.SetWithParams (SP_ER_USER, 'Unknown pageName: %1.', pageName);
    ex.DetectingMethod = 'HTTPAccess::HandleRequest';
    task.ErrorMgr.AddError(ex);
    raise ex;

end if;

-- Return the response. It is sent back to the Web browser.
return response;
```

Adapting iPlanet UDS Windows with WindowConverter

If you have already defined a number of window classes for an iPlanet UDS application, you can adapt these windows for use as Web pages. To adapt (or “convert”) existing windows, you use the `WindowConverter` class of the `HTMLWindow` project. This class provides methods that allow you to:

- convert a window class to an HTML document (that is, a Web page)
- convert a window to an HTML form
- convert an individual field widget into an individual HTML element
- “transfer” data from a submitted Web form into the iPlanet UDS window for processing

Note that the `WindowConverter` class does *not* convert the programming associated with a window— it simply creates the layout of a Web page using the existing definition of an iPlanet UDS window class. For example, if the `Display` method associated with a window performs data validation for the window’s fields, data validation will not be automatically provided for the Web page. After converting the window, you must add any logic required for data validation and processing. See [“Sharing Window Code with the Web Page” on page 151](#) for information on how to add validation and processing logic to the page builder service.

The following steps describe how to convert an iPlanet UDS window into an HTML document (a Web page).

- **To convert an iPlanet UDS window (or widget) to an HTML document**
 1. Include the `HTMLWindow` project as a supplier plan to the project that contains the window.
 2. Create a `WindowConverter` object.
 3. Create your `UserWindow` object and load data into it.
 4. Set the `SourceWindow` attribute of the `WindowConverter` object to the window you want to convert.

5. Prepare the iPlanet UDS window for conversion (to allow for essential differences between iPlanet UDS windows and Web pages).

To prepare a window for conversion, you must do the following (in any order):

- a. Assign HTML formats to window text.

You must provide formatting tags for any text that appears on the window. Use the `AssignFormat` method to assign HTML formats to the text in the field widgets.

- b. Convert push buttons.

For all push buttons that will appear on the Web page, you must either:

- specify the button type using the `AssignButton` method (see the iPlanet UDS online Help), or
- assign an anchor to the button using the `AssignAnchor` method (see next step)

For each push button on the window that does not have an anchor and that you wish to include on the Web page, you must use the `AssignButton` method to specify whether it should be a Reset button, a Submit button, or ignored.

- c. Assign anchors to iPlanet UDS widgets.

For each iPlanet UDS widget that you want to act as a link to another Web page, you should assign an anchor to the widget. You can assign anchors to iPlanet UDS field widgets in two ways:

- You can use the `AssignAnchor` method; see the iPlanet UDS online Help.
- Alternatively, you can use the `HTML Options...` command in the Window Workshop; see [“Using the HTML Options... Command” on page 149](#).

Push buttons that have anchors assigned to them are converted into HTML text strings with borders. The text string with a border simulates the appearance of a push button, with the HTML text string providing the button’s label and the border providing the button’s shape. The anchor for the push button provides the processing that takes effect when the button is clicked.

d. Assign Image Files to Graphic Widgets

If the Web page will contain images, you should assign image files to each field that will display an image. Then you provide URLs to the image files and move all the images to a location on the Web server or iPlanet UDS server. You can assign image files to iPlanet UDS fields in two ways:

- Use the `AssignImage` method; see the iPlanet UDS online Help.
 - Use the `HTML Options...` command in the Window Workshop; see [“Using the HTML Options... Command” on page 149](#).
6. Use the `Assign` methods on `WindowConverter` (`AssignAnchor`, `AssignButton`, `AssignFormat`, and `AssignImage`) to prepare the iPlanet UDS window for conversion.
 7. Use the `WindowToDocument` method to convert the window to a Web page or use the `WindowToForm` method to convert the window into a Web form, which you can include on one or more Web pages.
 8. Add logic to your page builder service to perform data validation and processing for the Web page. See [“Sharing Window Code with the Web Page” on page 151](#) for more information.

CAUTION When you are manipulating a `UserWindow` object in your page builder code, you should *not* invoke the `Open` method on it. The `Open` method will actually try to display the window, but there is no display attached to the server partition. You can load data into the window without opening it.

The following sections provide more information about these steps.

Designing a Window for Use as a Web Page

When an iPlanet UDS window is converted into Web page, each field widget (with few exceptions, including tab folders) is converted to a corresponding HTML element. See the iPlanet UDS online Help for information about how iPlanet UDS field widgets are mapped to HTML elements; note that there are some widgets that are not converted.

Because the HTML elements are different sizes than the iPlanet UDS widgets, you should use the iPlanet UDS geometry management features to preserve the alignment of field widgets on the window when the window is converted. The *iPlanet UDS Programming Guide* describes the iPlanet UDS geometry management features used to create windows that are portable across window systems:

- grid fields
- field size partnerships
- field size policies

You should use these same techniques to ensure that your Web page is correctly formatted.

Using the HTML Options... Command

In the Window Workshop, choosing Widget > HTML Options enables you to set properties on the currently selected widget for use with the WindowConverter class. When you choose Widget > HTML Options, iPlanet UDS opens the HTML Options dialog, where you can set the Anchor URL property and, for picture fields, picture graphics, and picture buttons, the Image Source URL property.

► To set the HTML options for a widget

1. Select the widget.
2. Choose Widget > HTML Options.
3. In the HTML Options dialog, enter the Anchor URL and/or Image Source URL.

Anchor URL property The Anchor URL property allows you to assign a URL to be used as an anchor for the widget. When the window is converted into a Web page, the corresponding HTML element will be an anchor.

HTMLink attribute The Anchor URL property corresponds to the HTMLink attribute on the `FieldWidget` class. The HTMLink attribute allows you to assign a hypertext link to current field widget programmatically. For complete information on the HTMLink attribute, see the Display Library online Help.

Image Source URL property The Image Source URL property, available for picture fields, picture graphics, and picture buttons only, specifies the location of the image to be displayed on the Web page. The image file that you assign to a field can be located on the Web server or the iPlanet UDS server. The format of the image file can be any format supported by the Web server. The Image Source URL must provide the address of the actual image file relative to the Web server.

If you do not assign an image file to provide the image for the picture on your window, or if the URL is incorrect, the Web browser displays a broken picture image in place of the field.

HTMLImage attribute The Image Source URL property corresponds to the HTMLImage attribute on the `PictureButton`, `PictureField`, and `PictureGraphic` classes. The HTMLImage attribute allows you to specify the location of the image for the picture button, picture field, or picture graphic programmatically. For complete information on the HTMLImage attribute, see the Display Library online Help.

Converting a Window

When the `WindowToDocument` method converts a window into an HTML page, there is only one Web form on the page. Panels on the window are converted into tables, not individual forms. If you want to include more than one form on a Web page, you must construct your own page—use the `WindowToForm` method to convert an iPlanet UDS window into a form for inclusion on the page.

Converting outline fields When an outline field is converted into an HTML unordered list, iPlanet UDS displays only the nodes in the outline field that are currently open. Any nodes that are initially closed in the outline field are ignored in the corresponding HTML list and the Web user has no way to access them. Therefore, before converting a window that contains an outline field, you should be sure to open all nodes that you wish to display to the Web user.

Note that the size of the HTML list is determined by the content of the outline field, not by its `VisibleLines` attribute. Because all visible nodes are included on the Web page, the list on the Web page can be significantly larger than the outline field on the iPlanet UDS window, which displays only the “visible lines” and allows the user to scroll through the total number of nodes.

Sharing Window Code with the Web Page

Ideally, your Web page and iPlanet UDS window will share code for the following types of functions: window initialization, data validation, and data processing.

The preferred way to share code is to use specialized, independent methods to perform data validation and processing, rather than use the `Display` method for these purposes. This may require that you modify the original `Display` method for a given window class, to break out all initialization, data validation, and operation code for that window. Then, the `Display` method for the window and the page builder service can invoke these methods as necessary.

After you create independent initialization, data validation, and processing methods, you can streamline the original `Display` method. Your new `Display` method can start by invoking the initialization methods, and then provide an event loop that invokes data validation methods in response to events on the window.

Your page builder service can invoke the same initialization methods on the window before converting it into a Web page. After the end user enters data onto the Web page, the page builder service can invoke the `LoadParameters` method to load the Web page parameters into the original window. When the data is loaded into the window, the page builder service can invoke the original data validation and processing methods directly on the window itself.

LoadParameters method The `LoadParameters` method on the `WindowConverter` class enables you to use the same data validation and processing methods for a Web page as for the corresponding iPlanet UDS window. After you load the parameters into the window, you can perform data validation and processing on the corresponding fields in the window exactly as if the end user had entered the values directly into the fields. See the iPlanet UDS online Help for information.

Initializing a window Before converting a window to a Web page, remember to perform the initialization you normally perform on the window. The following code fragment from the `Confirm` method in the `SoftWear` application illustrates using a `Setup` method to initialize the confirmation window before converting it:

```
-- Now construct the next page to display. This will use the
-- ConfirmationWindow from the Forte client application to
convert.
w : ConfirmationWindow = new;
w.Setup(tmporder, tmpbasket);

-- Now reset converter's sourceWindow to w, a instance of
-- ConfirmationWindow.
```

```
converter.sourceWindow = w;
```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder • **Method:** Confirm

The following two code fragments illustrate how the SoftWear application uses the same processing method within the `Display` method for an iPlanet UDS window and within a page building method for the page builder service.

The first code fragment illustrates the use of the `FillOrderFromWindow` method:

```
when self.<orderButton>.Click do
  status_line = 'Submitting order...';
  tmpOrder.ShopperID = info.ShopperID;

  -- use FillOrderFromWindow to process data
  innerOrderWindow.FillOrderFromWindow(tmpOrder);
  self.Window.UpdateDisplay();
```

Project: WWWSharedWindows • **Class:** OrderWindow • **Method:** Display

The second code fragment illustrates use of the `FillOrderFromWindow` method to process the data on the window after the `LoadParameters` method has transferred the data from the Web page into the window:

```
innerOrderWindow : NestedOrderWindow = new;
converter : WindowConverter = new;
converter.sourceWindow = innerOrderWindow;
//LoadParameters method invoked to transfer data
converter.LoadParameters(request);

-- The FillOrderFromWindow will take the data that has been loaded
-- into the window from the Web request, and move it to an Order
-- object (in the tmpOrder parameter).
innerOrderWindow.FillOrderFromWindow(tmpOrder);
tmpOrder.ShopperID = shopperID;
//FillOrderFromWindow method invoked to process data
```

Project: WWWCatalogPageBuilder • **Class:** CatalogPageBuilder • **Method:** Confirm

Using Session Management

WebEnterprise session management features allow you to build session and state management into a Web application easily. These features automatically generate, validate, track, and delete sessions in a Web application.

This chapter describes why you would want to use session management, and describes the following features:

- unique session IDs that are automatically generated
- session properties that enforce, for a page or entire application, whether or not a session is required
- session objects, based on the `HTTPSession` class, that can hold any type of information for a single session
- a session table automatically maintained by iPlanet UDS to contain information for each active session

The Benefits of Session Management

Some Web applications have no need to track the actions of their Web users; users can visit any Web page with no restriction or control, and the Web application maintains no user-specific information (“state”). Such applications often want to attract as many Web visitors as possible, and consequently they show no sensitive data. A typical example of such an application is an on-line catalog that Web users can browse through freely. In this case, however, when a customer begins to select items to order, then session management becomes important; the Web application must start to maintain user-specific information in expectation of a purchase.

On the other hand, many Web applications do access sensitive data or do require tracking the actions of an individual end user. For example, these applications must be able to control access to individual Web pages in order to verify user identities, control data updates, control possible navigation paths, and so on. An example of this type of application is a Web health insurance profile program, in which each individual can only see his or her own personal health information. WebEnterprise offers session management features for this purpose.

In iPlanet UDS WebEnterprise, the *session management* features allow you to manage state, identity, and security information. Session management allows you to build continuity into an application as users navigate, however randomly, through a Web site. Session management also gives users the feeling that the application is tailored to them individually.

Following are a few examples of the uses of session management:

- automatic user validation
- responding appropriately (returning different pages or data) based on a user's recent actions
- tracking a user's progress through an application
- automatic timing out of sessions that have expired

WebEnterprise allows you to add session and state management features to the essentially stateless HTTP protocol. Furthermore, WebEnterprise stores and maintains all session management information on the server, rather than the client, eliminating the need to update and pass state information with each request or response, as you could do with the iPlanet UDS Web SDK.

By default, session management features are turned off. (Web applications based on the iPlanet UDS Web SDK require no adjustment to allow for session management features.)

Web session manager You enable session management by invoking the method `EnableSessionManagement` in your application's initialization code. Enabling session management automatically creates a `SessionMgr` object (*session manager*) specifically to manage sessions. Session management also creates a session table, which the session manager uses to create sessions and track session data for each unique browser session.

For information on enabling session management, see “[Enabling Session Management](#)” on page 173. For a description of the session manager and the session table, see “[Validating and Tracking a Session](#)” on page 160.

NOTE Session management features are available to all pages constructed and returned by an iPlanet UDS Web application. Specifically, whether a scanner service object or a page builder service object is responsible for coordinating a template or page request, both types of service object can use session management.

The Meaning of Session and State Management

The session management features of WebEnterprise encompass state management and more. The meaning of these terms can differ depending upon context or by product. In WebEnterprise, these terms are not exactly interchangeable, but do overlap. They have the following meanings:

State management This term refers to managing information (called *state information*) for a single Web user. In WebEnterprise, state information is maintained for the duration of a single browser session by default (for example, items ordered in a single shopping session). If you implement persistent storage for state information, then the information can span multiple browser sessions (for example, total dollars spent by a customer in all shopping sessions to date).

The term state information is more narrowly defined than “session management.” The information that is maintained as state information is always application-dependent. The actual data values that are stored are unique to each user. Typical state information often includes a unique ID to identify the user (or order, or record, for example). State information may also be referred to as *session data*, because it is the data relevant to a unique session.

Session management This term is used more generally to encompass aspects of managing multiple Web users or multiple Web pages for a Web application. It entails managing access control properties that determine whether valid sessions are required for users to access pages, controlling the route a user takes to navigate through an application, and determining when and how sessions are created. For example, what information must a user provide to start a session, along with what information is maintained for each user (the latter being the state information). It also entails terminating, timing out, identifying, and invalidating sessions.

The WebEnterprise “session management features” include both types of management.

In standard iPlanet UDS window-based applications, you can store state information on both the client and the server. To store state information on the client, you can use local objects. To store state information on the server, you can use transaction or session dialog duration service objects.

However, in a Web user interface you must handle state information differently because you cannot use local objects, nor can you use service objects that have transaction or session dialog duration. The preferred way to use state information is to use the session management features of WebEnterprise, as described in this chapter. However, you can also use cookies to store state information (see [“Using Cookies” on page 186](#)).

Applications that are based on the iPlanet UDS Web SDK releases 1.0 or 1.1 cannot use the WebEnterprise features, but can handle state information as described in the following two sections.

Session Management Features

When “session management” is enabled for an iPlanet UDS Web application, a session, identified by a unique session ID, is automatically created for each distinct Web browser session. A session can be automatically created with no validation, or created based on custom validation criteria (typically, a user ID). Within your application, you indicate which pages require a valid session in order to access them, by specifying a *session property* for each page. You also define what state information should be tracked during a session for individual users of your application. Then WebEnterprise uses the session ID, which can be accessed with the `CurrentSession` attribute, for the duration of the session to update or access the session’s information.

The following sections describe in more detail these features that comprise WebEnterprise session management. See also [“Validating and Tracking a Session” on page 160](#) and [Figure 7-1 on page 162](#) for a description of how these components work together.

If your application does not require session management, no action is required. Session management is disabled by default.

Session Properties for Web Pages

WebEnterprise allows you to require a valid session for each Web user that interacts with an application. If your Web application requires you to monitor or control access to specific Web pages or data, you can assign a session property to each of the sensitive pages. *Session properties* indicate whether a Web user must have an active, validated session to view a page or whether no session is required. You can assign session properties to a single template or page, or to a group of templates or pages that reside in the same directory.

The session properties are shown in the following table:

Property	Key Word	Description
session required	SESSION_REQUIRED	A page request must be associated with a current, valid session.
sessions created automatically	SESSION_AUTOCREATE	If no valid session is found for a page request, WebEnterprise automatically creates a new session.
session unspecified	SESSION_UNSPECIFIED	The default value: No session is required.

Setting session properties You can set session properties at a number of levels. You can set a default session property that will apply to all pages and templates, and you can override the default for single pages or for all pages maintained in a particular directory. Refer to [“Setting Session Properties for Pages” on page 176](#) for details about the various ways to set session properties.

Web Session Manager

When you enable WebEnterprise session management by invoking the method `EnableSessionManagement`, the Web access service object automatically creates an instance of the `SessionMgr` class. This object, called the *Web session manager*, performs session management tasks. The session manager associated with a given Web access service is available using the `WebSessionMgr` attribute on the `HTTPAccess` subclass.

The session manager is not a service object and its use and function are essentially transparent to both application developers and users. The session manager's tasks all relate to setting and using information in the session table, and include the following tasks:

- creating and updating the session table itself
- intercept, decrypt, and validate session IDs that are contained in incoming URLs (page requests)
- timeout expired sessions

These tasks are described in [“Validating and Tracking a Session” on page 160](#) and illustrated in [Figure 7-1 on page 162](#).

Much of the work of the session manager is performed within the `HTTPAccess` class, as a request is received and validated. Like the Web access service object, the session manager cannot be replicated.

Session Objects and the Session Table

WebEnterprise uses sessions to track interactions with an iPlanet UDS Web application. A *session* is defined as all interactions occurring between a single Web browser session and a (iPlanet UDS) Web application. After you enable session management and define what information should be tracked as state information, WebEnterprise automatically maintains state information for every session. The following sections all assume that you have invoked `EnableSessionManagement`.

A session starts with a browser's initial request to a Web access service object that has enabled session management. By default, sessions are created by the automatic invocation of the `CreateSession` method; you can override this method if you need to create sessions manually. When the session starts, iPlanet UDS generates a unique session ID (described below). A session ends either by timing out or by being explicitly deleted, using the `DeleteSession` method on the `SessionMgr` class.

For more information about using timeouts or deleting sessions, see [“Deleting or Timing Out Sessions” on page 174](#). For more information about how sessions are validated and timestamps are updated, see [“Validating and Tracking a Session” on page 160](#).

Session objects For each unique session the session manager automatically creates a *session object*, based on the `HTTPSession` class. The session object contains state information, if any, for a single user; it represents the client side application context. The session manager uses this object to track a user's navigation through a single Web application and to update *state information*—data that is specific for that user in that session. A session object always exists for a current session if session management is enabled.

Session table The session manager creates and uses a *session table* for session management. The session table contains a list of active session IDs, each with its corresponding session object. The session manager uses the session table to validate and time out sessions, and to store and retrieve session data in session objects.

While WebEnterprise maintains the session table automatically, you control what actual session data is stored for a session object. You use the session object to add and retrieve session data (state information) for a current session. To store session data, you use the `SetSessionData` method, and to retrieve session data you use the `GetSessionData` method (from the `HTTPSession` class). To refer to the current session, simply use the `CurrentSession` attribute of the `HttpRequest` class, as in the following line of code:

```
request.CurrentSession.SetSessionData('surname', Object);
```

See [“Working with State Information” on page 179](#) for specific information about using these methods.

By default, the session table is stored in server memory on the partition that hosts the `HTTPAccess` service. If the partition in which the session table is created shuts down unexpectedly, the session table is lost and its data cannot be reconstructed. See [“Using Persistent Storage for State Information” on page 181](#) if your application requires the server side session table to be persistent.

Sharing the session table Multiple Web access service objects within a single application can share and update a single session table. See [“Multiple Web Access Services Sharing Sessions” on page 173](#) for more information.

Session IDs

WebEnterprise creates and manages unique session IDs. Although you reference session IDs to get and set session data for individual sessions, you never use the actual numerical session IDs directly. Instead, you use the `CurrentSession` attribute of the `HTTPMessage` class (inherited by both `HTTPResponse` and `HttpRequest`).

WebEnterprise uses one of two approaches to store and pass session IDs:

- By default, WebEnterprise uses cookies to pass session IDs for a given session, setting the ID in the cookie with each response, and retrieving it with each request, for the duration of a session.
- WebEnterprise also embeds session IDs in URLs under certain circumstances. If WebEnterprise detects that a given browser has disabled cookies, it embeds an encoded form of the ID in each URL that is generated during that session. To see an example of a URL that includes a session ID, see [“Session IDs in URLs” on page 184](#).

Session ID generation When a session is created, the Web access service object assigns it a unique session ID. You can write a custom scheme for generating session IDs by overriding the `GenerateSessionID` method on the `SessionMgr` class.

By default, Session IDs are not persistent. A given ID lasts only for the duration of a browser session, and ends when the session is explicitly deleted or times out. Nor is the data stored for a given session (identified by a session ID) persistent. If your application requires persistent IDs, you must customize your application by adding persistent storage such as a database or file to manage IDs and any data that must be maintained between sessions. See [“Making Session IDs Persistent” on page 175](#).

Encryption of IDs Session IDs are stored in unencrypted form in the session table, and unencrypted IDs are used to validate current sessions or to time out sessions. WebEnterprise encrypts session IDs, using the encrypt key, for transmission across a network; this is the only time the encrypted version of a session ID is used. You can override the default encrypt key using the `SetEncryptKey` method on `HTTPAccess` class (See the iPlanet UDS online Help).

Validating and Tracking a Session

When session management is enabled, the session manager uses the session table to validate a session for each incoming request. Then it uses the session object corresponding to each session to maintain state information for each session.

► **The following steps describe how iPlanet UDS validates and tracks a session**

(Figure 7-1 illustrates and is keyed to the following steps.)

1. On receiving an incoming request from the Web access service object, the session manager checks the HTTP request header information for an encrypted session ID. If it finds one, the session manager decrypts it to get the “raw” session ID.

The session ID may have been generated either by the `CreateSession` method in application code or by a page with the session property of `SESSION_AUTOCREATE`.

2. The session manager looks for a session object with that session ID in the session table, and checks whether that session has timed out.

The session manager compares the time since the last session validation to the session’s timeout interval. Each session object can have a different timeout interval.

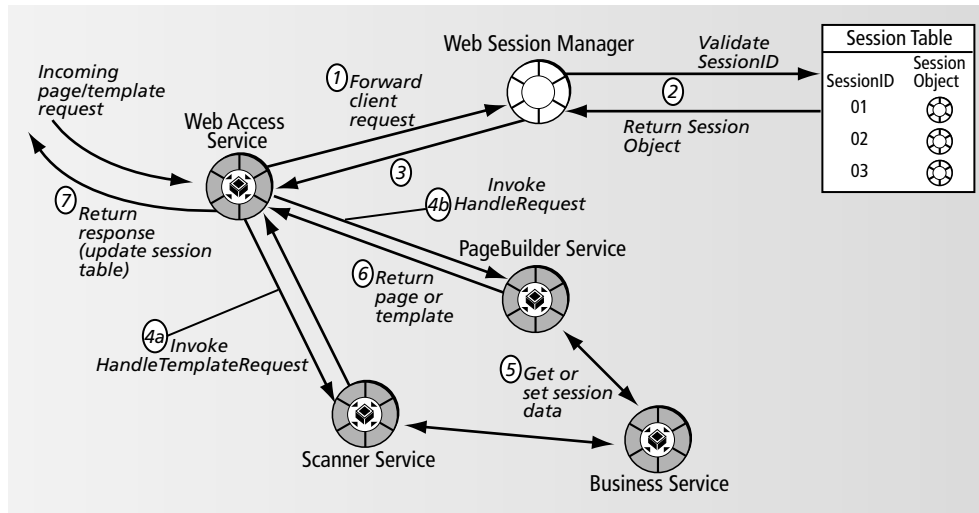
If the session is valid, the session manager updates the validation timestamp to the current time. (The session manager updates the timestamp—and session object—again, when an `HTTPResponse` is returned, so the validation timestamp is updated on both the request and the response for a given session.)

If a session has expired, or a request comes in with no session ID, what happens next depends on the session properties associated with the requested template or page, as follows:

- If the property is `Session_Autocreate`, `WebEnterprise` creates a new session for the request.
- If the property is `Session_Required`, `WebEnterprise` redirects the response to the page designated by the `SessionCreationURL` attribute for the Web access service. This URL leads to a page in which the user can enter the information required to begin a new session, if desired. If the `SessionCreationURL` attribute is unspecified, an exception is returned to the user. (For more information see “[Setting the SessionCreationURL](#)” on page 172 and “[Session Creation Page](#)” on page 85.)
- If the property is `Session_Unspecified`, no special action occurs, because no session is required. A `NIL` current session is attached to the request.

You can add criteria to the default validation by overriding the `ValidateSession` method.

From this point, each request is associated with a session object in the session table (or with a `NIL`).

Figure 7-1 Validating and Tracking a Session

3. The retrieved session object is then available to the application code as the `Request.CurrentSession` attribute.

You can use the `CurrentSession` attribute to set and get session data (using the `SetSessionData` and `GetSessionData` methods) because this session object is tracked and available to you between consecutive requests in a session.

4. The current request is available to the application as the request parameter (of type `HttpRequest`) for the `HandleTemplateRequest` (4a) and `HandleRequest` (4b) as well as the `HandleTemplate` and `HandleTag` methods.
5. Before a response is returned to the Web browser, the session object for that session ID is updated in the session table, if necessary.
6. A response is returned to the Web access service object.
7. The Web access service passes the requested page to the browser.

Typical Session Management Scenarios

In a typical secure application, a user must *log in* (provide some information to be validated) before gaining access to any protected application window.

In a traditional windows-based application the login process is relatively easy to implement because users cannot arbitrarily access any window within an application. A user must have acceptable access rights to access a protected window and must follow a prescribed navigation path to the window. A login window controls access to protected windows by using a security mechanism that grants (or denies) users access.

In contrast, in the Web environment, a user can easily request an arbitrary page from a Web site—for example, by simply entering the complete URL from the Web browser command line or by using a bookmark. In so doing, a user might circumvent a typical windows-based security scheme, by avoiding completely the login window.

WebEnterprise addresses this security issue with the `SESSION_REQUIRED` session property. To see how this property works, let us first consider how to implement page level user-validation without using the `SESSION_REQUIRED` property.

Adding user validation without session management By default in WebEnterprise, no session management is enforced—that is, no page requires a session in order to be viewed. This is equivalent to the property `SESSION_UNSPECIFIED`. To implement page-level security for any given page, the following code, or its equivalent, must be added for *each* page requiring protection:

```
<?FORTE If DisallowPageAccess>
<?FORTE REDIRECT "login.htm">
-- assumes login.htm allows a user to create a valid session for
-- future access to this page
<?/FORTE If>
```

Or, for a page requiring user validation, the following code, or its equivalent, must be added in the `HandleRequest` method to check page access:

```
HandleRequest()
if PageName.Compare('somesecurepage') = 0 then
if mySecurityMechanism.DisallowPageAccess(request.CurrentSession) then
-- return a message with a link to a page the user can access
-- with his current access privilege
end if;
... normal processing for this page..
else if....
```

So, without the benefit provided by the `SESSION_REQUIRED` property, session management must be individually implemented on every page, requiring a significant coding and maintenance effort.

Adding user validation with session management Using WebEnterprise session management, this same scenario can be streamlined as follows.

➤ **To define default page-level session properties, and then override session properties as needed**

1. Require that all pages and templates accessed through the `myAccess` service are associated with a valid session. Use one of the following approaches:

Specify a default session property for all templates and pages returned by `myAccess` service:

```
myAccess.SetDefaultSessionProperty(SESSION_REQUIRED);
```

Or, set the default session property for all `.htm` files in and below a particular directory (where `/'` represents the root of the path) or for page builder files that use the directory/file name naming convention:

```
myAccess.SetSessionProperty('/',SESSION_REQUIRED);
```

Now you have effectively specified that, by default, a valid session is required to view any page returned for this application or any page builder page whose name starts with `/'`.

2. Define a session creation page that users can access to enter the information required to create a valid session (for example, a username and password). Set its session property to *not* require a valid session.

For example, define a template named `login.html` in the `Login` directory under the `documentRoot` specified on the `Scanner` that is invoked by the `myAccess` service. The session property for this template, however, cannot be `SESSION_REQUIRED`, but must be set explicitly to override the default property, as follows:

```
myAccess.SetSessionProperty('Login/login.html',SESSION_UNSPECIFIED);
```

3. Define a validation page that validates information entered in the login page. Like the login page, this form must be accessible without a valid session.

If the login information is valid, this page invokes the `CreateSession` method to create a session for the client.

```
myAccess.SetSessionProperty('ValidateLogin',SESSION_UNSPECIFIED);
```

4. Override the session-required property for any directory containing templates, or individual templates, that *do not* require a valid session.

For example:

```
myAccess.SetSessionProperty('BankInfo',SESSION_UNSPECIFIED);
```

5. In those directories, override properties for any template or page that *does* require a valid session.

```
myAccess.SetSessionProperty('BankInfo/quotes.htm',SESSION_REQUIRED);
```

Using the `SessionPropertyConfig` file Alternately, you could specify all the above session properties in the `Session Property` configuration file.

► To specify session properties using a session property file

1. Assign the session properties file name.

For example, invoke the following code in an initialization method:

```
myAccess.SetSessionPropertyConfigFile('sessprop.cnf');
```

This file should be placed in the directory as indicated by the environment variable `FORTE_WW_DOCUMENT_ROOT`. If this variable is not defined, you must specify the absolute path to the file.

2. In the session properties file, add rows to specify session properties for a path or file.

To assign the same properties as in the previous programmatic steps, the file would look like the following:

```
/,SESSION_REQUIRED  
Login/login.html,SESSION_UNSPECIFIED  
ValidateLogin,SESSION_UNSPECIFIED  
BankInfo,SESSION_UNSPECIFIED  
BankInfo/stockquotes.htm,SESSION_REQUIRED
```

Now, assume a user requests the following URL:

```
http://www.forte.com/cgi-forte/fortecgi?ServiceName=MyAccess  
&TemplateName=Bankinfo/stockquotes.htm
```

The service MyAccess knows that requests for this template must be accompanied by a valid session. To determine whether the incoming request has a valid session, the service MyAccess invokes the `ValidateSession` method on the `HTTPAccess` class.

The `ValidateSession` method simply compares the time since the last timestamp for the session to the session's timeout interval (`HTTPSession.SessionTimeout`), and returns `TRUE` if the session has not timed out.

About SESSION_REQUIRED

The following scenarios describe two ways you might use `SESSION_REQUIRED`.

All Pages are Available to All Users

In the first scenario, assume the following application characteristics:

- All pages are accessible to a user request associated with a valid session.
- A valid session is simply any session that has not expired. No additional information or qualification is required.

In this scenario, you simply create a session on each user's first request, as long as the session criteria (for example username and password) entered by the user are valid. Then, any pages or templates requiring a valid session are accessible assuming that a valid session accompanies the request for the page or template. All pages and users are treated "equally."

(This is demonstrated in the previous example—the login and validation pages are accessible without a session, but all other pages require a valid session.)

Different Pages are Available to Different Users

A more complex scenario might require more conditional access management—for example, when a valid session does not necessarily confer the right to see *all* pages in the application. Access to a given page depends on access rights of the requesting client. That is, all users and pages are not equal. The application defines what constitutes a valid session based upon some criteria, such as:

- username as entered in a login page, or perhaps as looked up in an access control database
- the client host (associated with the CGIEnv object in HTTPAccess)
- any other criteria that may be saved in the session object

In traditional window applications, access to each window is controlled by the parent window, so typically the security check code goes in the parent window. In contrast, in a Web application any page can be requested from any other page, so the security check code must be controlled from one central place. In WebEnterprise, the `validateSession` method fills that purpose. The `validateSession` method is invoked for each request for a `SESSION_REQUIRED` page or template returned by the Web access service.

This scenario requires overriding the `ValidateSession` method to include additional session validation code such as the following:

```

-- The default ValidateSession checks for expiration of timestamp.
-- Must invoke it first.
if super.ValidateSession(request) = FALSE then
  return FALSE;
end if;
begin
-- The access service decides whether the requested page should
-- be allowed to be returned to the client. If different pages are
-- to be handled differently, this is where the code goes.
if request.TemplateName = 'securePage.html' then
mySession = request.CurrentSession;

-- userName was previously added to the session data in
-- the validateLogin page

loginID : TextData = (TextData)(mySession.GetSessionData('userName'));
-- the application's security mechanism will decide whether the
-- current user has access rights for viewing the requested page.

if mySecurityMechanism.AllowPageAccess
  ('SecurePage.html',loginID) = TRUE then
  return TRUE;

-- If ValidateSession returns TRUE, there is
-- a valid session that can access the requested page/template.

else
return FALSE;
-- If ValidateSession returns FALSE, the page will not be returned
-- to client.
end if;

-- For a SESSION_REQUIRED page, if ValidateSession returns FALSE,
-- the Web access service redirects the user to the URL indicated
-- by SessionCreationURL.

end if;
end;
return TRUE;

```


About SESSION_AUTOCREATE

The SESSION_AUTOCREATE property is useful when an application has many pages, all of which may be accessed in any order by the client, and all of which potentially require client session tracking.

A good example is a retail catalog example like ShopCart, where the catalog has numerous pages or items. The application should track a session, no matter from what page a user enters the application. For example, even if a user uses a bookmark to jump to a product he had previously bookmarked, the application should allow this type of user navigation.

Case 1: Using SESSION_UNSPECIFIED This type of application could use the default SESSION_UNSPECIFIED property. In this case, for each new *template* (each item page), you would have to include a FORTE tag at the top of the template like the following:

```
<?FORTE Execute CreateSessionIfNoneExists>
```

Then you would have to code the tag CreateSessionIfNoneExists to create a valid session used to access this site.

And, if any *page* in the catalog is returned by a page builder service, you would have to add code to check for an existing session in the HandleRequest method.

```
HandleRequest()
if PageName.Compare('SomeSessionRequiringPage') = 0 then
if request.CurrentSession = NIL then
self.WebSessionMgr.CreateSession(request);
end if;
.. normal processing for this page..
else if....
```

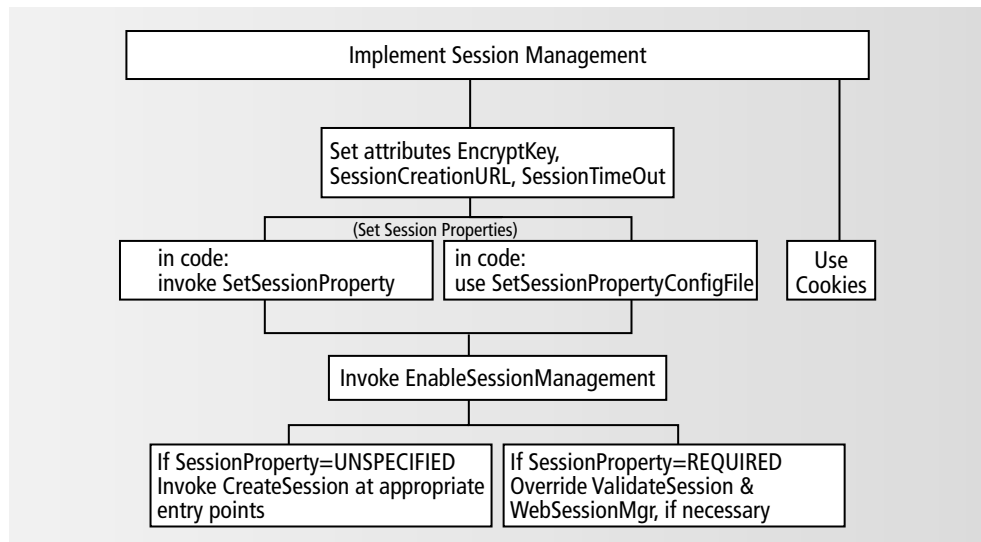
Case 2: Using SESSION_AUTOCREATE An alternative to the above case is to use the SESSION_AUTOCREATE property. When this property is specified for a page or template, the Web access service automatically creates a session for any request for the template or page. This session property is a reasonable alternative for all of the catalog templates or pages in a catalog application, avoiding the need to explicitly write the code to create a session and embed it in each template or page.

Implementing Session Management

Using WebEnterprise session management is optional. However, for all but the most simple, internal applications, you will probably want to use standard WebEnterprise session management.

To use session management, you must perform a number of steps, summarized here, shown in [Figure 7-2](#), and explained in depth in the remainder of this chapter.

Figure 7-2 Steps for Implementing Session Management



► To use WebEnterprise session management

1. Set (customize) desired session management attributes. See [“Initializing Session Management Attributes”](#) below.
2. Enable session management. See [“Enabling Session Management”](#) on page 173 below.
3. Set session properties. See [“Setting Session Properties for Pages”](#) on page 176.
4. Define and use state information. See [“Working with State Information”](#) on page 179.
5. Modify URLs. See [“Modifying URL Links for Session Management”](#) on page 182.

Alternatives to session management If you do not enable session management, then any user may access any page of your application and you cannot set a timeout interval after which Web clients will be automatically disconnected from your application. However, with some coding you can maintain state information for Web application users, by using one of the following techniques:

- embedding state information in URLs or using data in hidden form fields
- using cookies (See [“Using Cookies” on page 186](#))

Initializing Session Management Attributes

If you use session management, you can customize the following settings:

- encrypt key (use `SetEncryptKey` method)
- timeout interval (use `SetSessionTimeout` method)
- session creation page (set `SessionCreationURL` attribute)

If you use an administration window, you can change these whenever you start the application. If you do not need to change them often, you can use the initialization code. While technically these settings are optional, you will usually set them.

Setting the Encrypt Key

The *encrypt key* is used to encrypt session IDs. Although there is an arbitrary default encrypt key, you probably will want to set your own key. To do so, invoke the `SetEncryptKey` method on your `HTTPAccess` service object. See [“Session IDs” on page 159](#) for more information about how the encrypt key is used, and the iPlanet UDS online Help.

Setting the Session Timeout Interval

The timeout interval is the time interval after which an inactive session expires. The default timeout interval is 30 minutes.

To set the timeout interval, invoke the `SetSessionTimeout` method on your Web access service. For example, to set the session timeout interval to 15 minutes, you could include the following code in the `Init` method for the service object’s class:

```
-- 'yy:mm:dd:hh:mm:ss:ms'
shopSessionTimeout : IntervalData = new(
    value = '00:00:00:00:15');
CartService.SetSessionTimeout(shopSessionTimeout);
```

When you set a timeout interval for your application, you may want to let your users know what the interval is, so that they are not surprised if their sessions time out.

Setting the SessionCreationURL

The SessionCreationURL attribute points to a login page, where users have the option of entering the information required to start a new session. For more information, see [“Session Creation Page” on page 85](#).

While you are not required to set a value for this attribute, if you do not do so and users attempt to access a page or template requiring a session, they will get an exception. If you set this attribute, then users are automatically redirected to the page that you specify, where they have the option of entering the information required to start a new session.

Set this value to a page that is designed to gather the information required to start and validate a new session for a client. This page must have the session property SESSION_UNSPECIFIED, so that users can enter the page without having an existing session.

The ShopCart example sets this attribute as follows:

```
SessionCreationURL = new;  
SessionCreationURL =  
    task.Part.OperatingSystem.GetEnv( 'WWW_CGI_URL_BASE2' );  
SessionCreationURL.Concat(  
    '?ServiceName=ShopCartService&TemplateName=/Shopping/shopmain.htm' );
```

Project: ShopCartAccess • **Class:** PaymentAccess • **Method:** Init

For more information, see the iPlanet UDS online Help.

Enabling Session Management

By default, all session management features are disabled.

► To enable session management

1. In a method in the initialization code for your application, invoke the `EnableAccess` method on the Web access service object.

For information on initializing an application, see [“Initialization Tasks” on page 212](#).

2. Invoke the `EnableSessionManagement` method on the same service.

This creates a Web session manager and session table for session management.

These two methods must be invoked in this order. For more information on either method, see the method descriptions for the class in the iPlanet UDS online Help.

Multiple Web Access Services Sharing Sessions

Multiple Web access service objects within a single application can share a single session table and session manager. This configuration allows applications with multiple Web access services to scale to accommodate many users. Using this design, one session manager performs session management tasks for multiple Web access service objects, using one session table. After the session manager has created a session, all services can see and update the session object.

► To allow multiple Web access services to share sessions

1. Invoke the `EnableAccess` method as described in the previous section.
2. Invoke the `EnableSessionManagement` method as described in the previous section.

This creates a Web session manager and session table for session management.

3. Each subsequent Web access service wishing to share the session table should invoke the `ShareSessions` method (instead of the `EnableSessionManagement` method), specifying the name of the Web access service from [Step 1](#). Then each service can share and update the session data in the session table.

Optional Customizations

The following sections describe some ways that you can customize session management features, depending upon the requirements of your application.

Deleting or Timing Out Sessions

Because end users communicate with an iPlanet UDS Web application only by initiating requests for Web pages, a user can stop using your Web site without any explicit notification to your application. You may decide that letting sessions time out is sufficient for your application, or you may take a more active role in deleting sessions at appropriate points in your application.

By default, a session will time out after 30 minutes, based on the attribute `SessionTimeoutInterval`. This means that a session object in the session table will be deleted when the time since its last timestamp exceeds 30 minutes. You can change this timeout interval as described above. You can also force a validation using the method `ValidateSession`.

If you want to explicitly terminate a session, you use the `DeleteSession` method from the `SessionMgr` class. Deleting a session may be appropriate in some applications to effect a “transaction”; if the end user continues on in the application, another new session can simply be created. The `ShopCart` example explicitly deletes sessions when the user decides to make a purchase, and starts a new session if the user then resumes shopping.

```
-- Handle ProcessOrder:
-- Once the user confirms the order, we must delete the
-- current session so user can start another shopping basket.
elseif pageName.Compare(source='ProcessOrder',ignorecase=TRUE) = 0 then

    --Before returning to non secure browser mode you
    --must destroy the current session.
    self.WebSessionMgr.DeleteSession(request);

    response.CookieParameters = NIL;
    redirectLocation : TextData = new;
    redirectLocation =
        task.Part.OperatingSystem.GetEnv(
            'WWW_CGI_URL_BASE');
    redirectLocation.Concat(
        '?ServiceName=ShopCartService&TemplateName=shopping/shopmain.htm');
    response.Location = redirectLocation.Value;
    response.AssignResponse(
        'Forte Shopping Cart... Taking you back to Shopping');
```

Project: ShopCartAccess • **Class:** PaymentAccess • **Method:** HandleRequest

Making Session IDs Persistent

As described under “[Session IDs](#)” on page 159, session IDs are not persistent unless you customize your application. By default, WebEnterprise maintains state information for active sessions in a session table that is stored as a cache table, in memory on the server. This means that if the server goes down unexpectedly, the session table is lost and cannot be recreated. All session IDs are lost, and any active Web users in mid-session must get a new session in order to continue, and their previous session data is lost.

You can make session IDs persistent by adding persistent storage (typically a database, or a file) to store IDs and any session data that should be maintained between sessions.

Specifying a Non-Default Session Manager

If you choose to customize any session management features, you may need to create your own session manager object.

► To create your own session manager

1. In your application initialization code, invoke the `EnableSessionManagement` method to start session management.
2. Invoke the `SetWebSessionMgr` method in the `HTTPAccess` class.
3. Invoke the `EnableAccess` method.

These methods must be invoked in this order.

Mixing Secure Sockets Layer (SSL) and non-SSL

If your application requires the use of Secure Sockets Layer (SSL) for security, you can use multiple Web access service objects in the same application: one service object using HTTP and the secure service object using HTTPS. Depending upon your requirements you can do this in either of the following ways:

- You can use multiple Web servers with the same Web service.

If session management is enabled and using cookies, you must initialize the default cookie’s `DomainName`. Both Web servers must have the same domain name.

If session management without cookie support is allowed, then you should *not* use the variable `$$$FORTE.ExecURL` to create links to a different Web server (because the default `ExecURL` is a property on the Web access service object that represents the default Web server). Instead you should use the `BuildURL` method to construct the URL containing the embedded session ID.

- If many pages will be accessed over the SSL connection, you can create a second distinct Web access service object for the HTTPS server.

In this case, you must create two HTTPAccess subclasses, one for each port. The two Web access service objects should share sessions. The functionality of the state management is encapsulated in a `SessionMgr` class that is the `WebSessionMgr` attribute on the `HTTPAccess` subclass. By default, the Web access service creates its own instance of the `SessionMgr` class. To enable two Web access services to share sessions, you must assign the `SessionMgr` instance for one Web access service object to the other, using the `ShareSessions` method. If the two Web access service objects run in different partitions, then the `SessionMgr` class must be a distributed reference.

Setting Session Properties for Pages

As described in [“Session Properties for Web Pages” on page 157](#), session properties determine whether a valid session is required in order to return a page to a Web browser. You can set session properties in the following ways:

- Use a session property file to set session properties for directories of templates, or single templates. (See [“The Session Property File” on page 177](#).)
- Invoke the `SetDefaultSessionProperty` method on your Web access service, for all pages and templates returned by that service.
- Set the session property programmatically on individual pages and templates, using the `SetSessionProperty` method on the `HTTPAccess` class.

Precedence Since session properties can be specified in more than one way, an order of precedence is defined. The order is the same as the list above, the first item having the lowest precedence. The session property set for an individual page or template always takes precedence over a session property set for a directory or group of pages.

Setting a Default Session Property

You can assign a default session property for all pages returned by a single Web access service object. A session property applies to all Web pages returned by the given Web access service, unless you override the session property for one or more pages, as described in the section that follow.

To set the default property for a Web access service object, invoke the method `SetDefaultSessionProperty`.

As with HTML templates, you can use `SetDefaultSessionProperty` on a Web access service object to set properties for any pages generated by a page builder method or page builder service object. The session property applies to all the page builder pages unless you override the session property for one or more pages with the `SetSessionProperty` method, as described under [“Page Builder Pages” on page 178](#).

The Session Property File

You can use a *session property file* to set and modify session properties for all templates (HTML files) that can be returned by a single Web access service. If you have multiple access services, you must create one session property file per access service.

A benefit of using this file is that you can change session properties for pages while a Web application is running. If the session property file is updated, the Web access service automatically incorporates the updates.

NOTE This file can not include any pages that are returned by the page builder service object.

Format The session property file expects a single row for each directory or file with the format: *path, session-property*. A comma is required between the two elements. White space is insignificant. You can include comments using either `//` or `--`.

The pathname must not contain `‘..’` (the shortcut for relative paths). This restriction is for security purposes; it prohibits someone from bypassing the session property specifications. A path or filename for a template that contains `‘..’` raises a standard iPlanet UDS “Invalid Request” exception.

Example

A sample file is shown in [Code Example 7-1](#).

Code Example 7-1 Sample Session Property File

```
/wc/, SESSION_UNSPECIFIED
/wc/catalog, SESSION_AUTOCREATE
/wc/orders, SESSION_REQUIRED
/wc/orders/login.htm, SESSION_UNSPECIFIED
```

Because the root level directory `/wc` has the `SESSION_UNSPECIFIED` session property, end users can get to the login page (`login.htm`). While processing the login page, the Web application creates a session. Users need not have a valid session to view any pages in the directory `/wc/catalog`, but must have a valid session in order to view any other pages from the directory `/wc/orders`.

Deleting session properties Do not delete session properties by deleting a row from the session property file. Instead, you must modify the entry in the session property file to re-specify the session properties in their new form.

Overriding Session Properties for Individual Pages

You can override session properties for individual pages, for subdirectories of HTML files, and for groups of pages generated by a page builder service object.

Pages in Files and Directories

You can override the session property for an individual page that was not returned by the page builder service object in one of two ways:

- Update the session property file to set the session property for the template.
- Set the session property programmatically, using the `SetSessionProperty` method of the `HTTPAccess` class.

Page Builder Pages

You can override the session property for a page builder page only by calling the `SetSessionProperty` method of the `HTTPAccess` class. Because this method uses the directory/file name convention to indicate groups of files, if you want this method to apply to a group of page builder files, you must use the directory/file name naming convention to name these pages.

For example, to set a session property for a group of non-page-builder pages stored in the `PAGEX` directory, you can invoke `SetSessionProperty` and specify `PAGEX`, the directory where these pages are stored. To apply this same method invocation to a group of page builder pages, start all their names with `'PAGEX/'`. The method invocation on the `PAGEX` directory would apply to pages named `PAGEX/A`, `PAGEX/B`, and so on, but would not apply to pages named `MYPAGE1` and `APAGE/REL`.

The following method invocation sets the `SESSION_REQUIRED` session property for the page builder pages named `PAGEX/A`, `PAGEX/B`, and so on:

```
myAccess.SetSessionProperty( 'PAGEX' ,SESSION_REQUIRED) ;
```

Working with State Information

State information is information that must be maintained for a *single user* of an application. For example, when a user of the SoftWear Catalog application places his first order, a shopper ID is stored as state information for the duration of the shopping session. All subsequent orders placed by the user are added to a single shopping basket, which the user can view at any time. When the shopper checks out, the shopper ID is no longer valid.

The data that constitutes state information is always application-dependent. State information may range from just a unique ID, to a record of the entire path of Web pages touched by an individual user. The purpose of state information is to capture and use whatever information is necessary to meet the needs of the user and the application.

You update state information programmatically, as the Web user moves from page to page, based upon actions taken by the user or by data entered or selected by the user. You can store short term information (such as the ID of an item being ordered in this session), and, if you use persistent storage for state information, you can store long term information that does not change between sessions (such as the user's credit card number).

You can track state information for templates processed by the scanner service and for pages built by the page builder service. A session object contains state information for each active session.

NOTE If you want to enforce any type of session management on a static page and you cannot use cookies to do so, then you must use either the scanner or the page builder service object to return the page with state management implemented. This is one circumstance in which you would use an iPlanet UDS service object to return a static page (and in which you should *not* maintain the static page on the Web server).

Defining What Constitutes State Information

You can manage (that is, store and retrieve) state information using either one of two general approaches:

- Use the predefined class `HTTPSession`, and manage session data using name-value pairs.
- Subclass the `HTTPSession` class, and manage session data using the custom attributes you define on the subclass. You would first define each attribute to be a potential type of session data.

Using the `HTTPSession` class The predefined `HTTPSession` class offers total flexibility with respect to what data can be stored and retrieved for a session. This flexibility is achieved because all session data is set using name-value pairs, and fetched using the name(s) used to save it. Thus, you can save *any* data for *any* session, and the data saved for one session can be totally different in value and in type from the data saved for another session. To retrieve data, you simply ask for the value associated with the name you used to store the data.

For example, you could use code like the following to add new session data:

```
request.CurrentSession.SetSessionData('myDataName', object);
```

Then you must subsequently cast the object to its runtime type.

Using a Subclass of `HTTPSession` for State Information

You may find that subclassing the `HTTPSession` class makes state management somewhat easier.

Advantages and disadvantages By defining specific attributes in your subclass, you can eliminate the use of name-value pairs for looking up and saving data into the session. You also gain some additional type checking when you define the data types for the attributes on the subclass, and you can add *arrays* of session data, if necessary. However, you should note that you can define *only one subclass* of `HTTPSession` per application. The effect is that you are predefining exactly what type of data constitutes session data; so while you gain additional checking, you lose flexibility in what data can be stored as session data.

If you use a subclass of `HTTPSession`, you might use code like the following to add new session data, and you do not need to cast the input object:

```
(MyHTTPSession)(request.CurrentSession).myDataName=object;
```

- ▶ **To use a subclass of `HTTPSession` to store session data**
 1. Create a subclass of `HTTPSession`.
 2. Define attributes and data types for the subclass for each type of session data that you may wish to save or retrieve for a session.
 3. Instead of using the `SetSessionData` and `GetSessionData` methods on the `HTTPSession` class, you set and retrieve values for the custom attributes.

Using Persistent Storage for State Information

By default, session management uses a session table to store state information. Because the session table is stored only in memory, if the partition in which it is created shuts down unexpectedly, the session table is lost and its data cannot be reconstructed. If this risk is too high for your application, you should use persistent storage for your state information. Another advantage of using persistent storage is that state information can persist across separate sessions for the same client.

To implement persistent storage you must write the state information to a file or database as well as the session table. Then, if the partition containing the session table shuts down, the state information can be retrieved from either the file or database. Similarly, if a partition replicated for failover starts up, it can read from the file or database and resume any sessions that were in progress when the session table was lost.

Using cookies While you can use cookies for persistent storage, this approach is less reliable and secure than maintaining the state information centrally on a server, primarily because users can enable or disable the use of cookies at will. Also, if one user initiated multiple browser sessions from multiple machines, the sessions would be interpreted as two distinct sessions rather than parts of the same session (originating from the same user).

Modifying URL Links for Session Management

If you use session management, you must modify all URL links in your application to use a fully qualified domain name rather than a simple node name. The following changes are required:

- Set the attribute `DefaultCookie` on your subclass of `HTTPAccess`.
- Modify links to use the iPlanet UDS system variable `$$FORTE.ExecURL`.

You should modify all links in all pages and templates, including static pages. If you do not modify links on static pages, those pages cannot participate in session management.

Your links should explicitly use the suffix “.htm” for the `TemplateName` parameter— the suffix is not assumed.

Setting the DefaultCookie Attribute

In your application initialization code, you should set the value for the `DefaultCookie` attribute for your Web access service. Setting this attribute is required to obtain consistent treatment of URL links by combinations of various Web servers and browsers; if you do not set this value, links executed by some combinations of server and browser may not work as expected.

Specifically, you must set the `Domain` attribute for the `HTTPCookie` value for the `DefaultCookie` attribute, to contain a fully qualified domain name. You can also set this attribute to allow multiple Web access servers in a single iPlanet UDS Web application to share state information.

You can specify the following additional attributes for the default cookie object:

- the `Expires` attribute, so the browser will save a cookie after the current browser session ends
- the `Secure` attribute, to assure that session IDs are only transmitted over secure encrypted channels

For example, the Init method for your Web access server might include the following:

```
DefaultCookie = new;
DefaultCookie.Domain = new;
DefaultCookie.Domain.SetValue(
    task.Part.OperatingSystem.GetEnv('WWW_COOKIE_DOMAIN'));
DefaultCookie.Path = new(Value='');
```

Project: ShopCartAccess • **Class:** PaymentAccess • **Method:** Init

Using the \$\$FORTE.ExecURL Variable in URLs

You must use the variable \$\$FORTE.ExecURL in all links in your templates and pages.

If you are upgrading an existing iPlanet UDS Web application to use WebEnterprise session management, you should modify existing links. Specifically, you should replace occurrences of *www.yourserver.com/cgi-forte/fortecgi[.exe]/* with \$\$FORTE.ExecURL.

The use of the \$\$FORTE.ExecURL variable generates an appropriate CGI or plug-in URL, sending all page requests through either fortectgi (or an iPlanet UDS plug-in) that consistently enforces session management.

Setting ExecURL WebEnterprise sets the value for \$\$FORTE.ExecURL automatically when you invoke the EnableAccess method, as described in [“The \\$\\$FORTE.ExecURL Variable” on page 87](#).

► To update pages and templates to use the \$\$FORTE.ExecURL variable

1. Update any form submissions to use \$\$FORTE.ExecURL.

For example, the ShopCart example submits a user’s login account and password to the session manager to be validated (passing through the Web access server which has enabled session management):

```
<form action="$$FORTE.ExecURL" method="POST">
```

HTML File: exstacct.htm

2. Update the URL links in your pages (the HTML tag HREF) to use `$$FORTE.ExecURL`.

Then, when a user navigates from one page to another, the session manager will validate that a session exists for each page which requires a valid session. For example:

```
<a href="$$FORTE.ExecURL?ServiceName=ShopCartService&
TemplateName=/Shopping/products.htm&category=$$currCategory.ID"
target="main">$$currCategory.Name</a>
```

HTML File: frcontent.htm

Although you must update all your URL links to use `$$FORTE.ExecURL`, when you view your Web pages prior to deployment, the links will display as missing or broken. This occurs because the value for the `$$FORTE.ExecURL` variable is only established at runtime.

Session IDs in URLs

When you have enabled session management, you never need to use a session ID explicitly. WebEnterprise automatically does what is necessary, based on whether or not the current browser session allows, or has disabled the use of, cookies.

If the Web client's browser does allow cookies (that is, the user has not explicitly disabled cookies), the session ID is passed as a cookie, and the session ID does not appear in URLs that are generated programmatically. As long as the browser allows the use of cookies, then the session ID is passed on each request and response as a cookie.

If an individual Web client has disabled cookies, WebEnterprise embeds the session ID in the URL itself, as a URL PATH_INFO variable derived from the variable `$$FORTE.ExecURL`. In this case, the generated URL is longer, because it includes an encoded version of the encrypted session ID, as shown in [Code Example 7-2](#).

Code Example 7-2 A URL with an embedded Session ID

```
http://www.forte.com/cgi-forte/fortecgi/frte_sid0A1E197E272D1C2321271A2818792678?
ServiceName=ShopCartService&TemplateName=/Shopping/products.htm&category=0001
```


Alternate Ways to Manage State Information

While the session management features of WebEnterprise provide a complete approach to managing browser sessions and state information, if you have a simple application you can implement state information using other approaches, briefly described below.

While these alternate approaches can be used in WebEnterprise, they require additional code and logic to construct and parse the state information, and are subject to restrictions, such as length and content of URLs. Generally speaking, it is preferable to use the session and state management features described earlier in this chapter.

Using Hidden Form Elements

If your state management requirements are relatively simple, you can use hidden form elements on the Web page to set, update, and communicate state information.

For example, the `Checkout` method in the `SoftWear` application stores the shopper ID as a hidden form element on the Checkout page. The shopper ID is stored on the client as a hidden form element as well as in the URL for links for other pages. The `Checkout` method gets the shopper's ID from the incoming Web request, uses that ID to get the shopper's shopping basket data, constructs an HTML page from this data, and adds the ID as a hidden form element on the outgoing Checkout page.

Using Page Parameters in Generated URLs

Another alternative is to embed state information as page parameters in URLs that are generated by the application and used as links to other Web pages in the application.

Using Cookies

You can use cookies to construct text strings containing session-specific information, and store the cookies on the Web client when you return a Web page; then you can retrieve, and use or modify as desired, the cookie information during subsequent interactions, even different sessions, with that user.

Cookies can offer a sort of persistent state information. If you use cookies to store state information, you do not risk the loss of the session table if a partition should shut down unexpectedly. However, cookies have some limitations, including the following:

- The content and length of cookies is limited.
- The number of cookies that can be placed on a user's machine is limited.
- Web users can disable the use of cookies entirely.

For instructions on how to use cookies, see the class reference for `HTTPCookie` in the iPlanet UDS online Help.

Partitioning and Deployment

This chapter provides information about how to partition, test, and deploy your iPlanet UDS Web applications. Topics covered include:

- service objects in iPlanet UDS Web applications
- default partitioning for iPlanet UDS Web applications
- modifying the configuration
- deploying the application

About Partitioning iPlanet UDS Web Applications

After creating your iPlanet UDS Web application, you should use the Partition Workshop to perform the following tasks:

1. Partition the final application for deployment.
2. Make the appropriate application distributions.

Configuring the application When you are ready to partition the application for deployment, use the `Configure as` command to configure the main project for the application. If your application includes a standard iPlanet UDS client (that is, a window-based client), use the `Configure as > Client` command. If your application does not include a standard iPlanet UDS client (only a Web page-based user interface), be sure to use the `Configure as > Server` command.

The following sections provide background information about the service objects in Web applications and the default partitioning scheme iPlanet UDS provides for Web applications.

About Web Application Service Objects

Chapter 3, “Setting Up a Web Application,” describes how you create the service objects for your Web application. When you partition your Web application using the Partition Workshop, iPlanet UDS creates a default configuration for the application based on those service objects.

Briefly, the service objects in your Web applications are:

Service Object	Description
Web access service object	The Web access service object, based on the <code>HTTPAccess</code> subclass, cannot be replicated. In addition, the Web access service object can be assigned only to a UNIX or NT server. It cannot run on Macintosh, VMS, or Windows.
page builder service object	This service object can be replicated.
scanner service object	This service object can be replicated.
business service object	Your application can define one or more standard iPlanet UDS service objects, which can be replicated as usual.

The following two sections provide further information about the Web access and page builder service objects.

Web Access Service Object

The HTTP project can run only on iPlanet UDS server platforms. Therefore, when you partition your Web application, you must assign the partition that contains the Web access service object to iPlanet UDS server node. You cannot assign your Web access service object to a Macintosh or Windows node.

The `fortecgi` program and the iPlanet UDS NSAPI plug-in use a single port number to locate the Web access service object; therefore, the Web access service object cannot be replicated for load balancing or failover. Because the Web access service object cannot be replicated, we recommend creating a separate page factory service object (page builder or scanner) to handle the bulk of the processing. The partition that contains the page builder or scanner service object can be replicated for load balancing to provide improved performance.

NOTE We recommend that the partition that contains the Web access service object be compiled, which provides improved performance for the partition.

CAUTION If you compile the partition that contains the Web access service object, you must be sure to install the compiled HTTP library in the deployment environments where your iPlanet UDS Web application will be running. The partition that contains the Web access service object needs to access the HTTP library, and if that partition is compiled, the HTTP library that it uses must also be compiled. See the *WebEnterprise Installation Guide* for information about the compiled HTTP libraries that are provided as part of the WebEnterprise product.

The default configuration for your Web application assigns all the service objects to a single partition. If your application does not require high performance, you can include your Web access service object on the same partition with your page builder or scanner service object. In this case, however, you cannot replicate the partition.

Page Builder Service Object and Scanner Service Object

The page builder service object and the scanner service object are standard iPlanet UDS service objects.

Using load balancing You can replicate the page builder and scanner service objects for load balancing or failover. If you have a high-performance application, we recommend that you assign the page builder and scanner service objects to their own partitions, which you replicate for load balancing.

Using code generation We also recommend that the partition that contains the page builder or scanner service object be compiled, which provides improved performance for the partition.

CAUTION If you compile the partition that contains a scanner object, you must be sure to install the compiled HTTP library in the deployment environments where your iPlanet UDS Web application will be running. The partition that contains the scanner service object needs to access the HTTP library, and if that partition is compiled, the HTTP library that it uses must also be compiled. See the *WebEnterprise Installation Guide* for information about the compiled HTTP libraries that are provided as part of the WebEnterprise product.

Default Configuration for Web Applications

When iPlanet UDS partitions an application, it assigns all compatible service objects to the same partition. If none of your service objects have replication turned on, all the service objects in your Web application will be on a single partition.

There are several changes you might wish to make to the default configuration:

- move the Web access service object to its own partition and mark that partition to be compiled
- move the page builder service object and scanner service objects to their own partitions, replicate those partition for load balancing and mark the assigned partitions to be compiled
- move the business service objects to their own partitions, replicate those partitions for load balancing, for failover, or for both load balancing and failover, marking the assigned partitions to be compiled if desired

See [“Modifying the Configuration” on page 191](#) for information about how to make these changes.

Modifying the Configuration

A Guide to the iPlanet UDS Workshops explains in detail how to modify a configuration.

This section provides some hints about typical changes you might need to make to the configuration for an iPlanet UDS Web application, including:

- creating a new partition for the Web access or page builder service object
- modifying a service object definition (turning on load balancing)
- replicating partitions
- assigning partitions to nodes
- moving partitions from one node to another
- creating a compiled partition (code generation)

Creating a New Logical Partition

If you need to place your Web access service object or page builder service object on a separate partition, you must create a new partition to contain it.

Use the New Logical Partition command to create a new partition for the Web access or page builder service objects.

► To create a logical partition

1. In the Logical Partition browser, select the service object you want to be in the new partition.
2. Select the Component > New Logical Partition command.

Modifying Service Object Definitions

You might need to change the definitions of the service objects in your configuration. For example, you might wish to turn on load balancing for the page builder or business service objects. Remember, you must not replicate the Web access service object.

► **To change the service object definition**

1. In the Logical Partitions browser, double-click the service object name to open the Service Object dialog.
2. In the Service Object dialog, update the appropriate properties. Properties that you are not allowed to change are read only.

To turn on load balancing, click the Load Balancing toggle.

Note that changing the definition of a service object to replicated will cause it to move from the original server partition to a replicated partition.

Assigning Partitions

After you have created a new logical partition for your Web access or page builder service object, you need to assign it to the appropriate node.

To assign a logical partition to a node, you simply drag the logical partition onto the appropriate node. The node must provide the resources necessary to run the particular partition.

Moving Partitions

To move a partition from one node to another, simply drag the partition to its new location. If the node is incompatible, iPlanet UDS displays an error dialog explaining why the service object cannot be moved.

Replicating Partitions

After you have changed the definitions of one or more service objects to allow replication, you can replicate the partition or partitions that contain them. A replicated partition is a partition that contains a service object defined as replicated for load balancing or failover. When a logical partition is replicated, you can assign it to any number of nodes in the environment.

► **To assign a logical partition**

1. In the Logical Partitions browser, select the logical partition you wish to assign.
2. Drag the logical partition to the node to which you wish to assign it.

The node must provide the resources necessary to run the particular partition.

Router partition When the service object in a partition is replicated for load balancing, iPlanet UDS automatically creates an extra partition called a router partition. The purpose of a router partition is to route the traffic between the partitions that are load balancing work for the service. Although the router partition is usually assigned to the same node as one of the server partitions that it is managing, it can be on any server node in the environment. You can move it if you wish.

CAUTION Do not move the Web access service object onto the router partition.

Replication Count property To replicate a partition on a given node, you use the Replication Count property on the Partition Properties dialog. This provides the total number of partitions to be automatically started on the node when the application starts.

► **To set the Replication Count property for an assigned partition**

1. Double-click the assigned partition name, or select the assigned partition and choose the Component > Properties... command.

The Assigned Partition Properties dialog opens.

2. On the Assigned Partition Property dialog, enter the total number of partitions to be started in the Replication Count field.

When the application starts, iPlanet UDS automatically creates the number of replicates you specify (unless the system manager overrides your setting).

Creating a Compiled Partition (Code Generation)

You can compile any partition in your Web application, except a partition that contains a Web access service object or scanner service object and that is assigned to a VAX/VMS node. iPlanet UDS does not provide a compiled version of the HTTP library for VAX/VMS platform. Therefore, a partition running on VAX/VMS that contains a Web access or scanner service object must be interpreted so it can access the interpreted HTTP library.

Compiled property To turn on code generation for a partition, you turn on the Compiled property for the assigned partition.

► To set the Compiled property for an assigned partition

1. Double-click the assigned partition name, or select the assigned partition and choose the Component > Properties... command.

The Assigned Partition Properties dialog opens.

2. On the Assigned Partition Property dialog, turn on the Compiled toggle.

After specifying that the partition is compiled, you may need to perform extra steps to produce the application distribution. See *iPlanet UDS Programming Guide* for information.

CAUTION If you compile the partitions that contain the Web access service object and scanner service objects, you must be sure to install the compiled HTTP library in the deployment environments where your iPlanet UDS Web application will be running. The partitions that contain the Web access service object and scanner service object need to access the HTTP library, and if these partition are compiled, the HTTP library that they use must also be compiled. See the *WebEnterprise Installation Guide* for information about the compiled HTTP libraries that are provided as part of the WebEnterprise product.

Using automatic compilation Note that if any of your partitions are marked as compiled and you wish to use the Auto-Compile option of the Make Distribution command, you must ensure that your environment is set up for automatic compilation. (See the *iPlanet UDS System Management Guide* for information on setting up your environment for automatic compilation.) If your environment is not set up for automatic compilation, you cannot use the Auto-Compile option and must compile the partitions by hand (see *A Guide to the iPlanet UDS Workshops* for information about compiling partitions).

Deploying the Application

To deploy a Web application, you must partition the application for each deployment environment in which it will run. After the application is correctly partitioned, you must create a separate distribution for each deployment environment.

► To deploy your Web application

1. In the Repository Workshop, double-click the main project for your Web application.
2. In the Project Workshop, choose the appropriate File > Configure as command.
3. In the Partition Workshop, select the environment from the environment drop list.

The Partition Workshop opens the configuration for the project or creates a default configuration if none already exists.

4. Modify the configuration as desired.

See [“Modifying the Configuration” on page 191](#) for information on modifying the default configuration.

5. When the configuration is complete, use the Make Distribution command to make the application distribution.

The application distribution is a representation of the application outside the repository—you use the application distribution to install the application in an environment.

Note that if any of your partitions are compiled and you wish to use the Auto-Compile option of the Make Distribution command, you must ensure that your environment is set up for automatic compilation. (See the *iPlanet UDS System Management Guide* for information on setting up your environment for automatic compilation.) If your environment is not set up for automatic compilation, you cannot use the Auto-Compile option and must compile the partitions by hand (see *A Guide to the iPlanet UDS Workshops* for information about compiling partitions).

6. The system manager must take the application distribution and install it in the environments (see the *iPlanet UDS System Management Guide* for information).
7. If you are deploying in more than one environment, select the next environment, make modifications to the configuration if necessary, and use the Make Distribution command to create the distribution. Repeat for any number of environments.

See the *iPlanet UDS Programming Guide* for complete information about partitioning applications and making distributions.

CAUTION Because all iPlanet UDS Web applications use the HTTP library, it must be installed in all deployment environments where the Web application is deployed. If any of the partitions in your Web application that access the HTTP library are compiled, the compiled form of the HTTP library must be installed in the deployment environment. WebEnterprise provides the HTTP library in both interpreted and compiled forms for installation in deployment environments. See *WebEnterprise Installation Guide* for information.

When you install the compiled HTTP libraries, you must perform the following platform-specific actions:

Platform	Compiled HTTP Library Setup
NT	Add \$FORTE_ROOT/userapp/http/cl0 directory to the "PATH" environment variable.
UNIX	After you have compiled a partition that accesses the compiled HTTP library, do <i>not</i> move the compiled HTTP library (for example, move \$FORTE_ROOT to a different disk or directory). If you do move the library after the partition is compiled, you must re-compile the partition so the partition and library can be correctly linked.
VMS	You must define an "HTTP" logical name in the Forte_Gbltable, translating to FORTE_ROOT:[USERAPP.HTTP.CL0]HTTP.EXE.

Managing iPlanet UDS Web Applications

This chapter describes several administrative aspects of iPlanet UDS Web applications, including:

- configuration options for fortectgi
- choosing between the fortectgi and fortensapi gateways
- autoregistration and manual registration
- the fortectgi.dat file
- using the Secure Sockets Layer (SSL)
- troubleshooting

About iPlanet UDS CGI and iPlanet UDS Web Server Plug-in Programs

In an iPlanet UDS Web application, the Web server communicates with the iPlanet UDS environment in one of two ways:

- through the Common Gateway Interface (CGI) API
- through an iPlanet UDS Web server plug-in, such as the iPlanet UDS Netscape Server API (NSAPI)

These two interfaces provide similar functionality using different APIs. The NSAPI interface was developed by Netscape and is supported by their current generation of Web servers; it is a higher performing alternative. Your iPlanet UDS Web interface can use either one or both of these interfaces. The following sections describe the relative advantages of each interface and options you have for setting up each program.

Every URL sent to the Web server contains a keyword indicating which interface to use (specifically, either the `fortecgi` executable or the `fortensapi` DLL), along with any additional information to be passed to the program.

Using `fortecgi` For each incoming request that specifies `fortecgi`, the Web server spawns a process that runs the `fortecgi` executable and forwards the additional URL information by using environment variables and standard input. The `fortecgi` program establishes a socket connection to an iPlanet UDS Web access service object, passes the request to it, receives an HTML response page, and passes it back to the Web server. The Web server then completes the request by returning the response page to the browser. For a picture of this process, see [Figure 2-2 on page 34](#).

The `fortecgi` program requires the creation of a new `fortecgi` process for each request and response. While process creation overhead is usually not substantial, in some applications with heavy use or many users, this overhead can represent a possible bottleneck.

Using a plug-in In addition to the CGI interface, Web server vendors such as Netscape and Microsoft also offer a plug-in API to reference external services. The plug-in API contains external interface routines in a dynamically linked library (DLL) form. The DLL is loaded into the Web server process the first time it is referenced and then remains resident. When an external request is received from a browser, the server dispatches a worker thread which invokes the plug-in DLL. The DLL uses plug-in API functions (rather than environment variables) to get the data required for the external service call and to return an HTML response to the client.

The plug-in API avoids the overhead of process creation that is inherent in the CGI mechanism, and therefore is a higher-performance interface. If you require optimum performance for many concurrent users, you should install the iPlanet UDS NSAPI plug-in in addition to the `fortecgi` program.

Certification information For installation information regarding the iPlanet UDS NSAPI plug-in, refer to the *WebEnterprise Installation Guide*. For information about what platforms, products, and versions are supported by the iPlanet UDS NSAPI plug-in, refer to the platform matrix at <http://www.forte.com/support/platforms.html>.

Example

The SoftWear example application can use either the `fortecgi` program or the iPlanet UDS NSAPI plug-in. The administration window, called Main Window and shown in [Figure 3-3 on page 68](#), contains a set of radio buttons in which you can choose whether to use the iPlanet UDS NSAPI plug-in or the `fortecgi` interface.

The ShopCart example also uses the iPlanet UDS NSAPI plug-in if you set the environment variable `WWW_PLUGIN_URL_BASE`.

Choosing between `fortecgi` and the iPlanet UDS NSAPI Plug-In

The decision to use the `fortecgi` program or the iPlanet UDS NSAPI plug-in depends primarily on the performance requirements of your application and the availability of iPlanet UDS NSAPI plug-in for your platform.

- If your iPlanet UDS Web application uses autoregistration to register the Web access servers, then *you must install `fortecgi`* even if the application uses the iPlanet UDS NSAPI interface.

`fortecgi` is required because automatic registration is only performed by `fortecgi`. However, in this case, all requests can be handled by NSAPI and `fortecgi` can be used only to perform registration.

- If you use manual registration to register the Web access servers *and* your application uses an iPlanet UDS Web server plug-in, you do not need to install `fortecgi`.

If your application has turned basic authorization on (on the Web server), then you must use manual registration.

With the exception of performing registration, both the `fortecgi` program and the iPlanet UDS plug-ins support the same functionality in an iPlanet UDS Web application. You do not forego any functionality by using one interface or the other, with the possible exception of which server platforms are certified to run the programs. For example:

- A Web server can support both `fortecgi` and plug-in connections simultaneously.
- An iPlanet UDS application can generate dynamic URLs that use either interface.

While the features supported are equal, some implementation details differ depending on whether your application uses `fortecgi`, a plug-in, or both. These implications are described in the following sections.

Setup Options for `fortecgi` and iPlanet UDS Plug-ins

The following sections describe some options you have with respect to the `fortecgi` and `fortensapi` programs. These options are:

- using automatic or manual registration to register a Web access service with `fortecgi` or the iPlanet UDS plug-in
- changing the port at which `fortecgi` can be reached

(See “[The `fortecgi` Executable](#)” on page 205 for information about changing the name or location of the `fortecgi` program.)

Security concerns and registration The `fortecgi` and `fortensapi` programs act as a Web server gateway for one or more Web access service objects. Each iPlanet UDS Web application requires one Web access service object to communicate with the `fortecgi` or `fortensapi` program. The Web access service object must register with the `fortecgi` or `fortensapi` program to enable Web clients to access the application, and can register in one of two ways:

- Autoregistration requires that basic authorization be off and that a secure Web server not be in use.
- Manual registration allows basic authorization to be on and a secure Web server to be used.

The `fortecgi` program detects how a service object is registering based on how the `EnableAccess` method is invoked. Multiple service objects may register with the same `fortecgi` program using either autoregistration or manual registration. For full information on the `EnableAccess` method, refer to the iPlanet UDS online Help.

You should choose a registration approach that meets your security requirements. If you use autoregistration, then it is possible for other entities to register with `fortecgi` if they can mimic the `EnableAccess` method, thereby exposing `fortecgi`, and possibly your Web server or iPlanet UDS application and data.

These two registration methods are described in more detail below.

Autoregistration

With autoregistration, whenever the Web access service object invokes the `EnableAccess` method, the service object is automatically registered with `fortecgi`. Likewise, the Web access service object automatically de-registers when it invokes the `DisableAccess` method.

The advantage of autoregistration is that it is automatic. However, it requires that basic authorization on the Web Server be off and it cannot be used with a secure Web server.

► To autoregister a Web access service object

1. In your iPlanet UDS code, invoke the `EnableAccess` method using the following syntax and including the `URLForForteCGI` parameter:

```
EnableAccess(serviceName=TextData, servicePort = integer,
URLForForteCGI=TextData)
```

If you use autoregistration, you can use the default port number of 1783 for the `fortecgi` program, or you can set a different port to be used for registration, as described below.

Setting a Port for Autoregistration

The environment variable `FORTE_CGI_REG_PORT` specifies the port number for the `fortecgi` program on the Web server and is used only for autoregistration. The Web access service object contacts the `fortecgi` program at this port to autoregister.

NOTE Do not confuse this port number with the one used to reach the Web access service object (see the iPlanet UDS online Help).

Limiting access to the port If you use autoregistration, you are advised to limit access to this port. If you do not limit access, then access to `fortecgi`, and therefore your iPlanet UDS application, may be compromised. The port number should be accessible only by nodes on the intranet, thereby limiting access to only those users considered “internal.”

Changing the port number If the variable `FORTE_CGI_REG_PORT` is not set and you are using autoregistration, then port number 1783 is assumed. You may set it to any port number that is available. Before changing the port number, you should verify with your system administrator that the port is available and also that it is only accessible to internal users. When changing the port number, you must change it in two places: on both the Web server and the iPlanet UDS server on which the Web access service object resides.

Autoregistration Requires fortectgi Program

In WebEnterprise, autoregistration of the Web access service object always uses the fortectgi program, whether or not you have installed and will use the iPlanet UDS NSAPI plug-in.

However, if your application uses manual registration, and you are using the iPlanet UDS NSAPI plug-in, then you do not need to install fortectgi at all.

Manual Registration

If your Web server has turned authorization on, then you must use manual registration. With manual registration you edit the fortectgi.dat file whenever the Web access service object starts to enter a row for that the Web access service object. To disable access, you edit the fortectgi.dat file to delete the row.

► To manually register a Web access service object

1. In your iPlanet UDS code, invoke the `EnableAccess` method without the `URLForForteCGI` parameter, as in the following:

```
EnableAccess(serviceName='myService', servicePort = 1789)
```

2. Edit the fortectgi.dat file to add a row identifying that service object. Enter a row in the format:

```
Service-Name Service-Port IP-address Protocol-version
```

The protocol-version parameter must be the same as that supported by the HTTPProject for the application; to get this value, see the ProtocolVersion constant in your HTTPProject and the iPlanet UDS online Help.

For example:

```
myService 1789 154.56.240.222 3
```

NOTE Use only one space between items. The items are not case-sensitive.

For a complete description of the items, see [“The fortecgi.dat File” on page 206](#).

3. Explicitly invoke one of the methods `SetPluginURL` or `SetExecURL`. This is necessary to set the value for the variable `$$FORTE.ExecURL` to be used in generated URLs.

Now users can access the application from the Web.

➤ **To manually de-register a Web access service object**

1. Delete the row that corresponds to the service object from the file.
2. In your iPlanet UDS code, use the `DisableAccess` method. See the iPlanet UDS online Help.

Use of fortecgi.dat by iPlanet UDS NSAPI Plug-in

To reduce file system overload, the iPlanet UDS Web server interface retains registration entries read for fortecgi.dat in an in-memory cache. Subsequent references to the iPlanet UDS application service are resolved without re-reading fortecgi.dat. The in-memory cache is reloaded whenever the fortecgi.dat registration file is modified.

Using an iPlanet UDS Web Server Plug-in During Development

With respect to application development, there is only one additional implication of supporting both the fortecgi interface and an iPlanet UDS Web server plug-in—the construction of target URLs. A given application must be able to respond to and generate URLs that use fortecgi or a plug-in, or both, depending upon which interface the application supports.

Specifically, a portion of the URL differs, depending on whether fortecgi or a plug-in should be called (see [“Using an iPlanet UDS Web Server Plug-in During Development” on page 203](#)). For an application that supports both interfaces, some logic must be added to determine which interface should receive the URL.

➤ **To use an iPlanet UDS Web server plug-in**

1. When you invoke the EnableAccess method on your Web access service, specify a value for the optional parameter called PluginURL.

The PluginURL parameter on the EnableAccess method enables applications to use an iPlanet UDS Web server plug-in. EnableAccess performs two functions:

- uses fortectgi to register your application with a Web server
- provides the application with the address of the Web server interface used to generate URLs (that is, the value for the ExecURL attribute and the \$\$FORTE.ExecURL variable)

If you do not specify a PluginURL parameter, EnableAccess causes the application to generate fortectgi interface URLs.

If you do specify a value for the PluginURL parameter, then that value is used to generate application URLs.

The iPlanet UDS Web server plug-in interface is invoked when a Web server request references a document of type “.forte.” By convention, the document name “web.forte” should be used. For example, the following code causes the MyService Web access service to use fortectgi to register with the Web server at www.forte.com, and to generate all URLs using an iPlanet UDS Web server plug-in:

```
MyAccessService.EnableAccess(  
    serviceName = 'MyService',  
    servicePort = 1234,  
    URLForForteCGI = 'http://www.forte.com/cgi-forte/fortectgi.exe';  
    PluginURL = 'http://www.forte.com/web.forte'
```

You continue to use fortectgi for registration of iPlanet UDS services. Other than that, the NSAPI interface can be used for everything.

Maintaining the iPlanet UDS Web Site Files

A major task of the iPlanet UDS Web programmer, perhaps in conjunction with the site Web Master, is the site management of all the files that make up a Web site. As your Web site applications grow in size and number, the directory structure and naming standards in effect can have a direct impact on both the performance and ease of maintenance of your applications.

The following sections describe some considerations and requirements regarding the various types of files used by an iPlanet UDS Web application.

The fortectgi Executable

Changing the location A typical location for the fortectgi program is on the same machine as the Web server. However, as long as the Web Server can reliably access the machine on which the fortectgi program is placed, the fortectgi program can be installed on any machine. See your Web server documentation for more information about locating and accessing cgi programs.

If you do change the location of the fortectgi program, you must assure that the URL `http://servername[:port]/cgi-forte/` maps to the CGI directory where fortectgi is installed. You specify this mapping during installation of WebEnterprise; for more information see the *WebEnterprise Installation Guide*.

Changing the name The standard installation results in the fortectgi program being installed and identified as “fortectgi.” You may need to change the name of fortectgi, to meet your particular requirements. For example, some platforms may require the file name to include the extension `.exe`.

You can change the name of the fortectgi program to any name, for example `fortectgi.exe`. If you do so, then you must also use the new name in the following situations:

- in the EnableAccess method
- in the entry point page
- in each URL to be forwarded to the fortectgi program. For example:
`http://www.forte.com/cgi-bin/fortectgi.exe?serviceName=.`

The fortensapi DLL

The actual name and location of the fortensapi DLL is referenced only in the Netscape Web server's configuration files. The DLL must be accessible to the Netscape Web server process, which must find and dynamically load it as needed.

Administrative Files

Your iPlanet UDS Web application may require as many as three different types of administrative files:

- fortecgi.dat file (required)
- session property file (optional)
- handler file (optional)

Maintenance aspects of these three types of files are described in the next sections.

The fortecgi.dat File

As the fortecgi program runs, it creates a small fortecgi data file to record and track the currently active Web access service objects. The default name for this file is fortecgi.dat. The fortecgi.dat file contains a row for each Web access service object currently registered with the fortecgi program. The format of each row is:

```
Service-Name Service-Port IP-Address Protocol-Version
```

For example:

```
easyWeb 1500 192.104.236.128 3
softwear 6001 192.104.236.60 3
```

NOTE The fortecgi.dat file is not case sensitive; however, *only one space* should appear between each entry, with no spaces at the beginning of a line.

The following table describes each element:

Element of fortectgi.dat File	Description
Service Name	The name of the service as specified in the EnableAccess method for the corresponding Web access service object. The service name is not case sensitive.
Service Port	The port number on which the service object will listen for Web requests, and as specified in the EnableAccess method.
IP Address	The IP address for the iPlanet UDS server machine where the service object is running. Note that the IP address is not specified in the EnableAccess method.
Protocol Version	The version of the protocol used between the Web access service object and the fortectgi or iPlanet UDS Web server plug-in interface. For each Web access service object, this value must match the ProtocolVersion constant in the HTTPProject. However, fortectgi and the plug-ins can communicate with multiple Web access services using <i>different</i> versions. For more information, see the iPlanet UDS online Help.

Rows are deleted (automatically or manually) from the file as service objects de-register, so disk space should not be a problem for this file. If no Web access service objects are currently running, the file exists but is empty.

Editing fortectgi.dat You can manually edit fortectgi.dat. In fact, you must edit fortectgi.dat if you use manual registration (see [“Manual Registration” on page 202](#)).

You have some flexibility with respect to this file; your options are described below.

Specifying a location Assuring file security for fortectgi.dat is an absolute requirement. You must make sure that only persons or processes requiring access have it. The file can be updated in only two ways: by a privileged user directly editing the file (manual registration) or by Web server (or its cgi surrogate) based on the establishment of a connection with the registered iPlanet UDS server (autoregistration)

The environment variable `FORTE_CGI_REG_FILE` sets both the path and file name for the `fortecgi.dat` file. If this variable is not set, the default is to write a file called `fortecgi.dat` to the same directory as the `fortecgi` program. You may use this variable to indicate a different path and/or file name. Setting this variable has no effect on the `fortecgi` program itself, other than to tell it where to find the file.

NOTE If you set or change `FORTE_CGI_REG_FILE`, you must re-start the Web server, so that both the Web server and the `fortecgi` program read the correct value.

Access requirements The type of access required by the file depends on whether you are using autoregistration or not.

- If you are using autoregistration, the file must be writable so that it can be updated whenever a service object registers or de-registers.
- If you are using manual registration, the file need only be readable by `fortecgi`; however, the file must be writable for you to edit it.

iPlanet UDS NSAPI plug-in The means used by the iPlanet UDS NSAPI DLL uses to access the `fortecgi.dat` file are platform-specific. See the *WebEnterprise Installation Guide* for more information.

Using a different directory than `cgi-bin` Some Web servers are configured to prevent writing to the `cgi-bin` directory. In these cases, the `fortecgi.dat` file must be written to a different directory, and the environment variable `FORTE_CGI_REG_FILE` must be set to that path and file name.

The Session Property File

See [“The Session Property File” on page 177](#) for a description of the purpose of the session property file.

File name and location To set the name of the session property file name use the `SetSessionPropertyConfigFile` method on your subclass of `HTTPAccess`; there is no default or expected name. If you have set the environment variable `FORTE_WW_DOCUMENT_ROOT`, then that value is used as the path; if you have not set the environment variable, you should specify a full path and file name.

The Handler File

The HandlerConfigFile attribute of the HTMLScanner class specifies the name of the handler file. This file contains identification information necessary to dynamically load libraries that contain tag handler classes. See [Code Example 5-1 on page 126](#) for an example and description of the handler file.

The handler file is required only if you will dynamically load libraries containing the tag handler classes that implement the TagHandlerIface interface. The handler file is not required if you use static registration for your tag handlers, or if you use a subclass of HTMLScanner for your tag handler class.

You must create one Handler file for each scanner that must dynamically load libraries of tag handlers. Then you must set the value of the HandlerConfigFile attribute of the scanner service object to the name of the file.

File name and location To set the name of the handler file you use two attributes of HTMLScanner. Set the value for the HandlerConfigFile attribute to the filename for the handler file. Its value is appended to the value for the DocumentRoot.attribute, if specified. If you have set the environment variable FORTE_WW_DOCUMENT_ROOT, then that value is used as the path; if you have not set the environment variable, you should specify a full path and file name.

The value should use portable name format (see the Framework Library online Help).

You can override this value with an environment variable or a command line argument. The priority of these settings is as follows, listed from highest priority to lowest:

Command line argument Overrides all settings. Use the command line argument `-configfile` when starting the partition that contains the scanner service object.

Environment variable Overrides HTMLScanner's HandlerConfigFile setting. Use the FORTE_WW_HANDLER_CONFIG_FILE environment variable.

Attribute setting Overridden by the other two techniques. Use HandlerConfigFile as described previously in this section.

Template Files

The HTML template files must be located in a directory that is accessible to the scanner service object so that the scanner service can read the templates at runtime.

When you have chosen a location for the templates, you must identify the top-most directory by setting the attribute `DocumentRoot` for your `HTMLScanner` subclass (see the next section).

The iPlanet UDS Document Root Directory

Most Web servers use the concept of a *Primary Document Directory*. All files that can be returned to Web browsers (html, and image files, for example) are presumed to reside at some level in this directory, and are located through a path name relative to the primary directory. For these Web servers, all relative path names are presumed to be relative to the primary directory.

WebEnterprise also uses a *document root* directory. For example, a session property can be set for a high-level directory, and the property can be inherited or overridden for lower-level directories or specific files.

The `DocumentRoot` attribute of the `HTMLScanner` subclass specifies the top level directory that will contain HTML template files and other administrative files. This directory must be available to the scanner service and should be on the same machine as the scanner service.

The value should use portable name format (see the Framework Library online Help).

You can override this value with an environment variable or a command line argument. The priority of these settings is as follows, listed from highest priority to lowest:

Command line argument Overrides all settings. Use the command line argument `-docroot` when starting the partition that contains the scanner service object.

Environment variable Overrides `HTMLScanner`'s `DocumentRoot` setting. Use the `FORTE_WW_DOCUMENT_ROOT` environment variable.

Attribute setting Overridden by the other two techniques. Use `DocumentRoot` as described previously in this section.

If multiple programmers are working on the same application, each programmer might want to use individual paths. Using the environment variable makes setting up separate paths convenient.

Graphic, Image, and Binary Data Files

You can store graphic files (and use them in Web pages) in two general locations:

- on the Web server
- on the iPlanet UDS server (as simple graphic files or iPlanet UDS objects)

There are advantages to either location. Either location is acceptable, as long as the graphic files can be accessed, and opened, by the Web server and by the page builder or scanner service responsible for returning the graphic.

Web server storage Storing graphics on the Web server can provide better performance, but may compromise security, or be less convenient to maintain. If you store any graphics on the Web server, then you must also be sure to map the alias for your high-level graphics directory in your Web server.

To embed a graphic stored on a Web server in a template, you use a line like the following in the template source:

```
<img src = "/ShopCart/images/buynow.gif" border = ... width = ...>
```

In this case, the alias ShopCart has been mapped on the Web server to the actual physical location of the graphics files, including the file buynow.gif.

iPlanet UDS server Storing graphics on an iPlanet UDS server (for example, binary data objects that are returned in an HTTPResponse object) might be slightly slower in performance, but prove easier to maintain. If you store any graphics on your iPlanet UDS server (not as iPlanet UDS objects, but simply as graphic files on the same machine), then you must locate the graphics in the directory identified by FORTE_WW_DOCUMENT_ROOT (or a subdirectory).

To embed a graphic stored on an iPlanet UDS server in a template, you use a line like the following in the template source:

```
<img src = "$FORTE.ExecURL?ServiceName=foo&
pageName=ReturnImage&fileName=buynow.gif" border = ... width = ...>
```

NOTE If you find that images are not being returned in a Web page (but instead only the standard image icon appears), then check that the Web server is using the correct image path.

Using an Administration Window

Both of the iPlanet UDS examples use an administration window to enable and set various settings when starting the SoftWear application. You might find the use of an administration window helpful as you build, test, or deploy an iPlanet UDS Web application. However, there is no particular requirement that one be used, other than to obtain information that may change at runtime.

Figure 9-1 shows the ShopCart administration window. The SoftWear administration window is shown in **Figure 3-3 on page 68**.

Figure 9-1 Administration Window for the ShopCart Application



Because the administration window is not a user window, it should only be accessible to administrators of the iPlanet UDS Web application.

Initialization Tasks

You can use an administration window to perform a number of “initialization” tasks. The following tasks are all considered initialization tasks, and are good candidates for inclusion in an administration window:

- start one or more Web access services by invoking the EnableAccess method
- specify the use of session management by invoking EnableSessionManagement

- specify whether to use the iPlanet UDS NSAPI plug-in in addition to the fortectgi program
- specify whether to use a secure Web server
- specify preferred settings for application characteristics, such as the template or image directories, encrypt key, port numbers, and so on
- set environment variables (the environment variables are described in [Appendix B, “Environment Variables”](#))

Some of these tasks are demonstrated in the two sample administration windows.

Security Considerations

Many iPlanet UDS Web applications contain sensitive data. There are many ways you can add security to an application.

Authentication security features can be used to determine and control *who* accesses the data. An authentication function typically checks that only identified users can access data, and that a given user is really that user. Most authentication schemes, including those used by Web servers, use user IDs and passwords for this purpose. Basic authentication is described below.

Often you also want to protect the data itself from being viewed or changed by an unauthorized user. In this case, you can use Netscape’s secure sockets layer to encrypt data that is transferred, on the Internet, between the Web server and Web client. Note that packets sent between the iPlanet UDS server and the Web server are not encrypted (with the exception of the Session ID if session management is enabled); however, those communications typically occur within a company’s own network, not over the Internet.

Using Basic Authentication

A security feature, called basic authentication (sometimes called basic authorization), can be used to limit which users can access Web pages. Basic authentication can be turned on at several levels, for example, for the entire Web server, or for users accessing fortectgi only. The effect of turning basic authentication on is that a user can only obtain Web pages by entering a valid user ID and password that are accepted by the authentication program. So, data itself is not protected (as in encrypted), but access to the data is limited.

Setting basic authentication to “on” for fortectgi has the following effects:

- The Web access service object must register with fortectgi using manual registration.
- Web clients can only access the iPlanet UDS Web application by entering a valid user ID and password.

Using basic authentication has no effect on the iPlanet UDS application.

Using Secure Sockets Layer

You have the option of using the secure sockets layer (SSL) if your Web users are using a Netscape browser and your Web server is running Netscape Enterprise Server or a server that supports Netscape’s SSL. SSL offers security features for Web communications, including encryption of packets sent between the Web server and Web client. SSL uses the HTTPS protocol instead of the HTTP protocol. The Main Window for the SoftWear application contains two radio buttons that can be toggled to turn on and off the use of SSL (see [Figure 3-3 on page 68](#)).

To take advantage of SSL requires a few extra steps, described below.

► To use SSL

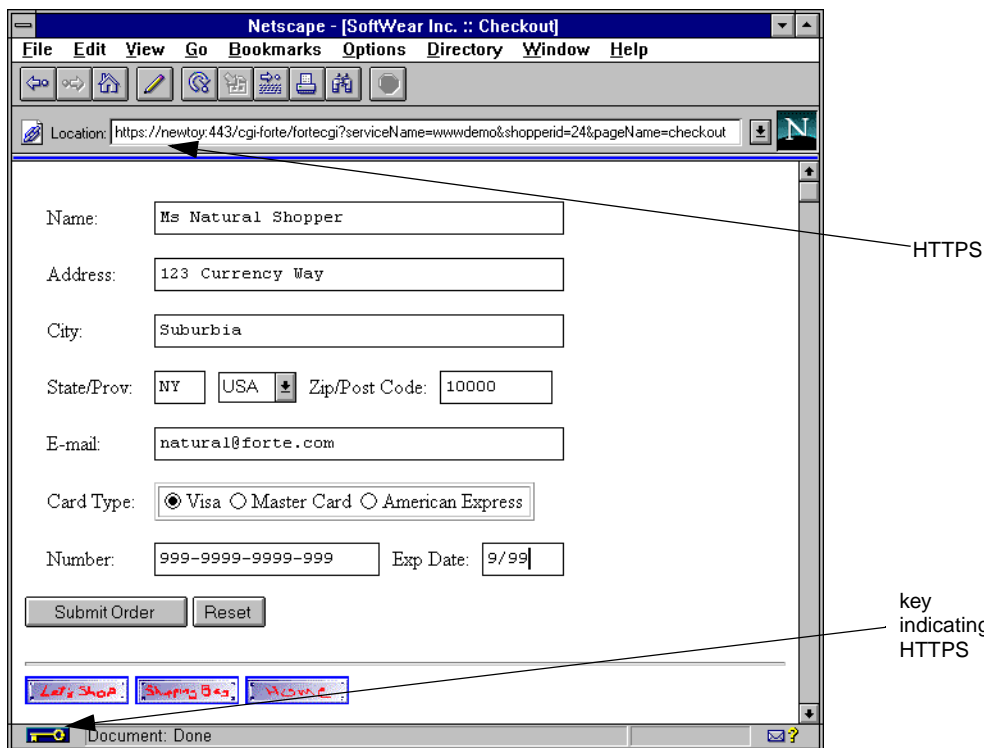
1. Modify all URLs that reference Web pages on the secure server.

For every URL to a Web page that is served by the secure server, you must modify the URL to use the HTTPS protocol rather than the HTTP protocol. In addition, if the Web server does not use the standard port number 443 for HTTPS connections, then you must also add in the non-standard port number.

2. You must use manual registration for the Web access service object.

You must invoke the EnableAccess method using only the first two parameters (serviceName and servicePort). See [“Manual Registration” on page 202](#).

Some effects of using SSL are more obvious to users than others. Users will not notice that packets are encrypted as they are sent between the Web browser and the Web server. Some users may notice that Web pages that are returned using HTTPS rather than HTTP show an unbroken key in the lower left-hand corner, as shown in [Figure 9-2](#).

Figure 9-2 Web Page with Key

Client Errors Reaching a Secure Server

If a Web client tries to request a web page (or a CGI program) from a Secure Web Server using HTTP instead of HTTPS, the client will fail to connect to the Web server. If the client is using Netscape Navigator, the browser returns a dialog box that says "There was no response. The server could be down or is not responding. If you are unable to connect again later, contact the server's administrator."

Diagnosing Problems with fortectgi or a Plug-in

After you have installed the WebEnterprise and imported the iPlanet UDS Web projects, you should be able to reach the fortectgi program or iPlanet UDS Web server plug-in from your Web browser.

► **To make sure that fortectgi or the iPlanet UDS plug-in is installed properly**

1. To test fortectgi enter a URL in the following format from your Web browser:

```
http://<server_name>/<cgi_directory>/<fortectgi_executable>
```

The *server_name* is the name of the Web Server node, and *cgi_directory* is the name of the directory on that node in which the fortectgi program resides, as in the following:

On UNIX:

```
http://www.forte.com/cgi-forte/fortectgi?
```

On NT/VMS:

```
http://www.forte.com/cgi-forte/fortectgi.exe?
```

This URL is like the URLs that the fortectgi program expects, with one exception. It does not include parameters with values for a serviceName, template or pagename, and so fortectgi, if reached, interprets it as an incomplete request.

2. To test a plug-in enter a URL in the following format from your Web browser:

```
http://<server_name>/<cgi_directory>/<fortectgi_executable>
```

The *server_name* is the name of the Web Server node, and *cgi_directory* is the name of the directory on that node in which the fortectgi program resides, as in the following:

On UNIX:

```
http://www.forte.com/cgi-forte/fortectgi?
```

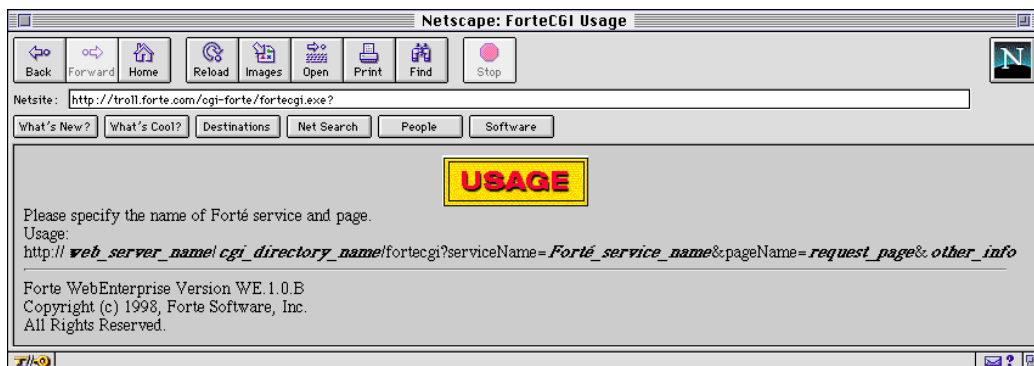
On NT/VMS:

```
http://www.forte.com/cgi-forte/fortectgi.exe?
```

3. Match the response to one of the following responses.

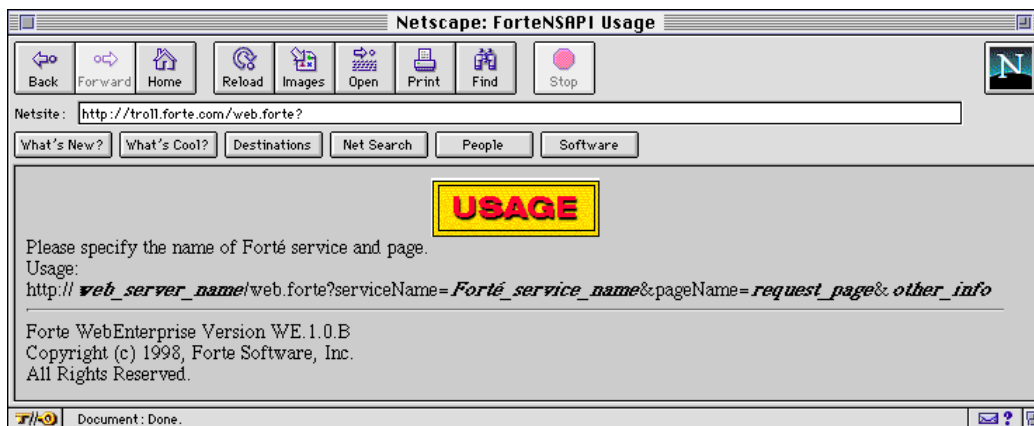
“Fortectgi Usage” Page

If fortectgi is installed correctly, authorization on fortectgi is off, and fortectgi receives a request with no parameters, it returns the ForteCGI Usage page, shown below.

Figure 9-3 The fortectgi Usage Page

“iPlanet UDS NSAPI Plug-in Usage” Page

If the iPlanet UDS NSAPI Web server plug-in receives a request with no parameters, it returns the Usage page shown below.

Figure 9-4 The iPlanet UDS NSAPI Plug-in Usage Page

“Attempt to Authorize Web User”

If fortectgi is installed correctly and with authorization turned on, and it receives a request with no parameters, you will see a dialog box prompting you for a user ID and password.

If you enter a valid user ID and password, you see the Fortecgi Usage page. If you enter Cancel, you see an “Unauthorized...” message. Either response indicates that fortectgi is installed properly. You should note that if your Web server uses authorization, then you must use manual registration; you cannot use autoregistration. For more information, see [“Manual Registration” on page 202](#).

“Not Found” Message

If you see a “Not Found” message, then there is most likely a mismatch in the name of the file (for example, fortectgi or fortectgi.exe) or the path name. The Web server was unable to locate the fortectgi program. Check the name of the fortectgi program and its path. Also refer to the *WebEnterprise Installation Guide*.

“Garbage Characters” on Screen

If fortectgi is installed incorrectly, you may see what appears to be a few “garbage” characters. This response indicates that the fortectgi program has been located in a directory that is not known as a cgi directory. The fortectgi program must be installed in a directory that is viewed by the Web server as a cgi directory; if you need assistance with this situation, ask your Web Master.

Troubleshooting Web Client Errors

This section lists some typical problems that Web clients and administrators may encounter along with suggestions for how to fix the problems. Errors seen by Web clients are displayed with an icon identifying the type of error, along with a more specific error message. The types of errors and icons are:

- Usage
- Request Failure
- Runtime Error
- iPlanet UDS Error

ForteCGI Usage Page

Problem The ForteCGI Usage page, shown in [Figure 9-3 on page 217](#), is returned when the URL contains no parameters. Either of the following URLs would result in this page being returned to the user:

```
http://www.forte.com/cgi-forte/fortecgi  
http://www.forte.com/cgi-forte/fortecgi?
```

Resolution If the URL was manually entered, retry it, making any necessary corrections. If the URL was automatically generated, then there may be an error in it which should be corrected. In particular, verify that the `serviceName` and `pageName` parameters are correct.

Client Request Failure Errors

All errors in this section appear on the Web client's Web browser screen along with the Request Failure icon.

“No ServiceName parameter found in the request URL”

Problem Fortecgi detected an incomplete URL, in which the `serviceName` parameter was missing. This error is also returned if the `serviceName` parameter name itself is misspelled, as in “`sericeName=...`”

Resolution Try the URL again, this time entering the `serviceName` parameter. If you do not know the `serviceName`, ask your iPlanet UDS Web administrator.

“Service Not Available”

Problem This message usually occurs when using manual registration. The fortecgi program finds the `serviceName` in the `fortecgi.dat` (or equivalent) file, but the service is unreachable. This situation occurs when a service has been disabled with the `DisableAccess` method, but the file has not been edited to remove the registration row. It can also occur if the file is updated to add a row, but the service has not been started.

A different cause for this error is that the `fortecgi` program failed to connect to the service due to network problems, such as inadequate privileges to access the port or if the Web access service object is no longer running but is still registered.

Resolution Try again later or contact your iPlanet UDS Web administrator to enable the service. If you believe the error is caused by network problems, contact your iPlanet UDS Web administrator.

“Service Not Found”

Problem The `serviceName` specified in the URL is not currently registered with the `fortecgi` program that received the request. The `serviceName` is not found in the `fortecgi.dat` file (or equivalent). This might be because the `serviceName` was misspelled, or because the service has not been enabled, or it might be because the iPlanet UDS application is not running.

Resolution Check the spelling of the `serviceName` and that the specified service has been enabled. If necessary, check that the application and the node are running.

“The iPlanet UDS service you requested is busy, please try again”

Problem The `serviceName` specified in the URL is enabled but currently busy.

Resolution Try again later. If you continue to receive this message, contact your iPlanet UDS Web administrator.

fortecgi Runtime Errors

Web client runtime errors that show the Runtime Error icon are typically due to various socket errors.

“Socket error:”

Problem Fortecgi has socket communication problem with the iPlanet UDS server. Following the above error is a more specific message, such as:

- `recv () #9` (indicates a failure to read from the socket)
- `send () #8` (indicates a failure to write to the socket)

Resolution Try again later. If you continue to receive this message, contact your iPlanet UDS Web administrator.

Client iPlanet UDS Errors

Client iPlanet UDS errors may occur at four levels, and are returned to the client with an icon indicating the level, and with the iPlanet UDS error text. There is no standard error text because the text is the exact iPlanet UDS error.

Problem An iPlanet UDS exception occurred during the Web client's use of an iPlanet UDS Web application. The level of severity of the error is indicated by which of four icons appears (the levels in increasing severity are Warning, User Error, iPlanet UDS Error, and Fatal Error). These messages are HTML versions of iPlanet UDS exceptions. The exceptions are raised in the iPlanet UDS application and may indicate a problem with the application itself.

Resolution Report the iPlanet UDS error message and events leading up to it to the iPlanet UDS Web administrator.

Object with NIL value returned by iPlanet UDS

An error to this effect has been returned when cookies are disabled, or are used, but host names are not fully qualified in the HTTP request URLs. When using WebEnterprise you should make sure that all URLs that are generated by the application and used for links use a fully qualified domain name.

Client Security Errors

“Status Code 401: Unauthorized...”

Problem The Web client is unauthorized to access the requested Web page. This occurs if the Web server has turned on basic authorization. In this case, the Web server expects a username and password, which are not passed on by the fortectgi program.

Resolution Either turn off basic authorization or obtain a username and password that will allow you to use the secure application.

Troubleshooting Web Administrator Errors

Most administrator errors occur during registration and de-registration of the Web access service object. These errors are of the type Request Failure, although the Request Failure icon does not appear. The error message appears on the administrator's iPlanet UDS windows, not on a Web browser screen.

“De-Registration Failure: Cannot write to fortectgi data file”

Problem The fortectgi program has inadequate permission to write to the file. The fortectgi program has located the file but cannot write to it to de-register the service.

Resolution Check permissions on the fortectgi data file and, if necessary, on its directory.

“De-registration Failure: fortectgi data file not found”

Problem The fortectgi program cannot locate the fortectgi data file (fortectgi.dat or its equivalent) so it cannot de-register the service object currently trying to de-register. The fortectgi data file may have been inadvertently deleted or moved, or the environment variable FORTE_CGI_REG_FILE may have been changed.

Resolution Check the existence of the fortectgi data file. If necessary, re-set the value of the variable FORTE_CGI_REG_FILE.

“Incoming registration or de-registration message is invalid”

Problem This message usually indicates a network problem.

Resolution Try your operation again later. If you continue to receive this error, call iPlanet UDS technical support.

“Registration Failure: Cannot create fortectgi data file”

Problem The fortectgi program cannot create the fortectgi data file (fortectgi.dat or its equivalent) so it cannot register the service object currently trying to register.

Resolution Check permissions on the directory where the fortectgi data file would be created. Make sure that the directory is writable. If necessary, verify the setting for the environment variable FORTE_CGI_REG_FILE, which specifies the path and name for the file.

“Registration Failure: Cannot write to fortectgi data file”

Problem The fortectgi program has inadequate permission to write to the file. The fortectgi program has located the file but cannot write to it to register the service.

Resolution Check permissions on the fortectgi data file and, if necessary, on its directory.

“Registration Failure: Duplicate iPlanet UDS server with the same port number”

Problem The fortectgi program found an entry in the fortectgi data file (fortectgi.dat or its equivalent) with the same port number and IP address as were given for the service currently trying to register, but with a different serviceName. The fortectgi program cannot register two services if they both use the same port number and IP address.

Resolution Check that you have specified the correct port number and IP address. If they are correct, check the serviceName. If necessary, verify the setting for the environment variable FORTE_CGI_REG_PORT for the two conflicting services.

Calling iPlanet UDS Technical Support

As you use WebEnterprise you may encounter problems or have questions. It will help expedite answers to your questions if you have information about your Web server available whenever you call.

► To obtain your Web server name and version

1. Use Telnet, as follows:

```
%telnet host_name port
```

host_name is the name or IP address of the Web server and *port* is the public access port to that Web server (typically port number 80).

2. Type:

```
HEAD /HTTP/1.0
```

3. Press return twice. You will see a response similar to:

```
HTTP /1.0 200 Document follows
Date: ...
Server: NCSA/1.5
Content-type: text/html
Last-modified: ...
Content-length: ...
```

You may be asked to provide the information found on the “Server” line.

The WebEnterprise Release Number

You may occasionally need to know the release number for the WebEnterprise kit that you are using, for example, if you need to call iPlanet UDS Technical Support.

To see the release number for the kit installed on the iPlanet UDS server side, open the Project Workshop for the HTTP project and check the value for the constant named ForteWebEnterpriseVersion.

To see the release number for the fortectgi and iPlanet UDS NSAPI plug-in that are installed on the Web server, enter a URL without a page or template name. You should see the standard iPlanet UDS “Usage” page, which contains the version number of WebEnterprise running on the server.

Example Programs

This appendix describes the sample programs included with WebEnterprise. Topics include:

- the EasyWeb example
- the SoftWear example
- the ShopCart example
- the SQLDemo example

About the Examples

WebEnterprise contains several examples to demonstrate various aspects of the product.

EasyWeb Example

The EasyWeb example shows the minimum requirements for adding Web connectivity to an iPlanet UDS application. For details, see [“Quick Tutorial: EasyWeb” on page 50](#). As you recreate EasyWeb, you gain an understanding of the interaction between the Web and iPlanet UDS interfaces.

SoftWear Example

The SoftWear example demonstrates several features of a Web interface to an iPlanet UDS application. Many examples in this manual are from the SoftWear application.

The components (projects, service objects, classes, and so on) demonstrated by the SoftWear example are based upon the recommended structure described in [Chapter 3, “Setting Up a Web Application.”](#) It uses a small “database” (actually a file created when the demo is run) to show how a Web page can be dynamically constructed based upon specific criteria entered by a Web user, such as a catalog item ID. SoftWear allows you to use either the iPlanet UDS NSAPI plug-in or fortectgi as your Web server interface.

Turn to [“Components of the SoftWear Example” on page 227](#) for information about the components of the SoftWear application. Turn to [“Installing the SoftWear Example” on page 230](#) for instructions on installing and running the example.

The ShopCart Example

The ShopCart example illustrates how to use iPlanet UDS templates and how to take advantage of session and state management.

The following features are illustrated by the ShopCart example.

- submitting forms
- traversing URL links
- using FORTE EXECUTE tags with parameters and simple and complex result sets
- creating result sets
- using the FORTE ITERATE tag to iterate over a result set list
- using the FORTE IF and ELSE tags
- using the FORTE REDIRECT tag
- using and registering static handlers
- using a default handler (HTMLScanner itself is an implicit handler)
- invoking templates from pagebuilders
- creating a session (using session property SESSION_AUTOCREATE)
- setting a default session expiration timeout interval
- adding data to a session, using both custom session object attributes and name-value pairs.
- retrieving data from the current session

- deleting a session
- implementing a custom session class
- sharing sessions across services and web servers
- detecting cookie support
- tracking a session with and without cookie support

SQLDemo Example

The SQLDemo example shows how the HTMLSQL project can be used to query a relational database and quickly display the query results in a Web page. Refer to [“Using the SQLDemo Example” on page 240](#) for instructions on installing and running this example.

Components of the SoftWear Example

The structure and components of the SoftWear example reflect the recommended structure for iPlanet UDS applications that will also support Web clients. This example demonstrates how components (for example, user windows) can be shared by the iPlanet UDS client side and the Web client side, as well as how and when they must be slightly modified.

Projects and .pex Files

The SoftWear example is included on the iPlanet UDS WebEnterprise CD in several .pex files that you can import. The table below summarizes the .pex files, the projects they create, and the purpose of each project. You must import all files, in this order, in order to run the SoftWear example.

Import File	To Get the Project	That Contains
www1.pex	WWWCatalog	The business service object, CatalogService
www2.pex	WWWSharedWindows	Web windows and two embedded windows that are used by the other user windows.
www3.pex	WWWCatalogPageBuilder	The PageBuilder service object and the methods that construct the various Web pages for the SoftWear example.

Import File	To Get the Project	That Contains
www4.pex	WWWCatalogAccess	The Web access service object for the SoftWear example.
www5.pex	WWWCatalogWindows	Windows designed for an iPlanet UDS application (no Web access) but which are converted to Web pages using the WindowConverter method.

The projects WWWCatalog and WWWCatalogWindows are the “pre-existing” projects, or the projects that support the iPlanet UDS application without Web access. The two projects WWWCatalogPageBuilder and WWWCatalogAccess are used only by the Web interface. Finally, the WWWSHaredWindows project contains windows that are used by both the Web and the iPlanet UDS clients.

iPlanet UDS Windows and Corresponding Web Forms

The user windows and corresponding Web pages are shown in the following table:

Window Name	Web Form Name	PageBuilder Method	Description
CatalogListWindow	prodlist	BuildProdList	Show summary of all software items.
CatalogPageWindow	prodDetail	BuildProdPage	Show detail on one item.
ShoppingBasketWindow	AddToBag	AddToBag	Enter order information for an item.
ShoppingBasketWindow	ShowBag	ShowBag	Show item(s) in basket
(none)	EmptyBag	EmptyBag	(after empty bag, return to prod list window)
OrderWindow	Checkout	Checkout	Decide whether to buy or cancel.
ConfirmationWindow	Confirm	Confirm	Confirm placement of order with order #.

Window Name	Web Form Name	PageBuilder Method	Description
MainWindow	none	none	Initialize application and start WWW access
none	(homepage .html)	(none - simple html file)	Entry point page for Web clients to start shopping.

Page Builder Methods Used by SoftWear

The CatalogPageBuilder class of the SoftWear example contains several examples of methods used to construct pages:

Method	Techniques Used
AddToBag	Does not create a page, but adds another item to a shopping basket on the ShowBag page
BuildProdList	Builds a list of products, each with a link (anchor) to the detail page for that product
BuildProdPage	Uses WindowConverter on the CatalogPageWindow window class
ButtonSet	Defines all buttons for the SoftWear application, with a link and graphic for each
CheckOut	Uses WindowConverter on the NestedOrderWindow window class
CompanyLogo	Create a graphic element to be embedded in multiple Web pages
Confirm	Uses LoadParameters method
EmptyBag	Deletes items from bag and returns the ProdList page
GetImageFile	Given image name, builds complete path to locate and return image
ShowBag	Uses WindowConverter

The SoftWear Data Files

The text and data used by the SoftWear application are stored in a number of files in the directory `$FORTE_ROOT/tmp/cattmp`. The `cattmp` directory is initially created when you click on the Make Database button in the Main Window, the first time you run the SoftWear application. In the `cattmp` directory you will find a number of files used to track shopperIDs, orderIDs, and to store product and shopper information.

You need only click on Make Database once; then you can enable and disable access as you like and the files remain. As long as the files are not recreated, the shopper and order IDs will continue in sequence.

If you click on Make Database again in the Main Window, the data files are recreated, and the shopper and order IDs will start over again.

Installing the SoftWear Example

You must install files on both the Web Server and iPlanet UDS Server to run the SoftWear example. On the Web Server you install images and the entry point Web page. On the iPlanet UDS server you install files to create the classes and methods to generate Web pages for the Web users, as well as iPlanet UDS windows for the iPlanet UDS clients.

► To install and run the SoftWear example

1. Find the SoftWear files on your Web server.

The installation of iPlanet UDS WebEnterprise includes the installation of the SoftWear files. A straightforward installation will result in a path like the following on your Web Server:

```
netscape/users/forte/examples/softwear
```

The `softwear` directory has two subdirectories:

Subdirectory	Contents
/docs	Contains the file <code>homepage.htm</code> that is the entry point Web page for the SoftWear application. It is a static page and is not created by the page builder service object.
/images	Contains several <code>.gif</code> files (images) used by the SoftWear example.

2. Find the SoftWear files on your iPlanet UDS server.

The installation of iPlanet UDS WebEnterprise includes the installation of the SoftWear files. A straightforward installation will result in a path like the following on your iPlanet UDS server:

FORTE_ROOT/install/examples/web/softwear

The softwear directory has two subdirectories:

Subdirectory	Contents
/catsrc	Contains the data used in the SoftWear application. It plays the role that a database would in a typical application.
/pex	Contains the five pex files needed to recreate the SoftWear application.

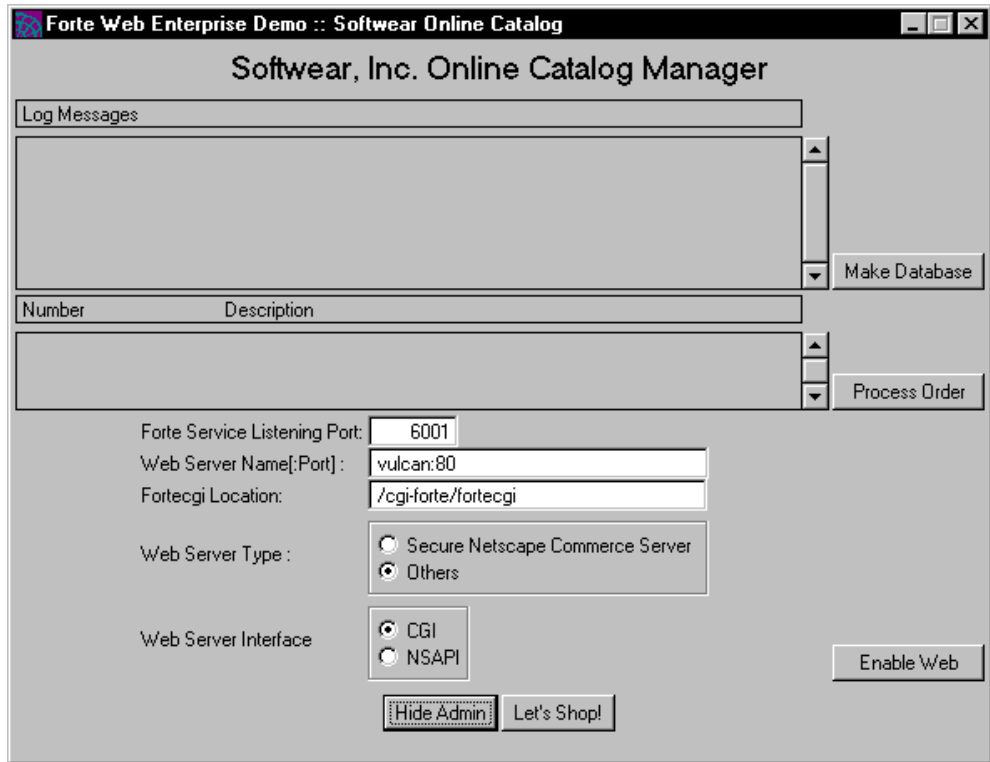
3. Import the .pex files for the SoftWear example.
 - a. In the iPlanet UDS Workshops, open a workspace that contains the Web projects HTTP, HTML, and HTMLWindow in the Repository Workshop. These projects are required by the SoftWear example and were imported during the installation of WebEnterprise.
 - b. Use the Import... command under the Plan menu option to import the .pex files in numeric order, starting with `www1.pex`.
4. In the Repository Workshop, open the WWWCatalogWindows project. Select the Test Run command from the Run menu, or click the Run icon on the toolbar.

An iPlanet UDS window appears with two buttons: Show Admin and Let's Shop.

5. Click the Show Admin button.

The Main Window for SoftWear appears. The Main Window is an administration window used to start Web access, oversee shopper activity, and approve orders. It is not the entry point Web page seen by Web users.

Figure A-1 Administration Window for SoftWear Application



6. Click the Make Database button.

This creates the SoftWear data files. Now you can use the SoftWear application as an iPlanet UDS client. At any time after you have made the database, you can click on Let's Shop to use the iPlanet UDS client windows.

The remaining steps are necessary to use SoftWear as a Web client.

7. In the Main Window enter information specific to your site. You must change the Web server name; the other fields have defaults.
 - o iPlanet UDS Service Listening Port: The port number 6001 appears. If this is not the port number you want to use for the Catalog Access service object, enter the desired port number here.
 - o Web Server Name [:Port]: The server name and port number vulcan:80 appear. Enter the name of your server, and enter a port number if you want to use a different port than 80.
 - o Fortecgi Location: `/cgi-forte/fortecgi` is displayed. You do not need to change this if your fortectgi executable is running on a Unix machine. However, if your fortectgi executable is running on an NT machine, change fortectgi to fortectgi.exe.
 - o Web Server Type: If your Web server is running Secure Netscape Commerce Server choose this button; the default is for all other servers. Note that you can only use Secure Netscape Commerce Server if you are also using the Netscape Navigator browser, because it uses Netscape's Secure Sockets Layer (SSL) and the HTTPS protocol, which are only supported by Netscape Navigator and Netscape Commerce Server.
 - o Web Server Interface: You can select the fortectgi or the iPlanet UDS NSAPI plug-in. You must have installed the iPlanet UDS NSAPI plug-in as described in the *WebEnterprise Installation Guide* for this selection to take effect.
8. Skip this step if you are not using Secure Netscape Commerce Server. If you checked Secure Netscape Commerce Server, complete the following steps:
 - a. In the iPlanet UDS Service Listening Port, enter port number 443 (the standard port for SSL) or the port number that you wish your Web server to use.
 - b. You must use manual registration. Edit the file `fortecgi.dat` to add a line in the format:


```
Service-Name Service-Port IP-Address
```

Make sure there is no leading space and that only one space separates the entries.

9. Click the Enable Web button. The status line indicates that Web access is being enabled; depending upon your network and server configuration, this may take a short time. After Web access is enabled, the Enable Web button text changes to Disable Web.
10. To access the application as a Web client, use your Web browser to access the SoftWear home page, using the URL:
`http://<server_name>/forte/examples/softwear/docs/homepage.htm`
 Note that the home page provides three links to the application: one for Unix, one for NT, and one for NSAPI. If you are running your `fortecgi` executable on a Unix machine, select the link for Unix. If you are running your `fortecgi` executable on an NT machine, select the link for NT. If you are using NSAPI, select the link for NSAPI.
11. When you are finished using the application, click on Disable Web in Main Window

Components of the ShopCart Example

The ShopCart example includes .pex files, HTML files, and image files. This example demonstrates how to use the iPlanet UDS templates and session and state management.

Projects and .pex Files

The ShopCart example contains several .pex files that you can import. The table below summarizes the .pex files, the projects they create, and the purpose of each project.

Import File	To Get the Project	That Contains
classes.pex	ShopCartClasses	The base classes for the application, the business service object, ShoppingService, the tag handlers, and the HTTPSession object.
access.pex	ShopCartAccess	The HTMLScanner classes and the HTTPAccess classes for the shopping and payment components of the application, and their associated service objects.
app.pex	ShopCartApp	The control window for the application that lets you initialize and enable Web services.

The ShopCart HTML Files

The ShopCart example contains a number of HTML template files which contain iPlanet UDS tags. The HTML template files are read directly by iPlanet UDS, not by your Web server. An HTML home page is also provided. The home page contains no templates, and will be read by your Web server.

The ShopCart Image Files

A few image files are also provided, for display in the browser windows. The image files are read directly through the Web server.

Distribution of ShopCart Components

The *WebEnterprise Installation Guide* describes how to install the example files. The ShopCart .pex files and HTML files are placed under the FORTE_ROOT path, because these files are accessed by your iPlanet UDS server. The image files and the home page are placed under your Web server's root directory, because they are accessed by the Web server, and not by iPlanet UDS.

Installing and Running the ShopCart Example

Description The ShopCart application illustrates the use of iPlanet UDS tags in HTML files and session and state management in a simple Web shopping application. The application allows you to select items for your shopping cart, view the cart contents, and pay for the items.

Pex Files shopcart/pex/classes.pex, shopcart/pex/access.pex, shopcart/pex/app.pex.

Mode Stand-alone or Distributed.

Special Requirements Must have the WebEnterprise HTTP Library installed in your workspace, must have a web server running that can access the fortectgi executable, must have a web browser.

► **To install the ShopCart example**

1. Find the ShopCart HTML and pex files below the following directory:

\$FORTE_ROOT/install/examples/web/shopcart

The shopcart directory has two subdirectories:

Subdirectory	Contents
/html	Contains two subdirectories, payment/ and shopping/, which contain the HTML files used by the ShopCart application.
/pex	Contains the three pex files needed for the ShopCart application.

If the files are not there, refer to the installation procedures described in the *WebEnterprise Installation Guide*.

2. Set the following four environment variables:
 - a. The iPlanet UDS document root directory, specified by FORTE_WW_DOCUMENT_ROOT, is a WebEnterprise environment variable that you set on the iPlanet UDS server:

```

-- On Unix:
FORTE_WW_DOCUMENT_ROOT
$FORTE_ROOT/install/examples/web/shopcart/html
-- On NT: (Note that the double backslash is required when
-- setting environment variables using DOS command syntax.)
FORTE_WW_DOCUMENT_ROOT
c:\\forte\\install\\examples\\web\\shopcart\\html

```

The ShopCart application will look for the HTML files it needs in this directory and its subdirectories.

The remaining three environment variables are *not* iPlanet UDS specific. You set these environment variables on the Web server node.

b. The CGI path for the ShopCartService:

```

-- On Unix:
-- Use fortecgi on Unix
WWW_CGI_URL_BASE
http://yourMachine.yourDomain.com/cgi-forte/fortecgi

-- On NT:
-- Use fortecgi.exe on NT
WWW_CGI_URL_BASE
http://yourMachine.yourDomain.com/cgi-forte/fortecgi.exe

```

c. The CGI path for the PaymentService:

```

-- On Unix:
-- Use fortecgi on Unix
WWW_CGI_URL_BASE2
http://yourMachine.yourDomain.com/cgi-forte/fortecgi

-- On NT:
-- Use fortecgi.exe on NT
WWW_CGI_URL_BASE2
http://yourMachine.yourDomain.com/cgi-forte/fortecgi.exe

```

You can run both services on a single Web server. Alternatively, you can run each service on a different Web server. Those Web servers could be on the same machine, or on different machines.

d. The cookie domain name:

```
WWW_COOKIE_DOMAIN .yourDomain.com
```

Set this to your domain name. Note that the domain name must contain a minimum of two dots, so you must include the leading dot.

3. This step is only necessary if you want to run ShopCart using NSAPI. To run under NSAPI, you must set one additional environment variable. Before doing so, make sure your WWW_CGI_URL_BASE and WWW_CGI_URL_BASE2 environment variables are set to the same CGI path. This is not a restriction in the product, but rather a restriction in the way NSAPI is implemented in this example. Now set the environment variable:

```
-- Use NSAPI on Unix or NT
WWW_PLUGIN_URL_BASE
http://yourMachine.yourDomain.com/web.forte
```

Note that after setting this environment variable, NSAPI will automatically be used by ShopCart. You do not need to make changes to the home page or the control window.

4. Find the ShopCart image files. If you followed the installation procedures in the *WebEnterprise Installation Guide*, the image files and the home page will be located below your Web server's root directory. For example, if you have Microsoft's Peer Web Server running, the image files might be in the following directory:

c:\InetPub\forte\examples\shopcart\images

If you have the Netscape Web Server running, the image files might be in the following directory:

/netscape/users/forte/examples/shopcart/images

These paths are based on where you set the root directory for your web server.

5. Set two aliases in your Web server:

Directory	Alias
C:\InetPub\forte\cgi_bin	/cgi-forte
C:\InetPub\forte\examples\shopcart	/shopcart

The `cgi-forte` alias must point to the directory where your `fortecgi` executable resides. The `shopcart` alias must point to the directory in which you have placed the `image` directory and the `doc` directory and their files for the ShopCart example.

6. Bring up iPlanet UDS. If the iPlanet UDS launcher was already running at the time you set the environment variables, you should terminate it before starting iPlanet UDS. Doing so will ensure that iPlanet UDS will pick up the new settings.
7. If it is not already there, import the HTTP library into your workspace. Then import the `.pex` files for the ShopCart example in the following order:
`classes.pex`, `access.pex`, `app.pex`.

► To Run the ShopCart example

1. In the Repository Workshop, select the ShopCartApp project. Click the Run icon on the toolbar.

A Shopping Cart Application control window appears.

2. Click the Initialize Web Services button. Then check the ShopCartService Enabled checkbox. It may take a moment before your cursor returns. Then check the PaymentService Enabled checkbox.
3. Bring up your Web browser. Make sure your `cgi-bin` path is configured correctly on your Web server and that you can actually execute the `fortecgi`. For example, based on the configuration described above, you would enter the following URL in your browser:

```
http://yourMachineName/cgi-forte/fortecgi.exe
```

After clicking return, you should see a Usage message.

4. Now enter the URL for the ShopCart home page:

```
http://<yourMachineName>.<yourDomainName>.com/shopcart  
/docs/homepage.htm
```

You should see a browser window with the heading: ShopCart, Inc.

This page provides two links to the ShopCart application. Use Let's Go Shopping (NT) if your `fortecgi` executable is running on an NT machine. Use Let's Go Shopping (Unix) if your `fortecgi` executable is running on a Unix machine. After you select the appropriate link, you will see a browser window with the heading: Welcome to the iPlanet UDS Shopping Cart Application.

5. Click Shop. Select a few items from each of the book categories: Science Fiction, Crime and Mystery, and Romance. Click the Buy Now icon to select an item. The Product List panel will appear. Change the value in the quantity field. Note that you must click the Update Quantities button to see your change take effect.
6. After adding a few items to your shopping cart, click View Cart. You should see the expected items.
7. Then click Check Out. You can click either the Login Existing Account button or the Create New Account button. Fill in fictional information. No real security checking is done. Then click the Login button.
8. A page will appear thanking you for your order. Click the Please Process My Order button. You will return to the application's main browser window. Click View Cart and you will see that your shopping cart is now empty.
9. When you are finished using the application, exit the browser. Then, back in iPlanet UDS, disable each of the services, then exit the Shopping Cart Application window.

Using the SQLDemo Example

Like SoftWear, the SQLDemo example application requires installation of files on both the Web server and the iPlanet UDS server.

Prerequisites To use the SQLDemo example, you must have a database and a database resource manager defined. The database can be an Oracle or Sybase database, or an ODBC data source. For more information about database resource managers, see the iPlanet UDS manual *Accessing Databases*.

► To install and run SQLDemo

1. Find the SQLDemo files on your Web server.

A WebEnterprise installation installs the necessary files on your Web server and results in a path like the following on your Web Server:

```
netscape/users/forte/examples/sqldemo
```

This directory has one file: `sqldemo.htm`. This file is the entry point page for users of the SQLDemo program.

2. Find the SQLDemo files on your iPlanet UDS server.

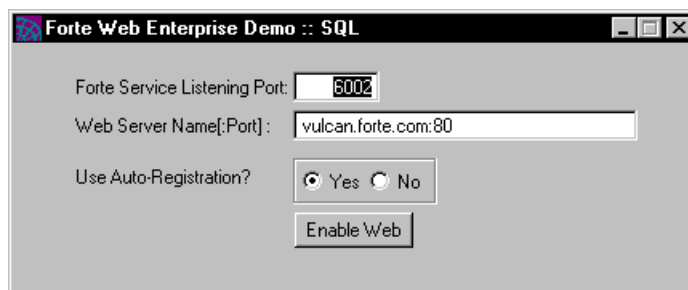
A WebEnterprise installation installs the necessary files on your iPlanet UDS server and results in a path like the following on your iPlanet UDS server:

FORTE_ROOT/install/examples/web/sqldemo

This directory contains one .pex file, `sqldemo.pex`, which contains the classes and methods for this example.

3. Import the `sqldemo.pex` file to create the project named WWWSQLDemo.
 - a. Open a workspace that contains the Web projects HTTP, HTML, HTMLWindow, and HTMLSQL in the Repository Workshop. These projects are required by the SQLDemo example and were imported during the WebEnterprise installation.
 - b. Use the Import... command under the Plan menu option to import the file `sqldemo.pex`.
4. Open the Service Object Properties dialog and modify the service object called MyMgr to use the name of your existing database resource manager.
5. (Complete this step only if you are running your Web server on NT.) Open the MainControl window class. Open the Display method, and find `cgi-forte/fortecgi`. Change `fortecgi` to `fortecgi.exe`. Recompile the method and save.
6. In the Repository Workshop, open the WWWSQLDemo project. Select the Test Run command from the Run menu, or click the Run icon on the toolbar.

You will see the MainControl Window, as shown below.



7. In the MainControl Window enter information specific to your site.
 - o iPlanet UDS Service Listening Port: The port number 6002 appears. If this is not the port number you want to use for the Catalog Access service object, enter the desired port number here.
 - o Web Server Name [:Port]: The server name and port number vulcan.forte.com:80 appear. You must update the server name to enter the complete domain address of your server, or you can enter an IP address. You need only enter a port number if you want to use a different port than 80.
 - o Use Auto-Registration?: Auto-registration is the default. Click on No if you prefer to use manual registration. If you use manual registration, you must edit the `fortecgi.dat` file, as described in [“Manual Registration” on page 202](#).

8. Click the Enable Web button.

The text of the button changes to EnablingAccess, Please Wait, as access is enabled. Then the text changes to Disable Access when the application is running and Web access is enabled.

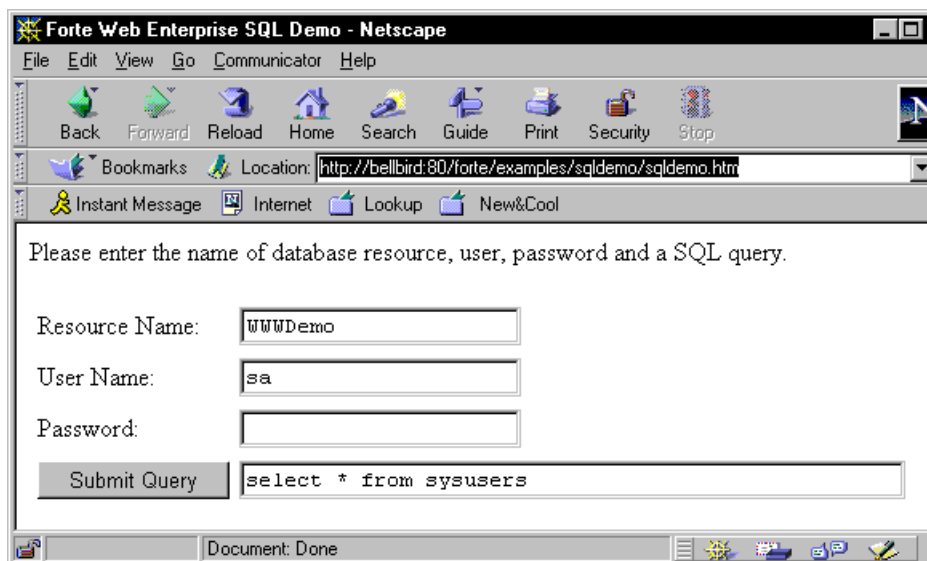
9. To use the application as a Web client, use your Web browser to enter a URL. If your Web server is running on Unix, enter it in the format:

`http://<server_name>/forte/examples/sqldemo/sqldemo.htm`

If your Web server is running on NT, enter it in the format:

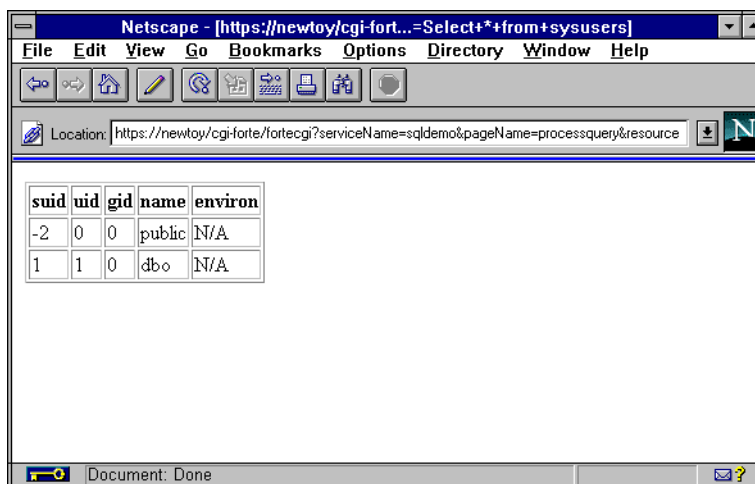
`http://<server_name>/forte/examples/sqldemo/sqldemon.htm`

The expected response is a page, similar to that shown below, that prompts you for the database resource name, your database user name and password, and a query.



10. Enter the information and any query.

The Resource Name must correspond to an existing database to which you have access. Your query may be any length. If there is an error in the query you will receive the User Error icon, along with the appropriate database error, such as inadequate access privileges or unknown table, and the appropriate iPlanet UDS error messages. An example of the response Web page is shown below.



Environment Variables

This appendix lists the environment variables provided for iPlanet UDS Web applications.

Environment Variables

The Web administrator can set the following iPlanet UDS environment variables.

FORTE_CGI_REG_PORT

Specifies the port number for the fortectgi program on the Web server. This variable is used only for autoregistration; the Web access service object contacts the fortectgi program at this port to autoregister. (Do not confuse this port number with the one used by the Web access service object; see the iPlanet UDS online Help.)

You are advised to limit access to this port. The port number should be accessible only by nodes on the intranet, limiting access to only those users considered “internal.”

The default port number used for autoregistration is port 1783.

If you change the port number, you should verify with your system administrator that the port is available and that it is only accessible to internal users.

CAUTION If you change the port number from the default, you must set this environment variable in two places: on the Web server and on the iPlanet UDS server on which the Web access service object resides. On the iPlanet UDS server side, you must set the value for FORTE_CGI_REG_PORT before starting the iPlanet UDS executable.

Syntax

FORTE_CGI_REG_PORT *port_number*

FORTE_CGI_REG_FILE

Sets both the path and file name for the `fortecgi.dat` file. If this variable is not set, the default is to write a file called `fortecgi.dat` to the same directory as the `fortecgi` program. You may use this variable to indicate a different path and/or file name. Setting this variable has no effect on the `fortecgi` program itself, other than to tell it where to find the file.

CAUTION If you set `FORTE_CGI_REG_FILE`, you must start the Web server after changing or setting the value, so that both the Web server and the `fortecgi` program read the correct value.

The type of access required by the file depends on whether you are using autoregistration or not.

- If you are using autoregistration, the file must be writable so that it can be updated whenever a service object registers or de-registers.
- If you are using manual registration, the file need only be readable by `fortecgi`; however, the file must be writable for you to edit it.

Some Web servers are configured to prevent writing to the `cgi-bin` directory. In these cases, the `fortecgi.dat` file must be written to a different directory, and the environment variable `FORTE_CGI_REG_FILE` must be set to that path and file name.

Syntax

FORTE_CGI_REG_FILE [*directory_specification*][*file_specification*]

FORTE_WW_DOCUMENT_ROOT

The FORTE_WW_DOCUMENT_ROOT environment variable specifies the root directory in which the scanner service will search for all subdirectories and HTML templates. This value can also be specified using the DocumentRoot attribute of HTMLScanner, or by a command line argument.

FORTE_WW_HANDLER_CONFIG_FILE

The FORTE_WW_HANDLER_CONFIG_FILE environment variable specifies the name and location (relative to the value of FORTE_WW_DOCUMENT_ROOT) of the scanner service object's handler file. This value can also be specified using the HandlerConfigFile attribute of HTMLScanner, or by a command line argument.

SYMBOLS

- \$\$FORTE.ExecURL variable
 - using BuildURL method instead 175
 - using for session management 183
 - using in links 183
 - when set 184
 - with session ID 184

A

- Add method
 - HTDd class
 - example 137
 - HTDI class
 - example 137
 - HTDt class
 - example 137
 - HTElement class
 - embedding text, Web page, in 140
 - HTHml class
 - Web page, appending HTML elements 138
- Administration window 212
 - enabling Web access in 67
 - setting encrypt key 171
 - SoftWear application, Main Window 67
- Anchor URL property 149
- AssignBinaryResponse method
 - returning binary data 91

- Assigned partition
 - Compiled property 194
 - moving 192
 - replication count 193
- AssignFormat method
 - HTML formats, assigning to window text 147
- AssignImage method 148
- AssignResponse method 144
- Author 76
- Authorization, Web server 218
- Autoregistration 201
 - default port 245
 - example 68
 - port number 201
 - requires fortectgi 65, 199
 - with NSAPI 199

B

- Basic authentication 213
- Binary data, returning
 - AssignBinaryResponse method 91
- BuildURL method
 - using 175

C

- Case sensitivity 206
- Catalog project 227
- CGI program 32
- Code generation, of partition 194
- Comments
 - placing tags in HTML 110
- Common Gateway Interface API 197
- Compiling partitions 194
- Configuration
 - modifying 191
- Configure as command 195
- ConvertToString method 138
- Cookie
 - for persistent state information 181
 - SessionID in 160
- Cookie domain name 237
- CreateSession method 158
- CurrentSession attribute 159
 - using 159

D

- Data validation 45
- Default configuration 190
- DefaultCookie attribute
 - setting 182
- Deploying a Web application 195
- Deploying an application 191
- De-registration
 - errors 222
- Deregistration, service object, Web access 222
- Dialog duration, Web access service object 65
- Display method, modifying for Web page 151
- DLL, fortensapi 206
- DLL, plug-ins, for 32, 198
- Document root directory
 - for template files 111
 - setting 210

- DocumentRoot attribute 210
 - for handler file 126
 - FORTE_WW_DOCUMENT_ROOT environment variable 247
 - setting for scanner service 70
- Domain attribute
 - for DefaultCookie 182
- Domain name 237
- DomainName
 - for default cookie 175, 182
- Dynamic Web pages 78

E

- easyhr.txt 52
- EasyWeb example 225
 - tutorial for 50
- ELSE tag
 - syntax description 130
- Embedding text, Web page, in 140
- EnableAccess method
 - and \$\$FORTE.ExecURL 88
 - using fortectgi.exe 205
 - using PluginURL parameter 204
- EnableSessionManagement method
 - in administration window 212
 - invoking 173
 - when sharing sessions 173
- Encrypt key
 - and sessionID 160
 - setting in initialization code 171
- Encryption 214
- Entry point Web page 83, 205
 - for SoftWare 229
 - for SQLDemo 240
- Environment variable
 - FORTE_CGI_REG_FILE 246
 - FORTE_CGI_REG_PORT 201, 245
 - FORTE_STACK_SIZE 133
 - FORTE_WW_DOCUMENT_ROOT 247
 - document root directory 210

Environment variable (*continued*)
 FORTE_WW_HANDLER_CONFIG_FILE 247
 tag handler file, identifying 126
 fortectgi stores URL information in 198
 WWW_PLUGIN_URL_BASE 199

Environment variables
 FORTE_WW_HANDLER_CONFIG_FILE 209

Error pages
 creating 92
 default 92

Exceptions
 HTML, mapped to icons 93

EXECUTE tag
 syntax description 128
 using input parameters 116
 writing HandleTag method for 117

Expiration
 session 158

Expires attribute (HTTPCookie class) 182

F

Failover 43, 72, 189, 191
 fatal.gif icon file 93
 FieldToElement method 140
 FieldWidget class, HTMLLink attribute 149
 File
 writing HTML to 143
 forte directory 230, 240
 Forte errors 221
 forte.gif icon file 93
 FORTE_CGI_REG_FILE variable 208, 222, 246
 FORTE_CGI_REG_PORT variable 201, 205, 223, 245
 FORTE_STACK_SIZE environment variable 133
 FORTE_WW_DOCUMENT_ROOT environment
 variable 208, 210, 247
 graphics in 211
 setting for ShopCart 236
 FORTE_WW_HANDLER_CONFIG_FILE
 environment variable 247

fortectgi
 connection with Web access service 198
 location of 205
 NSAPI as alternative 198
 overview 33
 process overhead 198
 using .exe extension 205

fortectgi.dat file 206
 FORTE_CGI_REG_FILE 246
 NSAPI plug-in cache of 203
 permission on 208
 URL, Web access service name 86
 Web access service object, creating 64

fortensapi DLL 206
 ForteWebEnterpriseVersion constant 224

G

GenerateSessionID method 160
 GetSessionData method 159, 162
 Graphics
 adding to a template 109
 adding to Web page 90
 on Forte server 211
 on Web server 211

H

HandleCondition method 104
 defining 123
 Handler file
 description 209
 example 125
 format of 125
 HandlerConfigFile attribute 209
 sharing 126
 HandlerConfigFile attribute 209
 FORTE_WW_HANDLER_CONFIG_FILE
 environment variable 247
 setting 126
 setting for scanner service 70

- HandleRequest method [143](#)
 - generic sample [145](#)
 - overriding [144](#)
- HandleTag method
 - defining [117](#)
 - example [119](#)
- HandleTemplate method
 - invoked by HandleTemplateRequest method [111](#)
- HandleTemplateRequest method
 - example [112](#)
 - overriding [111](#)
- HasEOL attribute
 - using [143](#)
- HasReturnAfterStart attribute
 - using [143](#)
- .htm suffix [182](#)
- HTML (Hypertext Markup Language)
 - defined [32](#)
- HTML editors [98](#)
- HTML Options... command [149](#)
- HTML projects [79](#)
 - when to use as suppliers [61](#)
- HTML templates
 - FORTE_WW_DOCUMENT_ROOT environment variable [247](#)
- HTMLImage attribute [150](#)
- HTMLLink attribute [149](#)
- HTMLScanner Class
 - DocumentRoot attribute
 - FORTE_WW_DOCUMENT_ROOT environment variable [247](#)
- HTTP (Hypertext Transfer Protocol)
 - defined [32](#)
- HTTPAccess class
 - overview [36](#)
- HTTPMessage class
 - CurrentSession attribute [159](#)
- HTTPS protocol [68](#), [233](#)
 - using [175](#)
- HTTPSession class
 - subclassing [180](#)

I

- Icon files [93](#)
- IF tag
 - syntax description [130](#)
 - writing HandleCondition method for [123](#)
- image [150](#)
- Image icon
 - appearing instead of a graphic [211](#)
- Image Source URL property [150](#)
- Images
 - broken picture image [150](#)
 - ImageSourceURL property [150](#)
- INCLUDE tag
 - syntax description [133](#)
- Init method [67](#)
- Initialization code
 - DefaultCookie attribute [182](#)
 - invoking EnableAccess [173](#)
 - RegisterTagHandler method [124](#)
 - setting encrypt key [171](#)
 - setting SessionCreationURL attribute [172](#)
 - setting timeout interval [171](#)
- Initialization tasks
 - list of [212](#)
- Invalid Request exception [177](#)
- IP address [223](#)
 - duplicate [223](#)
- ITERATE tag
 - instructional example [120](#)
 - iterator name [121](#)
 - syntax description [131](#)
 - writing HandleTag method for [117](#)

L

- Links
 - and session management [182](#)
 - missing or broken [184](#)
 - using \$\$FORTE.ExecURL variable [87](#), [183](#)
- Load balancing [43](#), [72](#), [189](#), [191](#)
- Logical partition, creating [191](#)

M

- Make Distribution command 195
- Manual registration 202
 - and basic authentication 214
 - example 68
 - using fortectgi.dat 207
 - with plug-in 199
- Master-detail report, tags for 110

N

- Name-value pairs, using 116
- New Logical Partition command 191
- Not Found message 218
- NSAPI plug in
 - fortensapi DLL 206
 - WWW_PLUGIN_URL_BASE 238
- NSAPI plug-in 197
 - defined 32
 - enabling with PluginURL parameter 204
 - fortectgi as alternative 198
 - overview 33
 - platform support 198

O

- Open method 148

P

- Page
 - and Forte window 141
 - default error 92
 - defined 32
 - guidelines for building 138
 - identifying requested 138, 144
 - static and dynamic 78
 - template 138
 - using HTML to create 139

- Page builder method 136
- Page builder service object 70
 - about 135
 - creating 72
 - invoked by HandleRequest 143
 - overview 38
 - partitioning 189
 - replicating 43, 70
- Page factory service object
 - overview 38
 - replicating 43
- Parameters
 - Forte tags 105
- Partition
 - assigning 192
 - code generating 194
 - replicating 192
 - stack size, changing 133
- Partition Workshop
 - modifying a configuration 191
- Partitioning 187–194
 - assigning partitions 192
 - creating a logical partition 191
 - modifying service objects 191
 - moving partitions 192
 - page builder service object 189
 - replicating partitions 192
 - scanner service object 189
 - Web access service object 188
- PDF files, viewing and searching 22
- Plug-in parameter (EnableAccess method) 66
- PluginURL parameter
 - EnableAccess method 204
- Port
 - Web access server 207
 - Web server 223
- Port #1783 202, 245
- Port #443 233
- Port #80 223, 233
- Primary Document Directory 210
- Programmer, Web 76
- Projects
 - business services projects 60
 - HTTP library 59
 - suggested hierarchy 59

Projects (*continued*)

- supplier plans 60
- tag handler 114
- user-defined and predefined 60
- Web access project 59

ProtocolVersion constant 202

Push button

- assigning an anchor to 147

R

REDIRECT tag

- syntax description 134

Reference partition 42

RegisterTagHandler method

- in initialization code 124

Registration errors 222

Registration, Web access service

- autoregistration and manual 200
- manual and automatic with NSAPI 199

Release number

- for Forte server 224
- Web server 224

Replication

- page builder service object 70
- service objects 70

Request failure error 219

request.gif icon file 93

Result set

- definition 106
- name 118
- returned by tag handler 117
- scope of 106

Result set member

- definition 106
- list 131

Router partition 193

Runtime error icon 220

runtime.gif icon file 93

S

Scanner service object

- defining 69
- described 40, 69, 102
- handler file,
 - FORTE_WW_HANDLER_CONFIG_FILE 247
- overview 38
- partitioning 189
- replicating 43
- root directory,
 - FORTE_WW_DOCUMENT_ROOT 247

Secure attribute 182

Secure Sockets Layer 175, 214, 233

- port number 233
- using 214

Service object

- in Web applications 188
- modifying definitions 191
- page builder 189
- scanner 69, 189
- Web access 188

serviceName URL parameter 86

Session

- beginning of 158
- CurrentSession attribute 159
- definition 158
- ending 158
- overriding ValidateSession 167
- page to create new 172
- SessionCreationURL attribute 172
- setting timeout interval 171
- setting timestamp 161
- shared by Web access services 173
- timeout interval 161
- timing out 158

Session ID

- creation 159
- customizing generation 160
- default use of cookies 184
- embedded in URL 184
- encrypted form 160
- in \$\$FORTE.ExecURL 184
- in URL 160
- passed as cookie 184
- unencrypted form 160

- Session management
 - definition 155
 - shared by multiple web access services 173
 - static pages 83
 - with static pages 82
- Session manager
 - creating non-default 175
 - creation of 154
 - shared by Web access services 173
- Session properties
 - defined 157
 - deleting 178
 - precedence 176
 - SESSION_AUTOCREATE 157
 - SESSION_REQUIRED 157
 - SESSION_UNSPECIFIED 157
 - setting 176
- session properties
 - page builder pages 178
- Session property file
 - naming 208
 - row format 177
- Session table
 - persistent storage and 181
 - shared by Web access servers 173
- SESSION_AUTOCREATE session property 157
- SESSION_REQUIRED session property 157
- SESSION_UNSPECIFIED session property 157
- SessionCreationURL attribute
 - setting 172
- SessionMgr class
 - creation of session manager 154
 - sharing between services 157, 176
- SetDefaultSessionProperty method
 - using 176
- SetEncryptKey method 171
- SetSessionData method 159, 162
- SetSessionProperty method
 - using 176, 178
- SetSessionPropertyConfigFile method 208
- Shared windows
 - definition 62
- SharedWindows project 227
 - defining 73
- ShareSessions method
 - using 173, 176
- ShopperID variable 89
- Socket errors 220
- SoftWear example
 - data files 230
 - how to run 230
 - Make Database 230
 - NSAPI in 199
- SourceWindow attribute 146
- SP_ER_ERROR 93
- SP_ER_FATAL 93
- SP_ER_INFORMATION 93
- SP_ER_USER 93
- SP_ER_WARNING 93
- sqldemo.htm file 240
- sqldemo.pex file 241
- SQLToTable method 142
- Stack size, changing 133
- Start method 67
- State information 47
 - defined 155
 - HTTPSession class 180
 - persistent 181, 186
 - session objects and 159
 - storing in client 47, 185
 - using cookies for persistent 181
- Static registration
 - example 124
- Static Web pages 78, 83
 - and session management 82
 - example 85
- Supplier plans
 - recommended 60

T

- tag handler file, identifying 209
- Tag handler project 60, 114

- Tag handlers 104
 - configured as library 124
 - creating 112
 - dynamically loading 209
 - handler file 209
 - returning a result set 117
 - writing code for 116
- Tag names
 - code for 116
 - one and two-part 110
- TagHandlerIface interface 104
- Tags, Forte
 - adding new to template 110
 - description of 103
 - order of in a template 110
 - parameters 105
- Templates
 - compared to HTML file 97
 - DocumentRoot attribute 210
 - embedding graphics 211
 - example of 100
 - in scanner memory 103
 - location for 210
- Timeout interval
 - session 161
 - setting 171
- Timestamp, session ID 161
- Tutorial (EasyWeb example) 50

U

- Unauthorized message 218
- Unregistering service object, Web access 222
- URL
 - defined 32
 - definition and format 86
 - pagename parameter 87
 - with sessionID embedded 184
- URLForForteCGI parameter
 - used to set \$\$FORTE.ExecURL 88

- Usage page 219
- usage.gif icon file 93
- user IDs (Web) 213
- User interface 48
- user.gif icon file 93

V

- ValidateSession method 166, 174
 - example 166
 - example of overriding 168
- Variables, Forte 106
 - \$\$FORTE.ExecURL 87

W

- warning.gif icon file 93
- Web access project 59
 - defining 64
- Web access server
 - defined 32
- Web access service object
 - dialog duration 65
 - IP address 207
 - overview 35
 - partitioning 188
 - port number 207
 - protocol version 207
 - service name 207
 - setting DefaultCookie 182
- Web application
 - compared to window-based application 44
 - data validation for 45
 - default configuration for 190
 - deploying 195
 - designing 40
 - partitioning 187
 - scaling 43
 - state information 47
 - structuring 40
 - user interface structure 48

- Web author 76
- Web browser
 - defined 33
- Web page
 - embedding text 140
- Web protocol 86
- Web server 33
 - port number 223
- Web site 33
 - steps for building 75
- web.forte name for plug-in 204
- WebEnterprise
 - release number for Forte side 224
 - release number for server side 224
- WebSessionMgr attribute 157
- Widget
 - assigning an anchor to 149
- WindowConverter class
 - examples 140
- WindowToForm method 140
- WriteToFile method 143
- WWW_CGI_URL_BASE environment variable
 - setting for ShopCart 237
- WWW_CGI_URL_BASE2 environment variable
 - setting for ShopCart 237
- WWW_COOKIE_DOMAIN environment variable
 - setting for ShopCart 237
- WWW_PLUGIN_URL_BASE environment variable 199
- www1.pex file 227
- www2.pex file 227
- www3.pex file 227
- www4.pex file 228
- www5.pex file 228
- WWWCatalogAccess project 228
- WWWCatalogPageBuilder project 227
- WWWCatalogWindows project 228
- WWWSQLDemo project 241
- WYSIWYG mode, HTML editor 109

