

Customizing WebEnterprise Designer Applications

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
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	21
Viewing and Searching PDF Files	22
Chapter 1 WebEnterprise Designer Application Architecture	25
Logical Architecture	26
Runtime Architecture	27
Web Application Server Architecture	28
Runtime Objects by Partition	29
Use of Express Services	31
WebEnterprise Designer Projects	31
HTTP Library	33
The ExpressHandlers Project	34
The html_modelHandlers Project	35
Customizable Subclasses in html_modelHandlers	37
Generated HTML Templates	37

WebEnterprise Designer Class Interactions	39
Class Interaction Diagram	39
Declared Type and Runtime Type	40
Life of a Template Request	42
Runtime Scenarios	43
Object Interaction Diagram Conventions	43
ExpressHTTPAccess	44
Starting the AccessService Service Object	44
Customizing ExpressHTTPAccess	46
ExpressScanner	47
Starting the ScannerService Service Object	47
ExpressClassHandler	48
Runtime Control Flow	48
Referenced Objects	51
Field Identification	51
Data Transfer	52
Result Sets	52
Connections Between Handlers	54
ExpressLogonHandler	54
Customizing ExpressLogonHandler	54
ExpressPageData	55
Foreign Result Sets	55
ExpressValueGenerator	55
ExpressLookupInfo	56
Modifying the Displayed Null Value	56
WebEnterprise Designer HTML Template Elements	56
Links	56
URL Parameters	57
Variables	60
Chapter 2 Customizing WebEnterprise Designer Application Classes	63
Overview	64
Before You Begin	64
Creating Customizable Classes	65
Creating a Single Customizable Page Handler Class	66
Creating a Full Set of Customizable Page Handler Classes	66
Customizing With the Page Handler Customization Wizard	67
Customizing a Page Handler Class	69
Customizing a Generated HTML Template	72
Deleting Customizations	72
Deleting Specific Customizations	73
Deleting All Customizations in a Class	74
Making Application-Wide Customizations	76

A Roadmap to Customization Examples	76
Page Handler Customization Wizard Help Files	77
Page Handler Customization Wizard Customizations	77
Customizing Manually	79
Locating Where to Customize	80
Overriding Methods in a Superclass	80
Local and Global Customizations	82
Error Reporting	82
Working with Business Classes	83
Business Class Record Status	83
BusinessClass Attribute IDs (ATTR_)	84
Changing the Value of an Attribute	84
Checking the Status of a BusinessClass Object	85
Undoing Changes Made to a BusinessClass Object	85
Customization Techniques: ClassHandler Classes	86
Creating a New Instance of a Business Class	86
Getting the Result Set	86
Getting the Initial Query	87
Customization Techniques: Business Rules	87
Where to Implement	87
Business Rules on the Browser	88
Customization Techniques: Data	88
Formatting Fields	88
Formatting Custom Fields	89
Decoding or Validating Fields	89
Processing Custom Fields on an HTML Form Submission	89
Processing an Insert or Update Form	90
Processing a Search Form	90
Global Customization	90
Chapter 3 Customizing Generated HTML Templates	93
How WebEnterprise Designer Uses HTML Templates	93
Common Templates	94
Business Class Page Templates	94
Link Page Templates	97
Logon Page Templates	97
Page Design Templates	97
Simple Page Design Templates	98
Fancy Page Design Templates	98
Fancy Page Design Variations	99

Customizing HTML Templates	100
Customization Types	100
Where to Customize	101
What Not to Customize	101
Regenerating After Customizing	102
Scenario 1: HTML Changes Only	103
Scenario 2: HTML and Model Changes	103
Scenario 3: Conflicting HTML and Model Changes	104
Customization Examples	106
Conventions Used with the Examples	106
Example: Customizing a Field on a Search Page	106
Example: Customizing a Font Size on a Data Page	109
Chapter 4 Customizing Page Designs	115
About Page Designs, Templates, and Pages	115
Page Designs and Web Page Production	116
When to Customize Page Designs	117
Page Design Elements and HTML Template Generation	117
Page Design File Names and Selectors	118
Page Design Code Generation Processing	122
Guidelines for Customizing Code Generation Directives	124
Example: Customizing a Page Design	124
Create a New Design Directory	125
Identify the Design with a Bitmap and Text	125
Clear Existing Generated HTML Templates	126
Customize the Design Files	127
Remove the Menu From the Data File	127
Modify the Menu Design File	129
Modify the Display Design File	131
Generate and Inspect	132
Fine-Tune the Customized Design	136
Chapter 5 Customizing Page Styles	137
HTML 4.0 and Style Sheets	137
WebEnterprise Designer and Style Sheets	138
Using HTML Style Elements with WebEnterprise Designer	139
Identifying the Style Sheet to Use	139
Using HTML Attributes	139
The class Attribute	139
The id Attribute	141
Using HTML Style Elements	142

Customizing Page Styles	144
Outline of Basic Procedures	145
Creating the New Style Sheet File	146
Modifying Existing Elements	146
Adding New Elements	147
Identifying the Style with a Bitmap and Text	148
Considering the Browser	150
Browser Caching	150
Browser Independence	150
Chapter 6 Customizing Error Pages	151
WebEnterprise Exception Handling	151
Default Exception Processing	152
WebEnterprise Exception Result Set Variables	154
HTMLScannerException Class Variables	154
HTTPAccessException Class Variables	155
Variables for All Other Exceptions	155
Customizing Error Pages	156
Modifying Default Error Pages	156
Creating Custom Error Pages	156
Customizing a WebEnterprise Designer HTML Application	157
The GetErrorTemplate Method	158
Example: Application-Specific Error Template	159
Example: Application-Specific Template with Custom Data	160
Customizing HTTPAccessExceptions	161
Errors in Error Customization	162
Chapter 7 Customization Examples	163
Introduction	164
Methods for Editing Generated Files	165
Using the Page Handler Customization Wizard	165
Customizing TOOL Methods Manually	165
Customizing Generated HTML and Text Files Manually	166
Example: Adding a Lookup Reference Page	166
What This Example Does	167
Creating a Lookup Link	168
Step 1. Add a Reference Page to the HTMLtutApp Model	169
Step 2. Capture the Search Page URL	170
Step 3. Create a Link with the Captured URL	171
Step 4. Pass the Selected Field Value	174

Example: Adding a Lookup Reference Page <i>(continued)</i>	
Testing Your Work Before the Final Step	175
Step 5. Remove the CustomerOrder-CustomerList Link	177
Usage Recommendations	177
Example: Passing Data with a Command Link	177
What This Example Does	178
Creating the Customization	179
Step 1. Add Insert and Update Commands (If Required)	179
Step 2. Add a Variable to Hold the Value	180
Step 3. Single Out One Instance of the Data	182
Step 4. Populate the Order Number with Incoming Data	183
Testing Your Work	184
Example: Automatically Populating Data on an Insert Page	185
What This Example Does	186
Creating the Customization	186
Step 1. Add a NewOrder Page to the HTMLtutApp Model	187
Step 2. Add a Variable to Hold the Value	189
Testing Your Work	191
Example: Adding a Drop List for Entering and Formatting Dates	191
What This Example Does	192
Creating Date-Formatting Drop Lists	193
Step 1. Define Drop Lists for Date Elements	193
Step 2. Override the BeforeInsert Method	195
Testing Your Work	196
Example: Removing a JavaScript Validation from a Page Mode	198
What This Example Does	198
Creating the Customization	199
Step 1. Apply the JavaScript to a Customer Page Field	199
Step 2. Remove the JavaScript Validation from a Template	200
Testing Your Work	202
Example: Displaying the Record Just Inserted	203
What This Example Does	203
Creating the Customization	204
Testing Your Work	205
Example: Validating a Whole Form	206
What This Example Does	206
Creating a Field Constraint with JavaScript	207
JavaScript Boilerplate	207
Step 1. Add the JavaScript Validation to the Template	208
Step 2. Add a Value Attribute to the Field Descriptions	210
Step 3. Modify the Insert Button	210
Testing Your Work	211

Example: Making a Field Mandatory	212
What This Example Does	213
Creating a Field Constraint with TOOL	214
Testing Your Work	215
Drop List or Radio List Example: Entering Lookup Information Manually	216
What This Example Does	216
Creating the Customization	217
Step 1. Add a Drop List Validation to the Field	217
Step 2. Generate the Lookup File	218
Step 3. Customize the Lookup File with Your Values	219
Testing Your Work	221
Drop List and Radio List Example: Removing <Not Selected> and <None>	222
What This Example Does	222
Creating the Customization	223
Technique 1: Customizing the Page Mode Template	223
Technique 2: Customizing the Scripts File	225
Testing Your Work	227
Chapter 8 Customizing Application Security	229
Security and HTML Applications	229
Authenticating Users	231
Creating the Logon Page	231
Code Generated for the Logon Page	232
How the Logon Page is Activated	233
Integrating the Application with an Authentication System	234
Example: Adding LogonSession Code	235
Restricting Access to Application Pages	237
Sharing a Security Environment Across HTML Applications	239
Avoiding Security Leaks	239
Customizing Subsidiary HTML Models to Share Security	240
Customizing the Main HTML Model to Share Security	240
Restrictions	241
Summary	242
Session Timeout	242
How Session Timeout Works	242
Finding the Ideal Setting for Session Timeout	243
Example: Customizing Session Timeout	244
Other Security Customizations	244

Chapter 9 Partitioning and Deploying a WebEnterprise Designer Application	245
About Partitioning a	
WebEnterprise Designer Application	245
About HTML Application Projects and Service Objects	246
business_modelServices Service Objects	248
html_modelHandlers Service Objects	249
Relationship Between the Service Objects	250
Creating a Default Partitioning Configuration	252
Modifying the Configuration	254
Testing the Application in a Distributed Environment	254
Deploying the Application	256
Making the Application's Template Files Accessible to the ScannerService SO	258
Copying the Application's Template Files to the Scanner Partition	258
Setting Document Root on the ScannerService Service Object Partition	259
Running the Application	260
The AccessService Log File	263
Memory Considerations	264
Index	265

List of Figures

Figure 1-1	Three-Tier Architecture	26
Figure 1-2	Anatomy of a WebEnterprise Designer Application at Run-Time	27
Figure 1-3	HTMLtutApp Partitions and Objects at Runtime	30
Figure 1-4	Code Generated from a WebEnterprise Designer HTML Application	32
Figure 1-5	HTTP Class Hierarchy	33
Figure 1-6	ExpressHandlers Class Hierarchy and Suppliers	34
Figure 1-7	Example Class Diagram	40
Figure 1-8	WebEnterprise Designer Class Interaction Diagram	41
Figure 1-9	Life of a Template Request	42
Figure 1-10	Example Object Interaction Diagram	44
Figure 1-11	Object Interaction Diagram of the Startup of the AccessService Service Object	45
Figure 1-12	Object Interaction Diagram of Startup of the ScannerService Service Object	48
Figure 1-13	Object Interaction Diagram of a Template Request	49
Figure 2-1	Naming Conventions Before and After Creating Customizable Classes	65
Figure 2-2	Page Handler Customization Wizard Opening the Appropriate Method Workshop	68
Figure 2-3	Page Handler Customization Wizard's Help on a Customization Topic	69
Figure 2-4	Page Handler Customization Wizard Help	77
Figure 4-1	Page Design Identifiers	126
Figure 4-2	Menu in simple and fancyMenu Designs	128
Figure 5-1	Style Element Examples	144
Figure 5-2	Style Identifiers	149
Figure 6-1	Default Scanner Exception Page	153
Figure 7-1	Lookup Link on Customer Order Insert Page	167
Figure 7-2	Customer List Page	167
Figure 7-3	Customer Number Returned with the Insert Page	168
Figure 7-4	HTMLtutApp Model Modified for a Lookup Link	177
Figure 7-5	Selected Customer on Master-Detail Page	178

Figure 7-6	LineItem Insert Page Showing Passed Value	178
Figure 7-7	Insert Mode of NewOrder Page	186
Figure 7-8	HTMLtutApp Model with Additional CustomerOrder Page	188
Figure 7-9	Requested Date Field Formatted with Drop Lists	192
Figure 7-10	Inserted Customer Order Showing the Date	192
Figure 7-11	IsAlphabetic Validation Error	199
Figure 7-12	Validation Working on Insert Page	202
Figure 7-13	Entering a New Customer Record	203
Figure 7-14	New Record Displayed	204
Figure 7-15	Restricted Fields on Insert Page	206
Figure 7-16	Restricted Field Error Message	207
Figure 7-17	Restricted Field on Insert Page	213
Figure 7-18	Restricted Field Error Page	213
Figure 7-19	Customized Drop List	216
Figure 7-20	Customized Drop List	223
Figure 9-1	Relationship between AccessService, ScannerService, Service, and DBService	251
Figure 9-2	Default Configuration for HTMLtutApp	253
Figure 9-3	Make Distribution Dialog	257
Figure 9-4	Obtaining URL Elements from the Properties Dialog	262

List of Procedures

- To copy the documentation to a client or server 22
- To view and search the documentation 22
- To create customizable subclasses for every business class page in your model 66
- To customize a page handler class using the Page Handler Customization Wizard 70
- To delete a customization 73
- To delete all customizations (entire class) 74
- To override a method 81
- To customize ExpressHandlers classes 91
- The general steps for creating a customized page design are 124
- To create and populate a new design directory 125
- To identify a page design 126
- To clear generated HTML templates 127
- To remove the menu block from the data file 128
- To create and modify the Menu design file 129
- To modify the Display file to create the required display and call the Menu file 131
- To verify your work by generating code 132
- To fix the error in the Menu file 134
- To create a new style sheet file 146
- To customize only existing elements in the new style sheet 146
- To customize a restricted style 147
- To customize generally available styles 148
- To identify the marketing style 149
- To customize an application's error page using the Customization Wizard 157
- To customize a method of an ExpressHandlers class in a HTMLtutAppHandlers class 166
- To set up the HTMLtutApp model for a lookup link 169
- To capture the URL of the Search page mode 170
- To customize the CustomerOrder page 172

To customize the Data template of the Customer List page	174
To test the work you have done so far	176
To finish the customization	177
To add the Insert and Update commands to a page (if necessary)	179
To add a variable to the Insert command link of the LineItem page	181
To add a forte iterate loop	182
To customize the Insert mode of the LineItem page to display the data	183
To test the customization	184
To modify the HTMLtutApp model	187
To populate the Customer Number value in the NewOrder Insert page	190
To test the customization	191
To define drop lists for the Requested Date field	193
To override the BeforeInsert method for this customization	195
To test the customization	196
To set a validation on the Name field of the Customer page	199
To remove the validation from the search template	200
To test the customization of the JavaScript validation	202
To code this customization with the Customization Wizard	204
To test the DoInsert method customization	205
To add a JavaScript validation script to restrict non-null field entry	209
To add a value attribute to the validated fields	210
To modify the Insert button	211
To test the customization of the JavaScript validation	211
To create a field constraint with TOOL	214
To test the TOOL customization	215
To add a drop list validation to the gifAddress field	217
To generate a lookup file	218
To customize the lookup file	219
To test your manually customized lookup file	221
To remove the <Not Selected> option from a drop list	224
To remove the <Not Selected> option from a drop list	226
To test your drop list customization	227
To create a Logon page	231
To activate the logon page	233
To customize an application's logon page using the Customization Wizard	235
To restrict access to a page	237
To customize subsidiary models to share the security environment	240

To customize the main model to share the security environment	240
To change the session timeout	244
To partition an HTML application	253
To test run an HTML application in distributed mode from the Partition Workshop	255
To test run your application in distributed mode from other workshops	255
To deploy your HTML application	256
To run a deployed HTML application	260

Preface

This manual describes the architecture of applications created by WebEnterprise Designer. The architecture includes both a hierarchy of TOOL classes and generated HTML templates. Once you are familiar with the WebEnterprise Designer architecture, you can begin to customize your applications.

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 21
- “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 for application developers. We assume that you:

- have programming experience
- are familiar with SQL and your particular database management system
- are familiar with HTML
- understand the basic concepts of object-oriented programming as described in *A Guide to the iPlanet UDS Workshops*
- have used the iPlanet UDS Repository Workshop
- have a basic understanding of WebEnterprise Designer as described in *Getting Started with WebEnterprise Designer*
- understand the basic concepts of WebEnterprise as described in *A Guide to WebEnterprise*

Organization of This Guide

The following table briefly describes the contents of each chapter:

Chapter	Description
Chapter 1, “WebEnterprise Designer Application Architecture”	Provides an overview of the architecture of both supplied and generated WebEnterprise Designer classes and how they interact.
Chapter 2, “Customizing WebEnterprise Designer Application Classes”	Describes how to use the Page Handler Customization Wizard, and describes each of the customizations you can implement with it.
Chapter 3, “Customizing Generated HTML Templates”	Provides an overview of generated HTML template files and how they interact. Also provides guidelines and examples of customizing them.
Chapter 4, “Customizing Page Designs”	Provides guidelines for customizing the provided page designs or creating your own.
Chapter 5, “Customizing Page Styles”	Provides guidelines for customizing the provided page styles or creating your own.
Chapter 6, “Customizing Error Pages”	Provides guidelines for customizing the error pages that display in the browser.

Chapter	Description
Chapter 7, “Customization Examples”	Describes 12 practical customizations for WebEnterprise Designer applications.
Chapter 8, “Customizing Application Security”	Provides information on creating a secure WebEnterprise Designer application, creating a logon page, and integrating the application with your own authentication system.
Chapter 9, “Partitioning and Deploying a WebEnterprise Designer Application”	Describes how to partition and deploy a WebEnterprise Designer application in a complex environment.

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.
monospace	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 (FORTE_ROOT) or acronyms (UDS, JSP, iMQ). Uppercase text can also represent a constant. Type uppercase text exactly as shown.
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.

WebEnterprise Designer Application Architecture

This chapter discusses the architecture of a generated WebEnterprise Designer application by examining the supplied and generated classes and illustrating how they interact.

Topics covered in this chapter include:

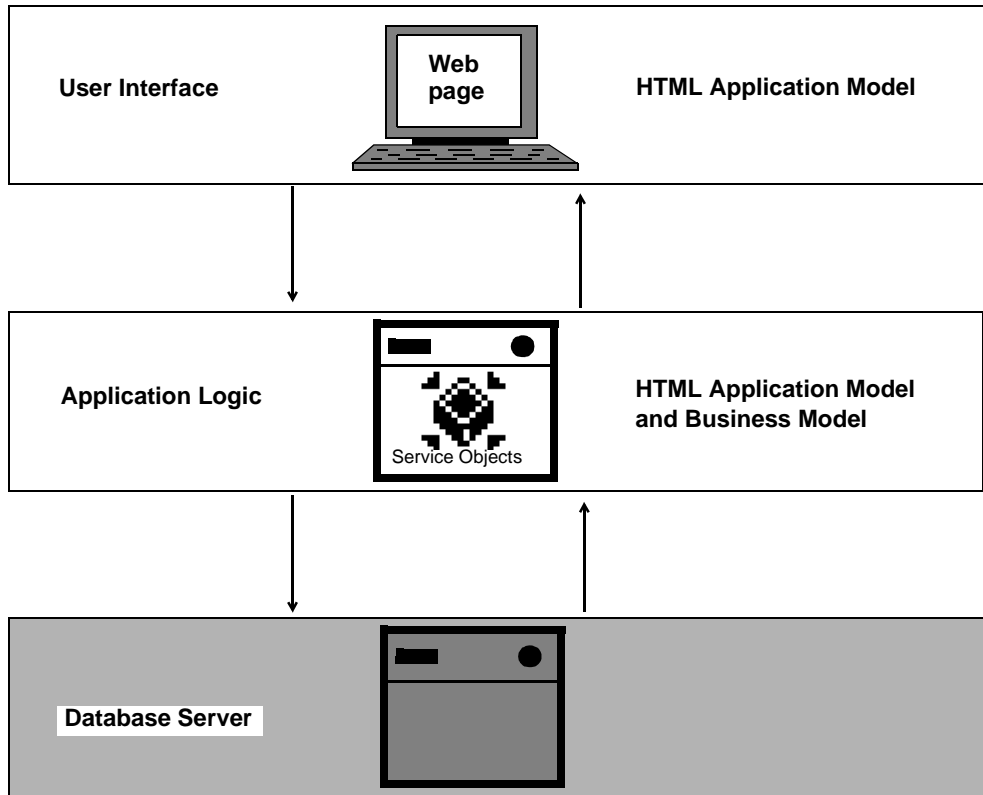
- a description of the runtime interaction between the Web application services and the business services
- a description of the generated WebEnterprise Designer project and its suppliers
- a diagram describing the relationship between generated classes and supplied classes
- the flow of control between methods and objects at runtime
- the flow of control between pages at runtime
- the effect of properties in the HTML Application Model Workshop on generated classes

Throughout this chapter, examples of general topics are drawn from the HTML Tutorial application. For example, you will see references to `CustomerOrderHandler` and `Display_CustomerOrder.html`—these elements exist in the generated HTMLtutAppHandlers project and HTMLtutApp HTML document directory.

Logical Architecture

WebEnterprise Designer supports a three-tier architecture. In a three-tier architecture, the user interface, application services, and database processing are divided into separate modules, as illustrated in **Figure 1-1**:

Figure 1-1 Three-Tier Architecture

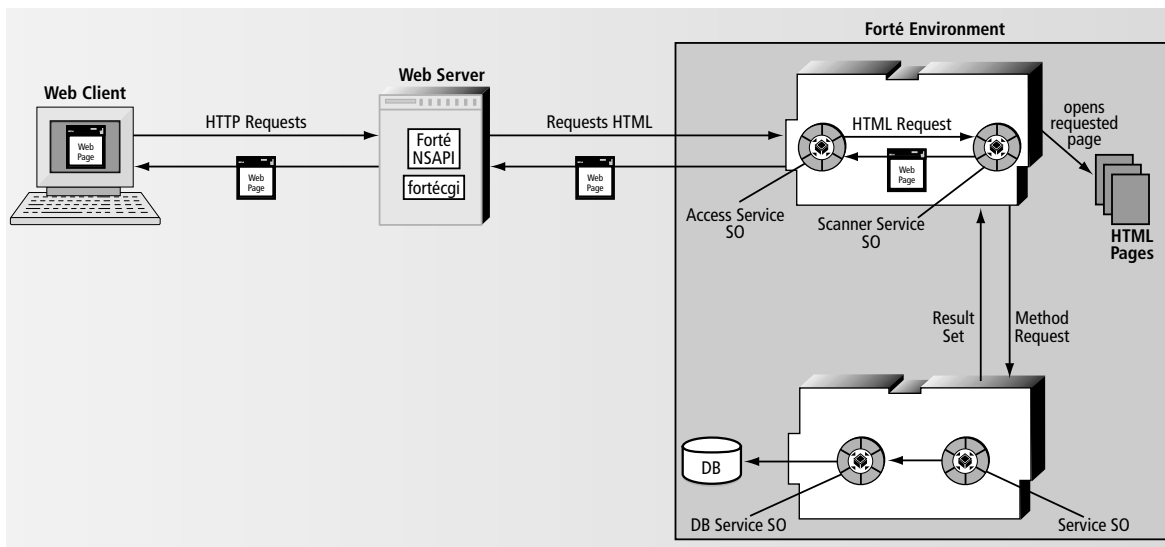


Using WebEnterprise Designer, you build the first and second tiers of your application. With the HTML Application Model Workshop, you create the user interface and Web application server; with the Business Model Workshop, you create the application business services, which provide access to database information.

Runtime Architecture

Figure 1-2 illustrates the interactions of the parts of a WebEnterprise Designer application at runtime.

Figure 1-2 Anatomy of a WebEnterprise Designer Application at Run-Time



In Figure 1-2, the Web Client at the left represents a client machine using a Web browser to view a WebEnterprise Designer application. When the user requests new information, the browser passes an HTTP request (a URL) to the Web server, requesting a Web page.

On the Web server, the fortécgi program or an iPlanet UDS Web server plug-in directs the HTTP request to the WebEnterprise Designer AccessService service object, which passes the page request to the ScannerService service object, along with any parameter values relevant to the page.

The ScannerService service object opens the requested HTML page template and processes the iPlanet UDS tags to create the page. When the service object encounters certain tags in the HTML template, it invokes the HandleTag and HandleCondition methods of the handler class associated with the template. The handler class uses the Business Service service object to access business services (which access the database), and returns any data needed in a result set. The handler class caches the result set in the AccessService for future use.

After the ScannerService service object populates the page template with the result set, it returns the requested page back to the AccessService service object, which delivers it to the Web server. The Web server in turn passes the page to the client Web browser, which displays the page.

The AccessService service object maintains state for all the Web clients it serves. Its dialog duration is session. Unlike normal session duration service objects, you may load balance the AccessService service object. When you do, each new browser request is routed to the next available replicate. Subsequent requests from the same browser session will be directed back to the replicate that handled the first request.

Web Application Server Architecture

An application generated by WebEnterprise Designer is comprised of a set of logical pages and associations between those pages. These logical pages are implemented by a combination of HTML templates and TOOL code. A TOOL class and a set of HTML templates are generated for each logical page.

The TOOL class generated for each logical page is the *bus_class_pageHandler* class, referred to as the “handler” or “page handler,” and is generated into the *html_modelHandlers* project. It is a subclass of the ExpressClassHandler class from the ExpressHandlers project. The handler implements the logic intended by the page, and formats any required data into a WebEnterprise result set for substitution into the HTML page built by the WebEnterprise scanner.

In particular, the handler page implements the WebEnterprise TagHandlerIFace interface and handles WebEnterprise tag requests made by the HTML templates generated for the logical page. The *html_modelScanner* class that is generated for the application instantiates each of the handler class objects defined by the application and, when required, dispatches to them tag requests made by their associated HTML templates.

HTML templates The set of HTML templates for each logical page defines how the data for the page will be displayed, and implements the links to other logical pages.

HTML templates are HTML files with embedded WebEnterprise tags. These tags cause:

- invocations of either the HandleTag or HandleCondition methods of the handler class associated with the template
- substitution of data generated by the handler class into the HTML page sent in response to the request for the WebEnterprise HTML template

In addition, WebEnterprise Designer generates what is required—principally, the Access and Scanner services—to invoke the appropriate handler class method when requests for HTML templates are made. The Web application server designer usually does not need to be concerned with the required mechanisms, but they are essentially described in the object interaction diagram of the template request process (see section [“Runtime Control Flow”](#) on page 48).

Customization guidelines Customizing a WebEnterprise Designer application involves modifying the generated HTML templates, the generated handler classes, or both. In general, you customize:

- the handler class when you want to affect the logic of the page
An example is when you want to modify the data returned by a search request.
- the HTML template when you want to affect the presentation of data or the connection between pages
Examples are when you want to display some fields in bold font, or move the location of a field on the page.

Often customization involves changes to both an HTML template and a handler class. For example, adding a custom field requires adding a tag reference for the field in the HTML template to display the data, and adding the logic to the associated handler class to compute the value and place it in the result set for the template.

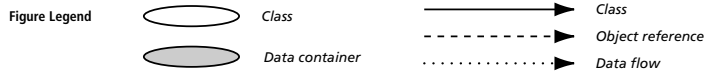
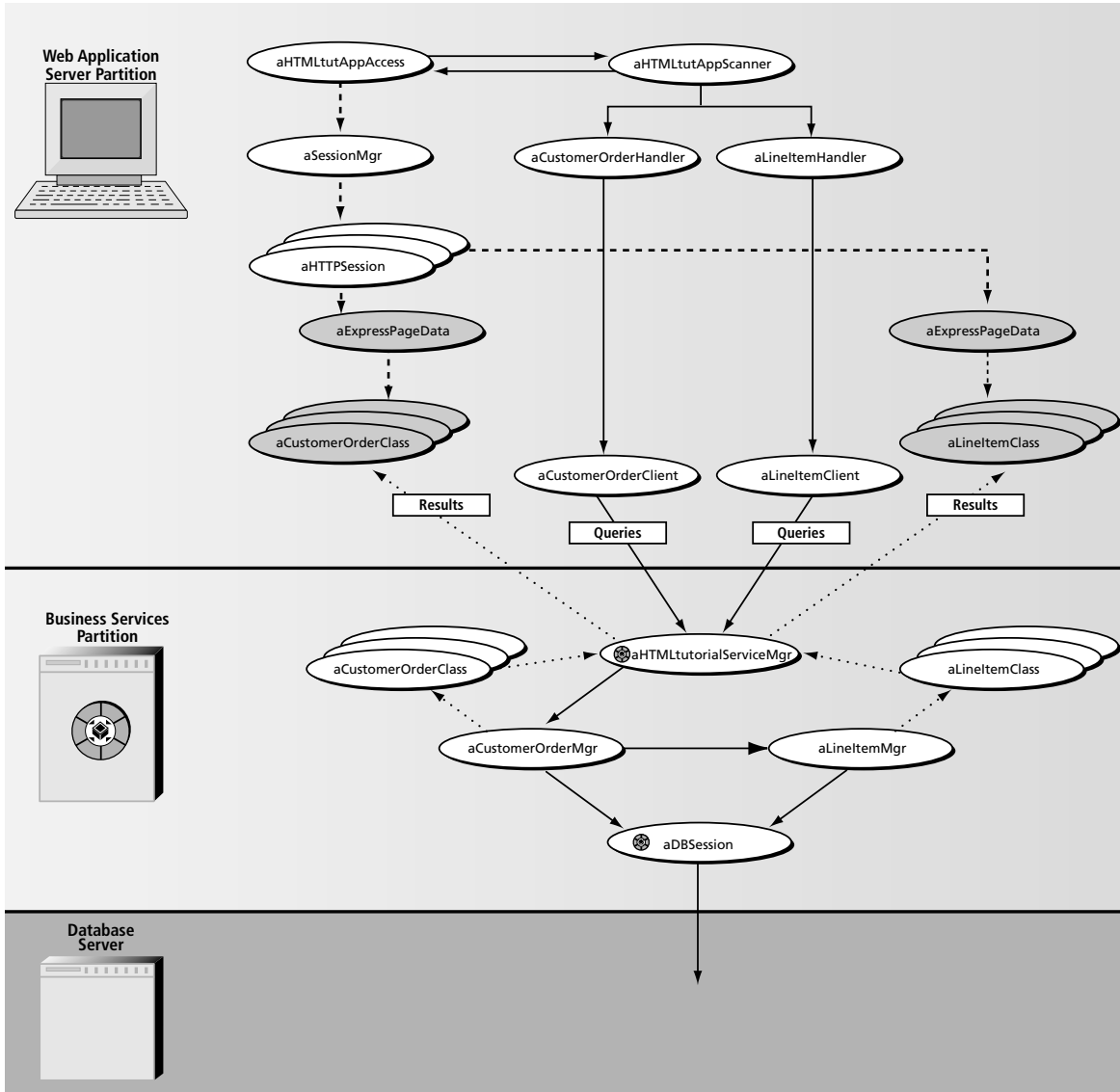
Examples of all types of customizations are provided in [Chapter 7, “Customization Examples.”](#)

Runtime Objects by Partition

[Figure 1-3](#) illustrates the two partitions and their key objects that typically exist at runtime in the deployed HTML Tutorial application.

NOTE You can customize every element in this diagram, except `aDBSession`, by modifying its class definition or HTML template. See [Chapter 2, “Customizing WebEnterprise Designer Application Classes,”](#) for details.

Figure 1-3 HTMLtutApp Partitions and Objects at Runtime



Multiple ovals means there may be many such objects (for example, an array of). The prefix “a” means an instance of the class with that name (for example, aLineItemHandler). Control flow arrows mean “invokes methods on.” The object at the beginning of an object reference arrow references the object at the arrow end.

Use of Express Services

The classes generated by the HTML application model are used in the Web application server partition. In addition, some of the classes generated by the business model (the *business_classClass*, *business_classQuery*, and *business_modelClient* classes) are also used in the Web application server tier. The Web application server is a client of the business services generated by the business model.

For a description of Express Services classes, see *Customizing Express Applications*.

WebEnterprise Designer Projects

This section describes WebEnterprise Designer projects, including the project generated from the HTML application, the libraries and projects that supply the generated project, and a brief overview of the HTML templates that are also generated for the application (these are described fully in [Chapter 3, “Customizing Generated HTML Templates”](#)).

[Figure 1-4](#) gives you an idea of what all this code is generated for:

Figure 1-4 Code Generated from a WebEnterprise Designer HTML Application

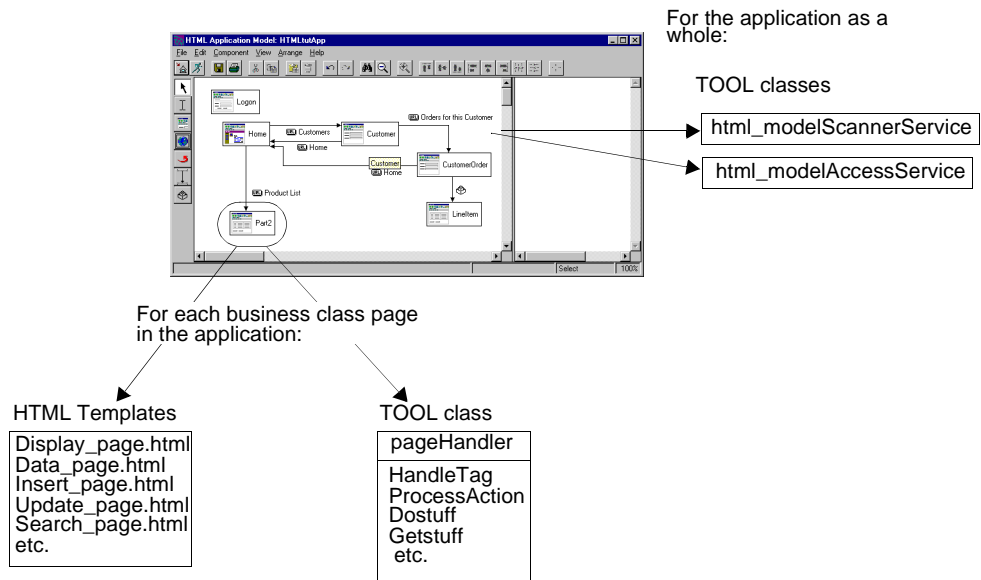


Figure 1-4 shows that, for each business class page, WebEnterprise Designer generates a set of HTML templates and a Handler TOOL class. In addition, it generates a Scanner service object and an Access service object (and corresponding classes that define each service object) for the whole application.

All of the TOOL code for the HTML application is generated into the *html_modelHandlers* project. The supplier projects of the *html_modelHandlers* project include:

- the HTTP library (a WebEnterprise project)
- the ExpressHandlers project (a WebEnterprise Designer project)
- the business models the HTML application uses and their suppliers, which include the ExpressDomains and ExpressServices projects

This section provides class hierarchy diagrams of the HTTP library and the ExpressHandlers project, followed by a detailed description of the *html_modelHandlers* project. Complete descriptions of the HTTP library and the ExpressHandlers project are found in iPlanet UDS online Help. Descriptions of ExpressDomains and ExpressServices are found in *Customizing Express Applications*.

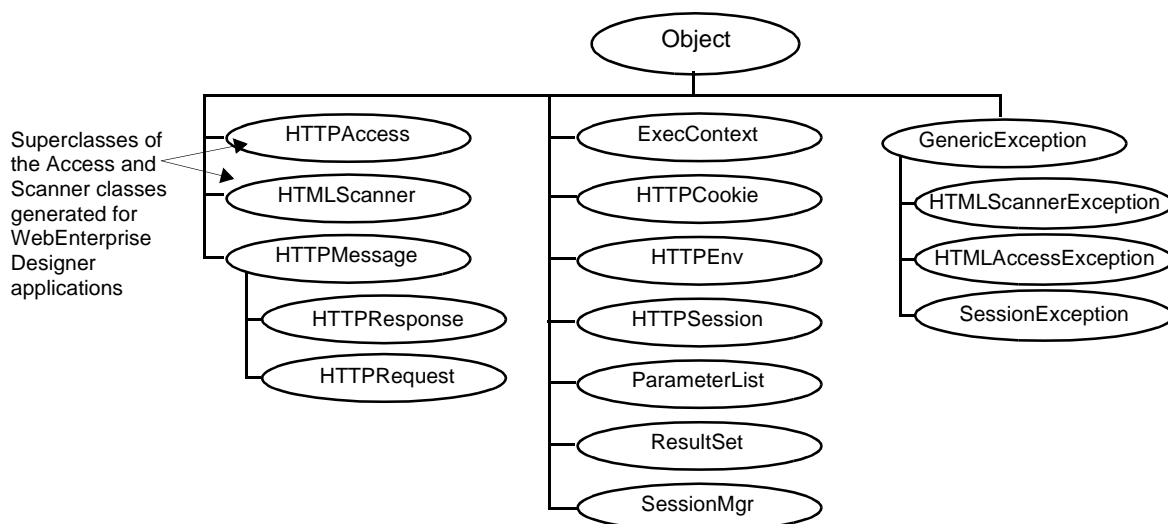
HTTP Library

The HTTP library provides classes that are used to:

- define service objects for an iPlanet UDS Web application
- manipulate HTTP requests and responses
- process HTML templates to generate Web pages
- Create and track unique client sessions

Figure 1-5 shows the class hierarchy for the HTTP library.

Figure 1-5 HTTP Class Hierarchy



In addition to the classes shown in Figure 1-5, the HTTP Library contains the TagHandlerIFace interface. This interface must be implemented in order for WebEnterprise to process the iPlanet UDS tags in the HTML templates. The ExpressHandler class (in the ExpressHandlers project, described in the next section) implements this interface.

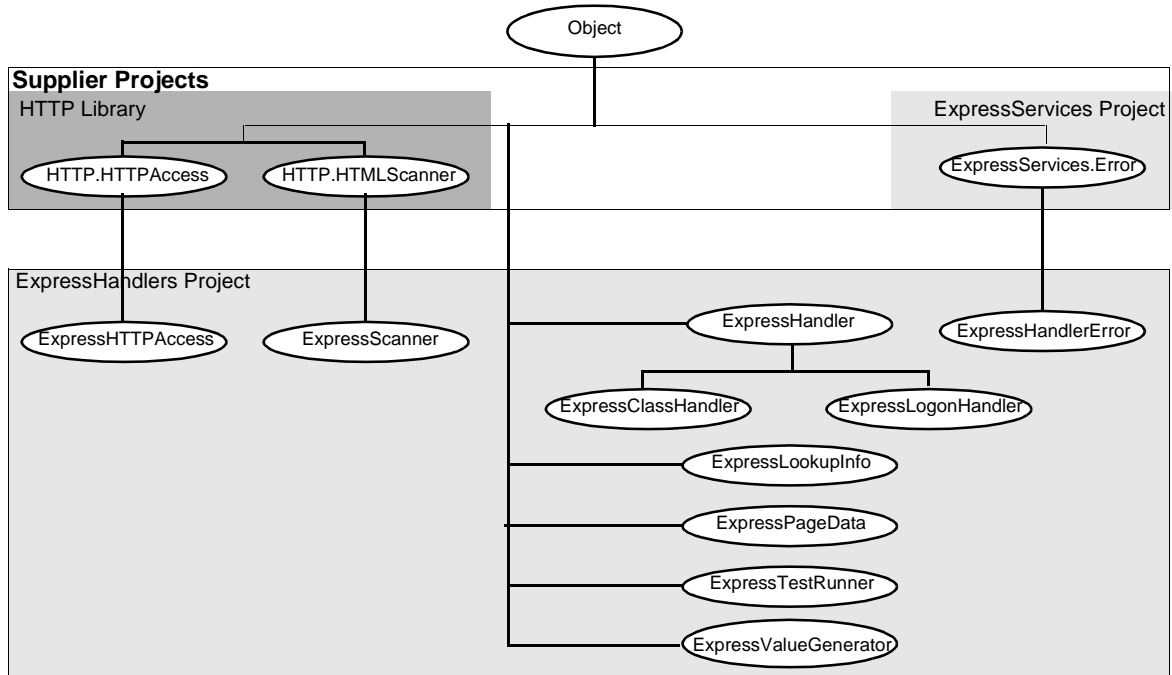
For descriptions of these classes, see iPlanet UDS online help.

The ExpressHandlers Project

The ExpressHandlers project contains a set of superclasses that provides generic functionality for the classes generated from the HTML application model. For example, the ExpressHandler class is the superclass for the *bus_class_pageHandler* classes generated for each of the application's pages. Refer to the online help descriptions of the ExpressHandlers classes for descriptions of these classes and their components.

Figure 1-6 shows the ExpressHandlers project's class hierarchy and the key superclasses of its supplier projects.

Figure 1-6 ExpressHandlers Class Hierarchy and Suppliers



It is the ExpressHandler class and its subclasses that implement the TagHandlerIFace interface (see Figure 1-5 on page 33).

The `html_modelHandlers` Project

The `html_modelHandlers` project contains TOOL classes generated from the HTML application model. These classes provide the Web application server logic for the pages in the model, and are subclasses of classes defined in the `ExpressHandlers` project.

You generate the classes in the `html_modelHandlers` project with either the File > Generate Web Application Server Code command in the HTML Application Model Workshop, or the `CompilePlan` command in the `Fscript` utility.

For a description of classes generated from the business model, see *Customizing Express Applications*.

WebEnterprise Designer generates the following classes in the `html_modelHandlers` project to implement the user application defined in the HTML application model.

Generated Class	Superclass	Custom?	Description
<code>bus_class_pageHandler</code>	<code>ExpressClassHandler</code>	No	Contains the information about the behavior of a page in an application. This class includes the code generated to implement the functions and behavior that you defined for this page.
<code>logon_pageBaseHandler</code>	<code>ExpressLogonHandler</code>	No	Contains the information about the behavior of a logon page in an application. This class includes the code generated to implement the functions and behavior that you defined for this page.
<code>logon_pageHandler</code>	<code>logon_pageBaseHandler</code>	Yes	Contains your customizations for <code>logon_pageBaseHandler</code> . This class is not regenerated when you regenerate your HTML application model code, thereby preserving your customizations.

Generated Class	Superclass	Custom?	Description
<i>html_modelBaseAccess</i>	ExpressHTTPAccess	No	Defines the Access service object, which contains the generated Web connectivity preferences for the application.
<i>html_modelAccess</i>	<i>html_modelBaseAccess</i>	Yes	Contains your customizations for <i>html_modelBaseAccess</i> . This class is not regenerated when you regenerate your HTML application model code, thereby preserving your customizations.
<i>html_modelBaseScanner</i>	ExpressScanner	No	Defines the Scanner service object, which instantiates and holds the handler objects.
<i>html_modelScanner</i>	<i>html_modelBaseScanner</i>	Yes	Contains your customizations for <i>html_modelBaseScanner</i> . This class is not regenerated when you regenerate your HTML application model code, thereby preserving your customizations.
TestRunner	<i>ExpressTestRunner</i>	No	A class that allows you to test run your application without deploying it.

A Yes in the Custom column means the class is a customizable class and will not be regenerated when you regenerate the HTML application model, which preserves your customizations.

WebEnterprise Designer also generates the following service objects in the *html_model*Handlers project:

Generated Service Object	Class Defined By	Description
<i>html_model</i> AccessService	<i>html_model</i> Access	Receives HTTP requests from the Web server and passes them to the ScannerService, then receives the response from the ScannerService and passes the page information back to the Web server.
<i>html_model</i> ScannerService	<i>html_model</i> Scanner	Converts requests from the AccessService service object into pages.

For information about using and partitioning these service objects, see [Chapter 9, “Partitioning and Deploying a WebEnterprise Designer Application.”](#)

Customizable Subclasses in *html_model*Handlers

By default, WebEnterprise Designer does not generate customizable subclasses of *bus_class_page*Handler. For information about creating customizable subclasses of these classes, see [“Creating Customizable Classes” on page 65.](#)

Generated HTML Templates

When you generate Web application server code for your HTML application, WebEnterprise Designer also generates HTML files. These files correspond to the pages in the model, implementing the flow between the pages and the presentation of data in the pages. They are generated into the *html_model* directory under the HTML document root directory.

WebEnterprise Designer generates the following HTML templates in the *html_model* directory of the document root directory. Templates whose names include *bus_class_page* are generated for business class pages; those with *link_page* in the name are generated for link pages. “Logon” templates are generated for logon pages

Generated HTML Template	Description
<i>Main_bus_class_page.html</i>	Top-level HTML definition of a business class page.
<i>Data_bus_class_page.html</i>	Provides the tags used to put the data (but not nested data) in the page to be displayed.
<i>Display_bus_class_page.html</i>	Layout of the Data Display portion of the page, including nested pages.
<i>Insert_bus_class_page.html</i>	Insert mode of the page.
<i>Menu_bus_class_page.html</i>	Menu area for the page (if the page design includes a menu).
<i>Search_bus_class_page.html</i>	Search mode of the page.
<i>Update_bus_class_page.html</i>	Update mode of the page.
<i>Header_bus_class_page.html</i>	Defines the headers of all the model's pages (if the page design includes them).
<i>Footer_bus_class_page.html</i>	Defines the footers of all the model's pages (if the page design includes them).
<i>Scripts_bus_class_page.html</i>	Scripts for the page.
<i>Main_link_page.html</i>	Top-level HTML definition of a link page.
<i>Display_link_page.html</i>	Layout of the data display portion of a link page.
<i>Logon_logon_page.html</i>	The page displayed when a non-validated user attempts access to any page in the application.
<i>LogonFailed_logon_page.html</i>	The page displayed when a user is denied access to the application.
<i>Validate_logon_page.html</i>	The page used to validate the logon. This page is never displayed; it redirects to the first page of the application or the logon failed page as appropriate.
<i>Start.html</i>	Defines a shortcut for a string used in deployment.
<i>manifest.txt</i>	Lists all the HTML files generated and so serves as a catalog of the directory.
<i>report.txt</i>	Reports on the process of generating the HTML templates, and what has been customized.
<i>bus_class_page_qq_field_name.inc</i>	Contains stored and displayed values of a lookup field, if one is specified in the model.

For more information on these templates, see [Chapter 3, “Customizing Generated HTML Templates.”](#) For brief information on the document root directory, see [“Setting Document Root on the ScannerService Service Object Partition” on page 259.](#) For more information on document root, see *Getting Started with WebEnterprise Designer* or WebEnterprise Designer online help.

WebEnterprise Designer Class Interactions

This section provides information on the interactions between the classes described in the previous sections. First, a class diagram of the class hierarchies and class references is provided as a map of the territory. This is followed by a description of the main functions of the key classes in the diagram.

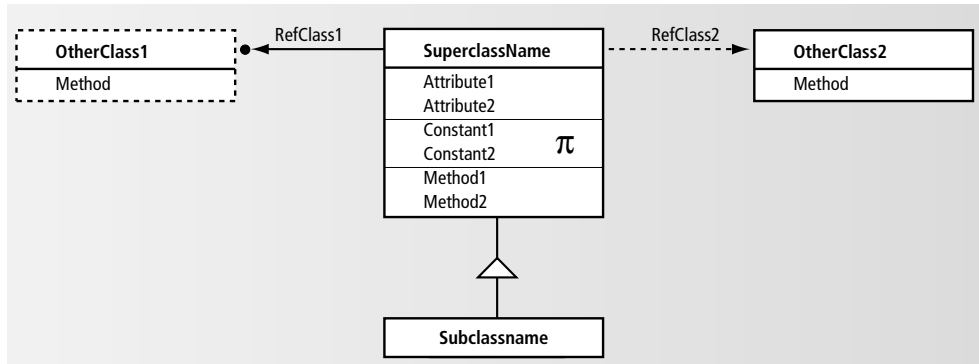
Class Interaction Diagram

This section presents a class interaction diagram of the WebEnterprise Designer class hierarchies. This class diagram uses OMT notation described in the book *Object-Oriented Modeling and Design* (by Rumbaugh, Blaha, Premerlani, Eddy, and Lorenson, published by Prentice Hall) to show the classes and their relationships.

The diagram conventions are:

- arrows between classes mean that the class at the starting end of the arrow contains an attribute whose type is that of the class the arrow points to
- the label of the arrow identifies the attribute
- a dot at the end of an arrow indicates a reference to an array
- a class whose bounding box is a dotted line is a class described in another diagram or manual, and details about the class are usually not given
- key classes display the attributes, constants, and methods referred to in the subsequent discussion

These elements are indicated in the order shown in [Figure 1-7](#). Constants are indicated by the icon used for constants in the Workshop (the pi sign).

Figure 1-7 Example Class Diagram

Declared Type and Runtime Type

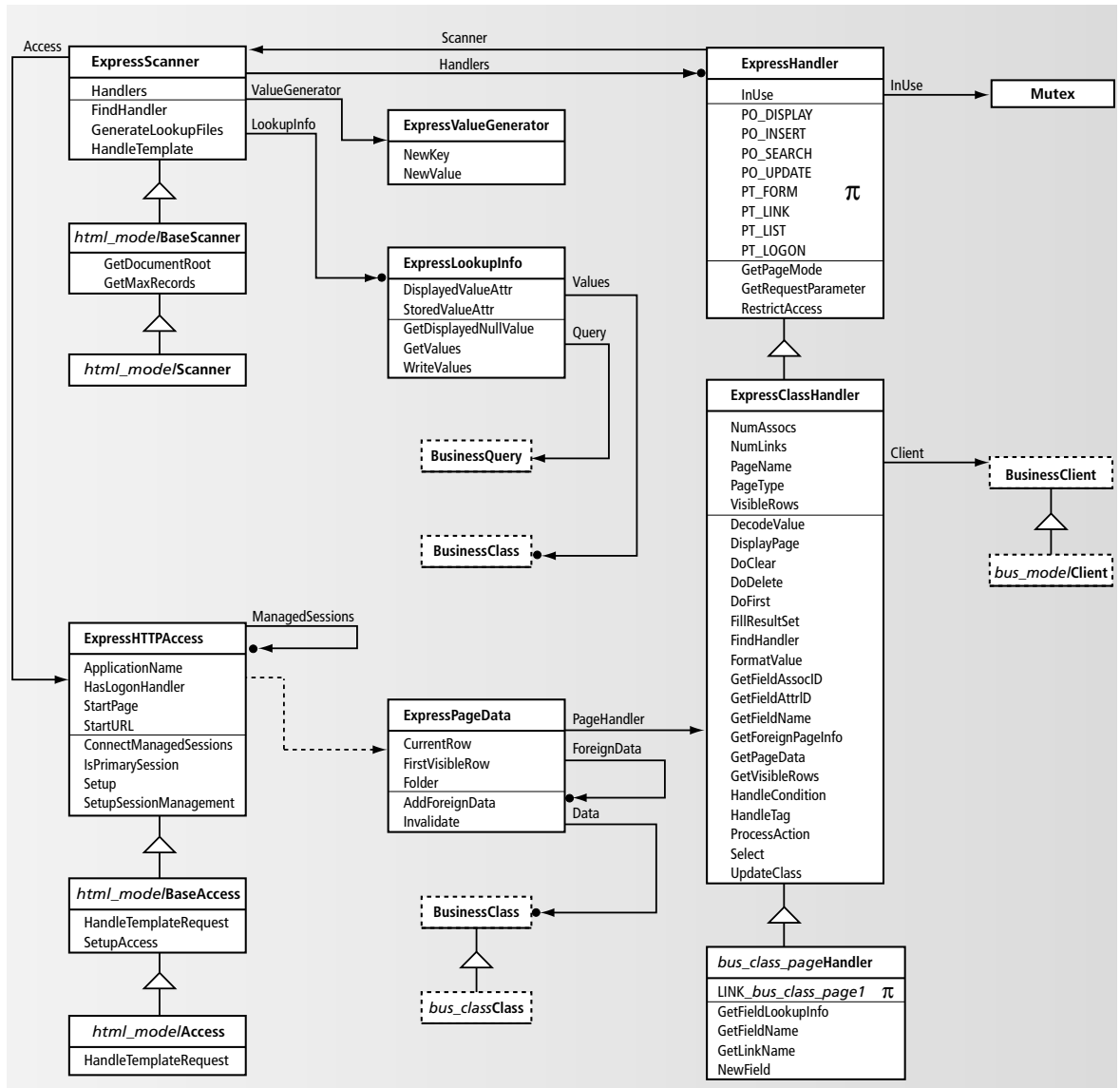
If a class has an attribute that references a superclass in a hierarchy, then at runtime the type of object referenced by that attribute will actually be the customizable subclass. For example, in [Figure 1-8 on page 41](#), the `ExpressHandler` class has an attribute `Scanner` of type `ExpressScanner`. At runtime, the attribute will hold a reference to the customizable `HTMLtutAppScanner` object.

This customizable subclass reference gives you control over the runtime application behavior, because your subclass object and its customizations will be present wherever you see a reference to the base class in the class diagrams in this chapter.

You commonly customize by creating a method in the subclass that overrides the method in the superclass. Overriding methods are called automatically.

Remember, though, to access attributes, methods, or events defined only for the runtime class (and not in the superclass), you must cast the object. Casting means identifying the class of a particular object. See the *TOOL Reference Guide* for more information about casting.

Figure 1-8 WebEnterprise Designer Class Interaction Diagram

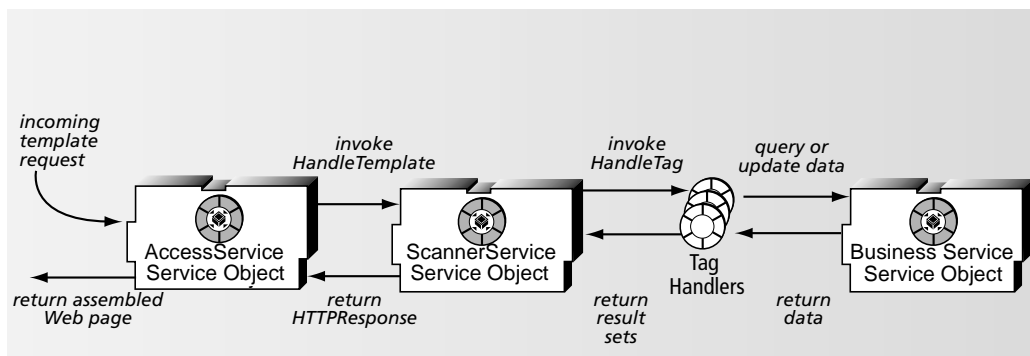


Life of a Template Request

As a context for this information, this section reviews how the parts of a WebEnterprise Designer application interact at runtime.

Runtime interactions of a WebEnterprise Designer application were discussed earlier in “[Runtime Architecture](#)” on page 27. [Figure 1-9](#) and the description that follows provide further detail on how a request from a user’s browser is fulfilled in an HTML application.

Figure 1-9 Life of a Template Request



The steps in the life of a template request, as depicted in [Figure 1-9](#), are:

1. The incoming HTTP request from the Web browser is passed to the AccessService service object *html_modelAccessService*.
2. The *html_modelAccessService* service object:
 - o registers itself as a WebEnterprise server
 - o defines the application’s session management security policies
 - o invokes the *HandleTemplateRequest* method on the Scanner service object *html_modelScannerService*
3. The *html_modelScannerService* service object processes the named template and builds an HTML stream.
4. When *html_modelScannerService* encounters a FORTE EXECUTE tag in the template, it invokes the *HandleTag* method of the page’s handler class.
5. The *HandleTag* method invokes the *ProcessAction* method to perform the requested action.

6. If the action includes retrieving data from a database, the handler class uses the Business Service service object to access the business services, and return data in a result set.
7. The DisplayPage method of the page handler builds the result set for the Web page.
8. If *html_modelScannerService* encounters a FORTE IF tag in the template, it invokes the HandleCondition method of the page's handler class.
9. The HandleCondition method determines whether the requested condition holds and returns either True or False accordingly.
10. *html_modelScannerService* then produces an HTTPResponse object, which contains the complete Web page, and passes this to the *html_modelAccessService* service object.
11. *html_modelAccessService* returns the now-complete requested Web page back to the client Web browser.

Runtime Scenarios

To provide even more detail of some of the steps described above, some of the information on individual classes in the following sections includes object interaction diagrams. Such diagrams are based on the Jacobson notation described in the book *Object-Oriented Software Engineering*.

Object Interaction Diagram Conventions

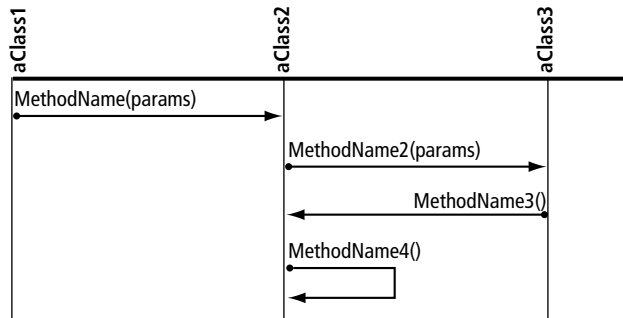
Object interaction diagrams illustrate how objects interact during runtime scenarios in a generated page that employs methods of the class under discussion. These diagrams show only the key objects involved in a scenario and the methods they invoke on each other. You can build diagrams like these on your own by running your application and stepping through it in the TOOL debugger.

The diagram conventions are:

- vertical lines represent objects
 - The object name is given at the top of the vertical line—its name is identical with its class name, but prefixed with the letter “a”.
- the methods that the objects invoke on each other are represented by horizontal lines between objects with an arrow indicating the direction
- the label on the arrow is the name of the method invoked

- time flows from top to bottom
- a method line that points back to the same object is an object that invokes a method on itself

Figure 1-10 Example Object Interaction Diagram



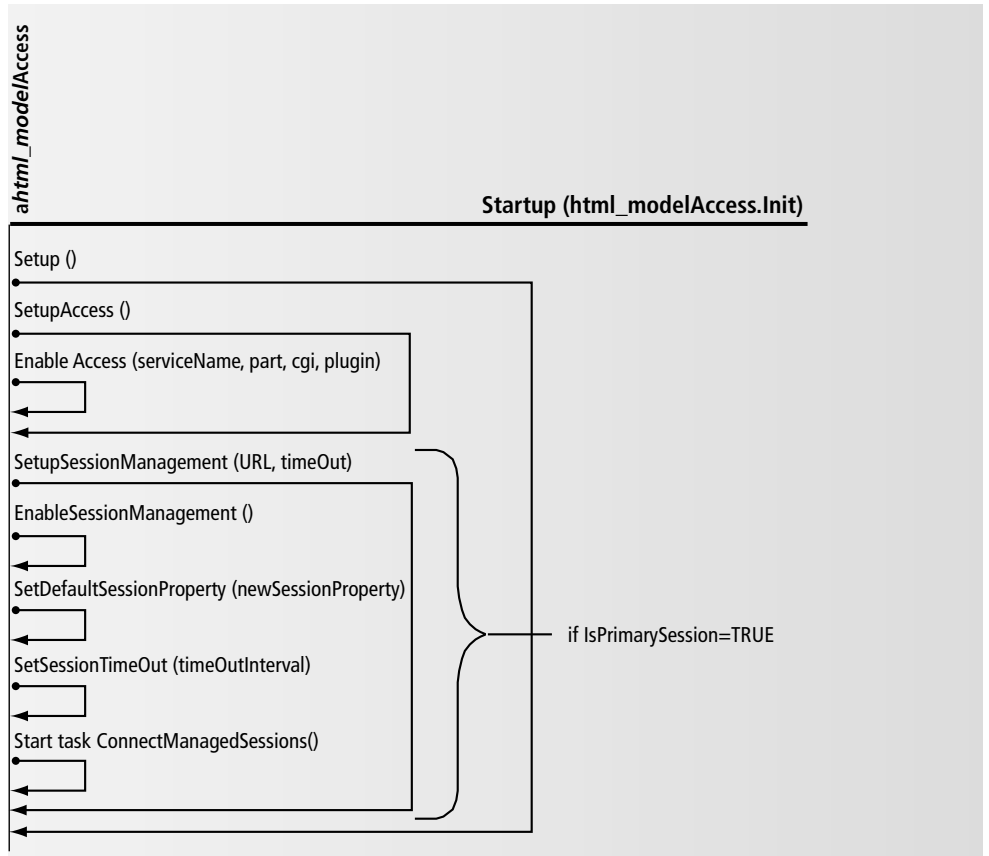
ExpressHTTPAccess

Step 1 and **Step 2** in “[Life of a Template Request](#)” on page 42 outline the functions of the generated *html_modelAccessService* service object in WebEnterprise Designer applications. The *html_modelAccessService* service object is based on the generated *html_modelAccess* class, which is a subclass of ExpressHTTPAccess.

The ExpressHTTPAccess class extends the WebEnterprise HTTPAccess class by defining a number of WebEnterprise Designer-specific attributes and a common initialization method called Setup. Every WebEnterprise Designer application has a generated ExpressHTTPAccess subclass and a generated service object based on the generated class.

Starting the AccessService Service Object

The details of starting the *html_modelAccessService* service object are shown in [Figure 1-11](#). Startup is the result of invoking the *html_modelAccess* class’s Init method.

Figure 1-11 Object Interaction Diagram of the Startup of the AccessService Service Object

The Setup method drives the initialization of the `html_modelAccessService` service object by invoking the following methods:

- SetupAccess method—registers the service object as a WebEnterprise server
- SetupSessionManagement method—defines the application’s session management security policies

Customizing ExpressHTTPAccess

All ExpressHTTPAccess customizations are accomplished by overriding or creating methods in the *html_modelAccess* class. Common customizations are:

- customizing the application's server registration

WebEnterprise Designer applications, by default, employ autoregistration (in which the service object automatically registers itself with *fortecgi*). To use manual registration, override the *html_modelAccess* class's *SetupAccess* method and modify its *EnableAccess* method accordingly.

- customizing the application's session management

The *SetupSessionManagement* method of the *ExpressHTTPAccess* class defines the WebEnterprise session management characteristics of the application. Depending on the setting of the *HasLogonHandler* attribute, you can define the application's HTML templates with the *SESSION_AUTOCREATE* property (if no logon page is defined) or the *SESSION_REQUIRED* property.

See *Getting Started with WebEnterprise Designer* for a description of WebEnterprise Designer default session management.

- authenticating logon information

When a logon page is defined, you must override the *LogonSession* method of the *HTTPAccess* class to validate the authentication information supplied by the client user (typically a username and password).

See [Chapter 8, "Customizing Application Security,"](#) for information on creating and using logon pages.

- setting up the primary application to perform session management for included applications

By default, each HTML application manages its own WebEnterprise sessions, including logon (if defined) and HTML template security. Another common model is to have one (primary) HTML application perform session management for itself and other (subsidiary) HTML applications. (An included HTML application is a subsidiary application.)

To create such a configuration, you must customize the primary HTML application and all of the subsidiary HTML applications as follows:

Subsidiary HTML applications: In each subsidiary application, customize the `IsPrimarySession` method to return `FALSE`. You can do this through the application-wide “Is subsidiary application” customization point in the Page Handler Customization Wizard. Just selecting this customization is enough; it will generate the proper method code. You need not customize the method code it generates.

Primary HTML application: You must customize the `ConnectManagedSessions` method of the primary application to instantiate and populate the `ManagedSessions` attribute with references to the `html_modelAccessService` service objects of the subsidiary applications. You can do this with the application-wide “Define subsidiary applications” customization point. Modify the code generated by this customization point to name your subsidiary application(s) in the line:

```
ManagedSessions.AppendRow( subsidiaryAccessService );
```

See [Chapter 8, “Customizing Application Security,”](#) for more details and an example.

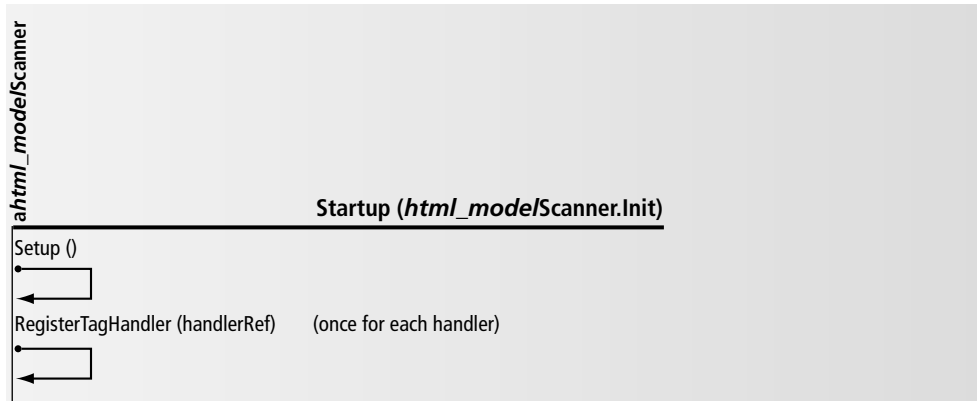
ExpressScanner

Procedures in [“Life of a Template Request” on page 42](#) described how the `html_modelScannerService` service object coordinates the assembly of the Web page that fulfills the template request in WebEnterprise Designer applications. The `html_modelScannerService` service object is based on the generated `html_modelScanner` class, which is a subclass of `ExpressScanner`.

The `ExpressScanner` class extends the WebEnterprise `HTMLScanner` class by defining a number of WebEnterprise Designer-specific attributes. Every WebEnterprise Designer application has a generated `ExpressScanner` subclass and a generated service object based on the generated class.

Starting the ScannerService Service Object

The details of starting the `html_modelScannerService` service object are shown in [Figure 1-12](#). Startup is the result of invoking the `html_modelScanner` class’s `Init` method.

Figure 1-12 Object Interaction Diagram of Startup of the ScannerService Service Object

ExpressClassHandler

The `ExpressClassHandler` class is the primary class used to implement application logic in a WebEnterprise Designer Web application server. A subclass of the `ExpressClassHandler` class is generated for every page in an HTML application (see [Figure 1-4 on page 32](#)). In [“Life of a Template Request” on page 42](#), it is the methods of these “page handlers” that the scanner invokes to translate iPlanet UDS HTML tags into runtime data.

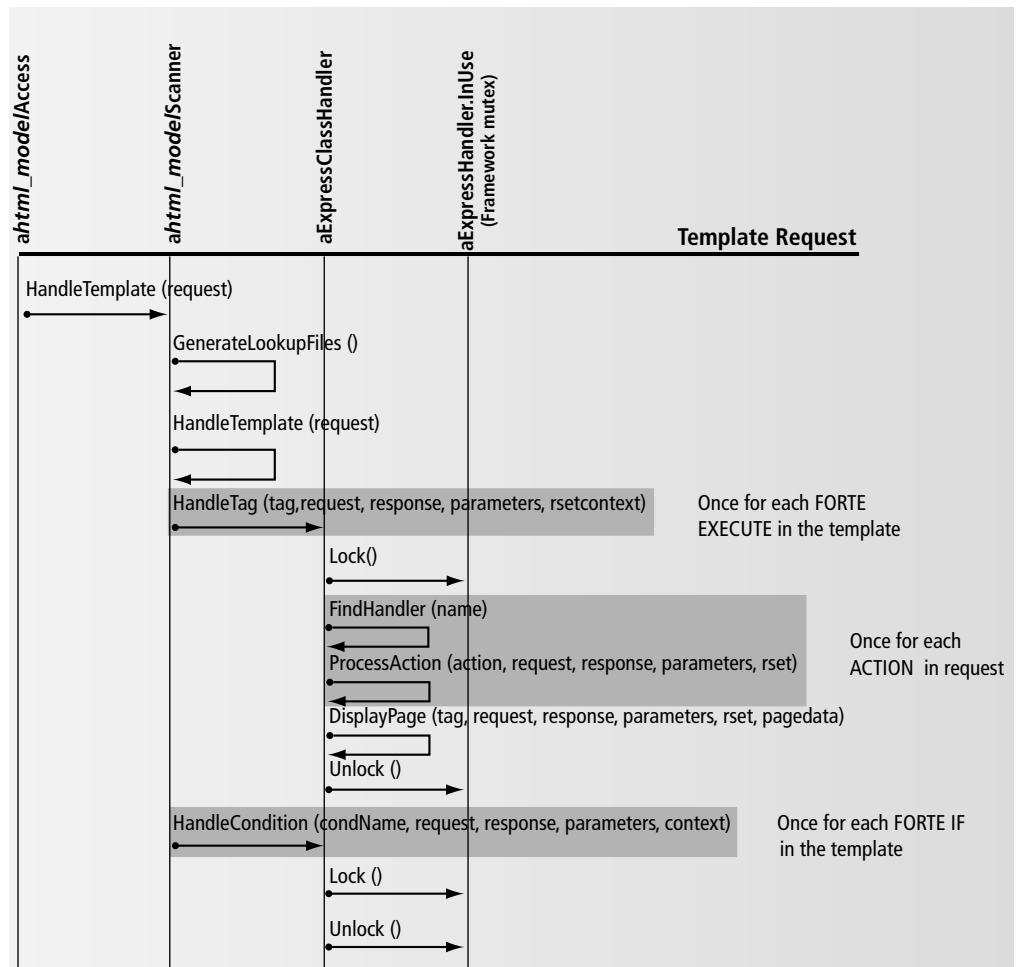
Page handler classes use two methods to respond to iPlanet UDS tags: the `HandleTag` method and the `HandleCondition` method. The method signatures for these methods are predefined in the `TagHandlerIFace` interface (of the HTTP library), which is implemented by the `ExpressClassHandler` class.

Helpful background to this section is the chapter on [Creating Pages Using Templates in *A Guide to WebEnterprise*](#), which describes what an iPlanet UDS HTML template is and provides an example of how the FORTE EXECUTE and FORTE ITERATE tags are used in a template to generate a frame.

Runtime Control Flow

The details of a template request are shown in [Figure 1-13](#). This diagram provides further details of the procedure described in [“Life of a Template Request” on page 42](#), and provides a starting point for the discussion that follows. It is offered here, because it shows the role played by the `ExpressClassHandler` class (actually, by the generated subclasses, or page handler classes) in this process.

Figure 1-13 Object Interaction Diagram of a Template Request



The generated *html_modelScannerService* service object instantiates the generated *bus_class_pageHandler* objects and invokes their `HandleTag` or `HandleCondition` methods when it encounters the following tags embedded in the templates requested by Web clients:

```
<?forte execute ...>  
<?forte if ...>
```

NOTE In this manual, as in *A Guide to WebEnterprise*, the tags are referred to in text in abbreviated form (for example, “the FORTE EXECUTE tag”), while the actual syntax requires a preceding question mark (as above). For a full description of iPlanet UDS HTML tag syntax, see *A Guide to WebEnterprise*.

Each EXECUTE tag invokes the `HandleTag` method of the associated `ExpressClassHandler` object. Each IF tag invokes the `HandleCondition` method of the associated `ExpressClassHandler` object.

The `HandleTag` method:

- invokes the `FindHandler` method to find the proper handler object to execute this tag request (this will be the current object, unless this is a request from a nested page)
- invokes the `ProcessAction` method to perform the action being requested
- invokes the `DisplayPage` method to format any result data into a WebEnterprise result set

Formatting the data for the current logical page might require other logical pages, if the current page has any nested or folder logical pages. For more information, see the section [“Foreign Result Sets” on page 55](#).

The `HandleCondition` method:

- determines whether the requested condition holds and returns either `True` or `False` accordingly

Referenced Objects

An `ExpressClassHandler` object (more particularly, a page handler object) holds a reference to the `html_modelScanner` service that uses it. The scanner is used to provide global services for the Web application server including key generation, and generation of lookup files.

A page handler object, at initialization, creates and holds a reference to the generated `BusinessClient` for the business class that the page is based on. The generated `BusinessClient` is used to retrieve and modify business classes.

Field Identification

Each business class page has a number of attributes specified for display as fields on one or more of its HTML page mode templates—Display, Search, Insert, and Update. These may be attributes of the primary business class or of associated business classes.

Primary and Associated Business Classes The business class that the page is based on is the *primary* business class for that page. An *associated* business class is one that is the target of an association from a primary business class, or from another associated class in the business model. The association can be either one-to-one, many-to-one, or optional.

Field Indexes and Association IDs

Each attribute is given a field index and association ID to identify it and place it in a WebEnterprise result set, so that it can be properly rendered on the page. In fact, a field index and association ID pair are given to all of the following items:

- each attribute that appears on any of the HTML page mode templates for the business class page
- each attribute that is indirectly referenced by the business class page
- each association needed to navigate from the primary business class to all associated business classes used by the page

The field index and association ID are integer quantities and are unique only to the current business class page. Fields are numbered consecutively from 1, within an association.

The association ID identifies the field or association uniquely within the business class page, and, if it is a field, indicates whether it is from the primary or an associated business class. A field from the primary business class has an ID of 0; fields from associated business classes have IDs unique within the business class page, numbered consecutively from 1.

Only associations that are actually used are assigned association IDs. An association is used if:

- one or more of the attributes of the associated business class is used by the business class page
- one or more attributes of any business class included in the associations emanating from the associated business class is used by the business class page

NOTE This definition can be applied successively.

Converting Between Field Index and Attribute ID

Every valid association ID and field index pair is associated with an attribute of the business class indicated by the association ID. Use the `GetFieldAttrID` method to find the attribute ID of this attribute.

You can also do the inverse conversion: to find the field index from the association ID and attribute ID, use the `GetFieldIndex` method.

Data Transfer

Data displayed on an HTML page is rendered into text at some point and WebEnterprise Designer applications perform this rendering automatically. The only time you might need to affect this is when you want to modify the textural representation of data to be placed on the form. Within the `ExpressClassHandler`, result data is contained in business class objects that you can manipulate directly, using the `FormatValue` and `DecodeValue` methods. (There are customization points available for such customizations. For more information, see [“Formatting Fields”](#) and [“Decoding or Validating Fields”](#) on page 89.)

Result Sets

Result data that is to be rendered on an HTML page is placed in a WebEnterprise `ResultSet` object. Of the total result set data retrieved for the Web client, the `ResultSet` object holds only that part that is to be rendered on the requested page. For example, if a user submits a query that returns a result set of 50 rows, but the page being rendered is a form with only one visible row, then one of the 50 rows is placed in the `ResultSet` object, along with the information that the row is a specific one of 50 rows.

For more information on result sets, see [“ExpressPageData”](#) on page 55.

Formatting Data into a WebEnterprise Result Set

DisplayPage and FillResultSet are the methods used to populate the ResultSet object. DisplayPage determines the specific business class objects to be formatted into the ResultSet object. FillResultSet fills the ResultSet object with a single business class invoking FormatValue on each attribute added to the ResultSet object.

DisplayPage is the last method invoked by the HandleTag method. Therefore, whatever changes you make to data in the result set, they must be made before the DisplayPage method is invoked.

For a descriptive list of all variables pertinent to a ResultSet object, see [“Variables” on page 60](#). For information on customizing DisplayPage, see [“Formatting Custom Fields” on page 89](#).

Decoding Data from a WebEnterprise Request

When an HTTP request is received, particularly a request from an HTML form, there are often parameters that must be decoded. When a request comes from a WebEnterprise Designer Insert or Update form, some of the parameters represent new data values for attributes in the result set. These values are updated by the UpdateClass method. When a request comes from a WebEnterprise Designer Search form, some of the parameters represent query constraints that need to be added to the Express query being run. Adding query constraints is done by the AddConstraints method. Both UpdateClass and AddConstraints use the DecodeValue method on each parameter. DecodeValue can then be used as a common point to implement field validation.

Result Sets and Session Management

Since only one instance of a handler class is instantiated for each logical page in the HTML application model, the handler class cannot hold any Web client-specific state, such as result data. To maintain client-specific state, the handler class uses WebEnterprise session management features. Each client’s result set is placed in an ExpressPageData object, which is then held by the WebEnterprise session using the page name as the key. To retrieve the result set associated with the current request, the handler class uses the HTTPRequest object passed on the HandleTag or HandleCondition request, and invokes the GetSessionData method on it, passing the current page name as the key.

The result set for a given Web client therefore becomes a collection of result sets, one for each logical page visited, and is maintained by the generated *html_model*Access service.

Connections Between Handlers

There are no explicit connections between handlers for different logical pages, but sometimes, for example with nested pages, one handler might need to access another. Two mechanisms are available for obtaining a reference to another handler:

- using the scanner's `Handlers` attribute

A scanner's `Handlers` attribute is an array that holds a reference to every handler. You can therefore compute the required `Handlers` reference if you know the its array index.

- using the scanner's `FindHandler` method

You can use this method to find any handler based on its name.

ExpressLogonHandler

The `ExpressLogonHandler` class is the other “page handler” class of WebEnterprise Designer applications. An application-specific subclass of the `ExpressLogonHandler` class is generated only if a logon page exists in the application. This class manages the process of user authentication for HTML applications.

The `ExpressLogonHandler` class is the superclass for the HTML application's `BaseLogonHandler` class and its subclass, `LogonHandler`.

The HTML application's `SessionCreationURL` attribute is set to the URL of the logon page, causing that page to be presented to any “new” (meaning unvalidated) client. When the page's form is completed and submitted, the `ExpressLogonHandler` class's `HandleCondition` method is invoked to validate the user's authentication data and either start a WebEnterprise session or throw an exception.

For further details on `SessionCreationURL`, see [“How the Logon Page is Activated” on page 233](#).

Customizing ExpressLogonHandler

Using a Logon page requires customization of the logon process. By default, no validation of the Logon page's user authentication fields is performed—the client is always logged in.

Overriding the application's LogonSession method performs this customization. You can do this by selecting the Logon Validation customization point in the Page Handler Customization Wizard. For detailed information on customizing the logon process, see [Chapter 8, "Customizing Application Security."](#)

ExpressPageData

The ExpressPageData class holds the result set for a particular business class page for a single Web client. The result set data is saved as WebEnterprise session data, with the logical page name as the key. The ExpressPageData class keeps track of the current row and the rows visible on the rendered HTML page by means of the CurrentRow and FirstVisibleRow attributes. The CurrentRow attribute is used for operations that act upon a single row, such as update and delete.

When search or select requests are processed by a page handler, the old data held by the associated ExpressPageData object is invalidated (with the Invalidate method) and replaced with the requested data. Insert, update, or delete requests can modify data held by the ExpressPageData object.

Foreign Result Sets

Displaying a logical page containing nested or folder pages requires access to the ExpressPageData objects for the nested or folder pages. Access to these objects is achieved with the ExpressPageHandler class's DisplayPage and GetForeignPageInfo methods.

To retrieve data for nested or folder links, the DisplayPage method calls the GetForeignData method to access the page handler for the nested or folder page. The results are stored in the nested or folder page's ExpressPageData object, which is attached to the parent page as an associated result set by the ExpressPageHandler class's AddForeignData method.

ExpressValueGenerator

The ExpressValueGenerator class is used to generate new keys for business classes when the insert command is used.

ExpressLookupInfo

The `ExpressLookupInfo` class is used to validate fields that have drop-down or radio-list validation. This class is the validation object used to retrieve and save the displayed and stored values. It is tied to an attribute of a business class in one of the included business models, and implements a lookup table object. The scanner holds one validation object for each unique stored and displayed attribute combination used by all fields in all pages in the application.

Normally, each field has its own validation object. However, if two fields use the same business class and the same stored and displayed attributes, they will each use a single copy of the validation object.

Modifying the Displayed Null Value

Override the `GetDisplayedNullValue` method if you want to display a string other than the default string “<NULL>” to represent a null selection.

WebEnterprise Designer HTML Template Elements

This section describes various features of WebEnterprise and HTML that are used by WebEnterprise Designer templates. These features include:

- links
- variable references
- URL parameters

For further information on these features, see the “Planning Web Pages” chapter in *A Guide to WebEnterprise*.

Links

Web pages use links to allow Web users to jump quickly from page to page. In the HTML application model, you define links to connect the pages. In addition, the commands that you specify for the individual pages are links to different modes of the page (the `Insert` command links to the `Insert` mode of the page, for example).

A link is an anchor (for example ``) with a URL (universal resource locator) that is embedded in an appropriate place in an HTML template. The two types of links previously mentioned are automatically formatted by WebEnterprise Designer. You can customize these or add your own to the page. This section discusses some elements of link formatting that are specific to WebEnterprise Designer.

A typical embedded URL reference in an HTML statement in a WebEnterprise Designer HTML template might look like this:

```
<a HREF="$$ (FORTE.ExecURL) ?ServiceName=eCoolService&TemplateName=eCool/Display_$$
(USER.TopPage).html&Action=MovieHandler.Last&Uniquifier=$$ (FORTE.UniqueID) "
target="_fortedisplay">
```

URL Parameters

Some of the URL parameters used in WebEnterprise Designer URL references are generic to WebEnterprise, and some are specific to WebEnterprise Designer. The URL parameters below that are identified as WebEnterprise parameters are described in detail in *A Guide to WebEnterprise*.

- **ServiceName**—(WebEnterprise) identifies the name of the Web access service
Syntax: `&ServiceName=html_modelService`

Example

```
ServiceName=HTMLtutAppService
```

This parameter is used by the interface (whether `fortecgi` or the `iPlanet UDS` Web server plug-in). The `fortecgi` program or the 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 the plug-in 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 the plug-in returns a Request Failure error to the Web user with the message "Service not found."

- **TemplateName**—(WebEnterprise) identifies the name of the WebEnterprise Designer template to be used to construct the page

```
Syntax: &TemplateName=html_model/page_mode_bus_class_page.html
```

Example

```
&TemplateName=HTMLtutApp/Update_Customer.html
```

This parameter is required and, on UNIX systems, is case-sensitive. Based on this value, the Web access service passes the current request to either the page handler or the ScannerService SO.

- Action—(WebEnterprise Designer) specifies the action requested

The ExpressHandlerClass class's ProcessAction method processes actions. This method compares the text value of the Action parameter with the known actions and executes the indicated method. By convention, these methods are named the same as the Action value with a "Do" prefix. For example the Next action invokes the DoNext method.

Syntax: `&Action=handler.action`

The *handler* prefix is optional. If unspecified, the current handler is assumed. However, sometimes the action must be executed by a handler other than the current one, which typically happens with nested pages. To execute the "Next" action on a nested page, the link must request the outer page (otherwise only the nested page would be displayed). However, when the outer pages handler gets the "Next" request, it must dispatch it to the nested handler. This is done by specifying the nested page in the Handler parameter.

Example

```
http://www.myServer.com/page.html?ServiceName=AppService
&TemplateName=myApp/Display_Movie.html&Action=Next
```

The `&Action=Next` parameter of this URL causes the page to be redisplayed after executing the handler class's Next function. (The Next function causes the result set to be scrolled to the next element.)

There can be several Action parameters. When there is more than one, each action is performed in turn.

- LinkName—(WebEnterprise Designer) specifies the name of the link that requested the current page

The name is formed by concatenating the name of the HTML application with the name of the link (as seen in the Link Properties dialog title bar) separated by a period.

Syntax: `LinkName=html_model.link_name`

Example

`LinkName=HTMLtutApp.CustomerLink1`

- **ReturnTemplate**—(WebEnterprise Designer) used to remember the template requesting a function

Typically, the ReturnTemplate is displayed after the page needed by the requested function. For example, when requesting the Update action, the update HTML template is displayed. After the update is complete, the original page should be redisplayed.

Syntax: `&ReturnTemplate=html_model/page_mode_bus_class_page.html`

Example

`c&ReturnTemplate=HTMLtutApp/Display_Home.html`

- **Selection**—(WebEnterprise Designer) used to specify the fields on the page that are being used to provide constraints for a select operation against the business service

The Selection parameter is used with the Select action (`&Action=Select`) on a search page. The fields specified in the Selection parameter then also appear as parameters providing the values for the fields being constrained.

Syntax: `&Selection=bus_class_page_qq_field`

Example

`&Selection=CustomerOrder_qq_CustomerNumber`

- **Uniquifier**—(WebEnterprise Designer) specifies a unique value; used to defeat caching done by the browser in some cases

This parameter and its variable are used when the requested page has been used before, and therefore contains data, but a “fresh” page is desired. To retrieve the fresh page, specify a unique number with the parameter, so that the retrieved page is not the cached one.

For example, if the user activates the Next command, the page being requested is exactly the same page as the current page, but with different data. If browser caching is enabled, the browser will satisfy this request from the cache, which means displaying the same data, not the new data. Using the Uniquifier parameter with the FORTE.UniqueID variable creates a request that does not

match any page in the browser's cache. This forces the request to be passed on to the application, which processes the Next command, thereby advancing the result set to the next row. Then, when the page is populated with data, the next row will be displayed.

Syntax: `&Uniquifier=$$ (FORTE.UniqueID)`

Example

```
&Uniquifier=$$ (FORTE.UniqueID)
```

Variables

To use a result set in a template, you first use the HandleTag method to generate the result set. As each member of the result set is generated, it is assigned a name. This allows you to refer to any member in a result set using an iPlanet UDS variable. Variable references contained in HTML templates have the following form:

```
result_set_name.result_set_member_name
```

The FORTE EXECUTE and FORTE ITERATE commands define the result set names. The variable names are defined by the application logic.

FORTE result set WebEnterprise defines one special result set—the FORTE result set—that always exists and into which WebEnterprise places various useful variables, for example FORTE.ExecURL (described below).

Other special result sets In addition to FORTE, WebEnterprise Designer uses three other result sets:

- USER—holds various static values for a request
- entry—used for all data that is being placed on the current page
- listentry—the iteration result set for the data rows for the current page

For more information on result sets and iPlanet UDS variables in WebEnterprise, see the chapter “Creating Pages Using Templates” in *A Guide to WebEnterprise*.

- FORTE.ExecURL—(WebEnterprise) specifies the path to the Web server used by the application

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 fortectgi program or the iPlanet UDS Web server plug-in.

EnableAccess starts the Web client's access to the current Web access service.

A URL embedded in a template might appear as:

```
<a class="alink" HREF="$$ (FORTE.ExecURL)?ServiceName=HTMLtutAppService&Template
Name=HTMLtutApp/Search_Customer.html&ReturnTemplate=HTMLtutApp/Display_$$ (USE
R.TopPage).html&Uniquifier=$$ (FORTE.UniqueID)" target="_fortedisplay">
```

And the actual URL link would be expanded to:

```
<a class="alink" HREF="http://www.forte.com/cgi-forte?ServiceName=HTMLtutAppSe
rvice&TemplateName=HTMLtutApp/Search_Customer.html&ReturnTemplate=HTMLtutApp/D
isplay_$$ (USER.TopPage).html&Uniquifier=$$ (FORTE.UniqueID)"
target="_fortedisplay">
```

- FORTE.UniqueID—(WebEnterprise Designer) returns a unique number; used to defeat the caching mechanism

See the item “Uniquifier” in the section “URL Parameters” on page 57.

Example

```
&Uniquifier=$$ (FORTE.UniqueID)
```

- USER.TopPage—(WebEnterprise Designer) defines the outermost logical page being displayed

With nested or folder logical pages, the TopPage is not always the same as the current logical page. This parameter is set by the `Display_bus_class_page.html` template to `bus_class_page`. The various `Data_bus_class_page.html` templates included by the Display page always refer to the TopPage parameter, rather than explicitly using the `bus_class_page` name.

Example

```
&TemplateName=HTMLtutApp/Display_$$ (USER.TopPage) .html
```

- `entry.RequestStatus_busClassPage`—the page’s request status

Each page has a request status formatted into this variable that describes the outcome of the request. For successful search requests, this variable contains a string that describes the current row number and the number of rows found by the search request. (for example, “row 1 of 5”).

- `entry.CurrentRow_busClassPage`—the current row number for the indicated page

When a search request returns more than one row, this variable indicates which row a given action (for example, Update) will operate on.

- `entry.CurrentRowIndex_busClassPage`—the index, within the visible rows, of the current row from the result set

For example, if the result set contains 10 rows, rows 3 through 7 are being displayed, and row 4 is the current row, then this variable contains the value 2, because row 4 is the second row from 3.

- `entry.FirstVisibleRow_busClassPage`—the row number of the first row being displayed from the result set

For example, if the result set contains 10 rows, rows 3 through 7 are being displayed, and row 4 is the current row, then this variable contains the value 3, because row 3 is the first row from the result set being displayed.

- `entry.list_busClassPage`—a list of data rows

This variable contains a list of data rows that are iterated using the FORTE ITERATE command.

- `entry.Rows_busClassPage`—number of rows in the result set
- `entry.VisibleRows_busClassPage`—number of rows being currently displayed
- `listentry.qqRowNumber`—value of the current iteration row of listentry
- `listentry.busClassPage_qq_fieldName`—the value of an individual field on a particular page for the current iteration row of listentry

Customizing WebEnterprise Designer Application Classes

In general, you customize an application's logic by using the Page Handler Customization Wizard. The Page Handler Customization Wizard provides a set of common customizations.

Topics covered in this chapter include:

- using the Page Handler Customization Wizard
- creating customizable classes
- deleting customizations and customizable classes
- online customization examples
- customizing generated handler classes
- adding business rules to a handler
- global customizations

Note that the examples and illustrations in this chapter use the example created in *Getting Started with WebEnterprise Designer*, and make references to the class and model names used there. Also, this chapter assumes some familiarity with [Chapter 1, "WebEnterprise Designer Application Architecture."](#)

Overview

Read this overview to learn some basic facts about customizing WebEnterprise Designer application classes. Before making customizations, make sure that the behavior you want cannot be accomplished by simply setting options within the HTML Application Model Workshop or Business Model Workshop and regenerating your application.

Before You Begin

The following are suggestions you should take into consideration before you begin your customizations.

- Use the Page Handler Customization Wizard (see [“Customizing With the Page Handler Customization Wizard” on page 67](#)) if possible, rather than create customizations directly in generated projects.
- You can choose to customize individual page handler classes, or make application-wide customizations by modifying classes by modifying the *html_modelAccess* or *html_modelScanner* classes.
- Always try to make your customizations in such a way that they continue to work after you later make changes to and regenerate your HTML application model. Test frequently after making changes.
- It is best not to create new classes in projects generated by WebEnterprise Designer. Try to keep to only generated classes in these projects (you can of course customize these classes—just do not add new ones). Add new classes in a separate supplier project.
- Decide whether you need to customize a few classes or many. Your decision will determine whether or not to turn on the Always Generate Custom Classes toggle. When this is turned on, WebEnterprise Designer automatically generates a “Base” class and a leaf-level (customizable) class for each business class page in the model. This may speed development if you will customize most or all generated classes.

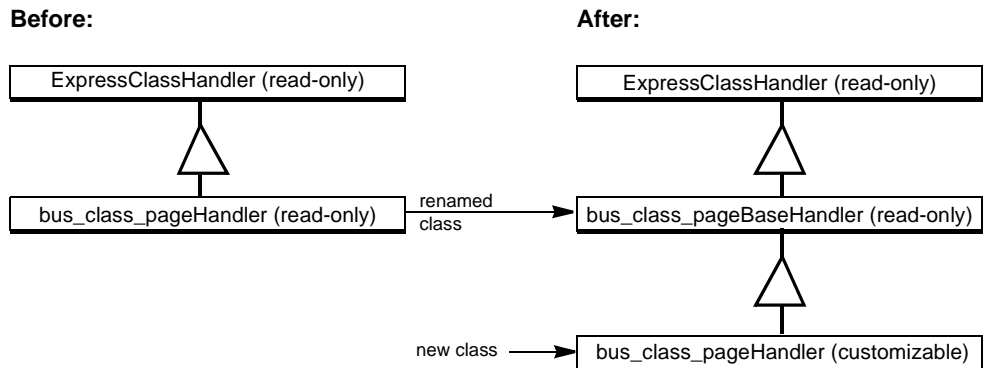
Creating Customizable Classes

WebEnterprise Designer generates both customizable and base (read-only) classes for all components of the HTML application model except for page handler classes, for which only base classes are generated. This strategy generates the minimum number of classes required by the application, and therefore the smallest image size for deployed applications.

To implement customizations that involve page handler classes requires first generating customizable classes for the page handlers. You do this automatically for an individual page when you start the Page Handler Customization Wizard for that page. Alternatively, you can create a full set of customizable classes for all pages in the model by enabling the Always Generate Custom Classes option on the Custom Generation Options dialog. (This dialog is displayed when you choose the File > Custom Generation Options... command).

When you create customizable page handler classes—whether a single one or a full set—WebEnterprise Designer automatically expands the class hierarchy, with a Base class above the leaf-level page handler class. All the components that were generated into the base class are copied to the leaf-level class. The base class is renamed to *bus_class_pageBaseHandler*, and the new, customizable leaf-level class is named *bus_class_pageHandler*. [Figure 2-1](#) illustrates this principle.

Figure 2-1 Naming Conventions Before and After Creating Customizable Classes



If you decide you do not want the customizable class, you delete it using the Page Handler Customization Wizard (see [“Deleting Customizations” on page 72](#)). This collapses the three-class hierarchy back into the two-class hierarchy and renames the read-only Base class to its original name.

Creating a Single Customizable Page Handler Class

The Page Handler Customization Wizard allows you to customize individual classes. When you invoke the Page Handler Customization Wizard for the first time on a specific class, WebEnterprise Designer automatically expands the class hierarchy for the particular class, as shown in [Figure 2-1](#). For more information, see [“Customizing With the Page Handler Customization Wizard” on page 67](#).

Creating a Full Set of Customizable Page Handler Classes

If you know you will be customizing many classes, it might be more convenient to generate the hierarchy at the onset of your development cycle. You do this by setting the Always Generate Custom Classes toggle in the Custom Generation Options dialog. When set, this option tells WebEnterprise Designer to create the full hierarchy automatically for every business class page in your model. When you use this option, note:

- Turning it on turns it on for all the business class pages in the HTML application model; it can then be turned off only by deleting all the customizations on each class using the Page Handler Customization Wizard. You should not delete the individual classes manually from the generated project.

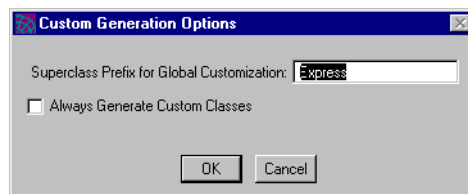
In other words, you create customizable classes for all business class pages in the model with one step, but you must remove individual customizable classes separately.

- Turning it off will affect only new classes created from the time you turned it off, resulting in some Base classes having “Base” in their names and some not.

► To create customizable subclasses for every business class page in your model

1. In the HTML Application Model workshop, choose the File > Custom Generation Options... command.

The Custom Generation Options dialog appears.



2. Turn on the Always Generate Custom Classes toggle.

This toggle causes WebEnterprise Designer to generate customizable leaf-level classes for every class in the business model or page in the HTML application model.

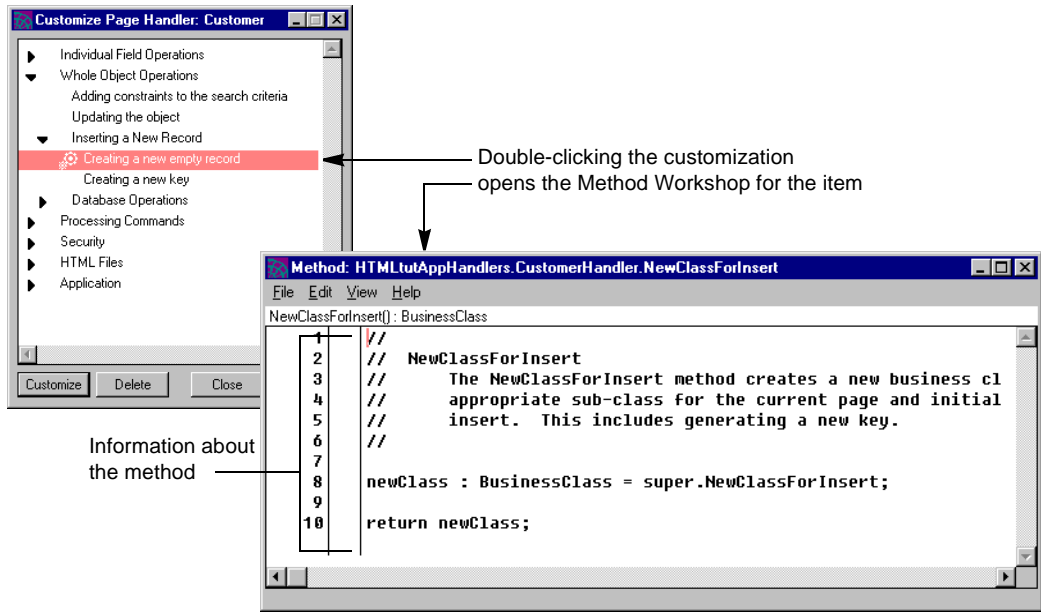
3. Click OK.
4. Generate code by choosing the File > Generate Web Application Server Code command.

See “[Deleting Customizations](#)” on page 72 for information on deleting customizable classes.

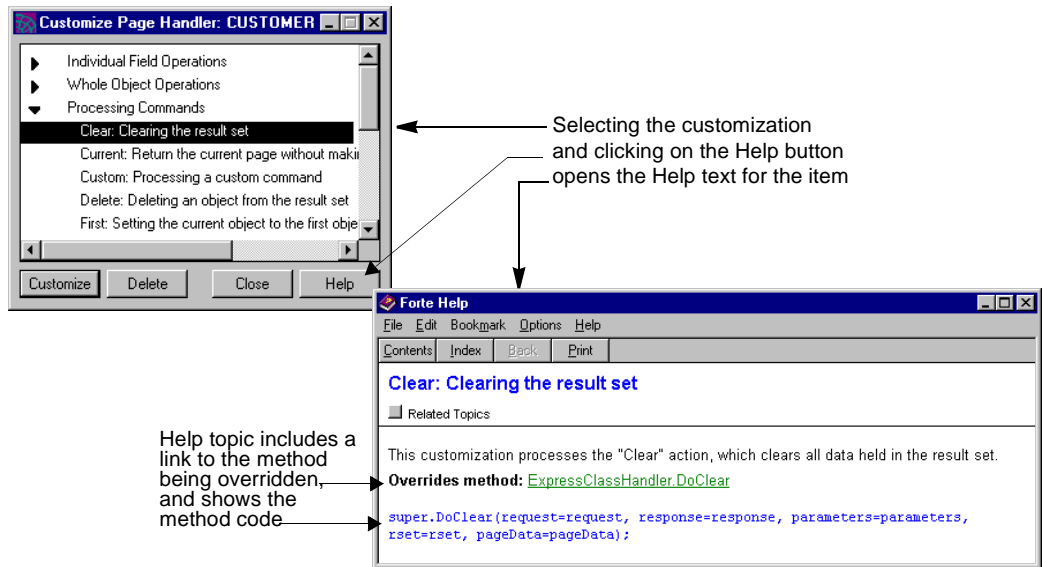
Customizing With the Page Handler Customization Wizard

The Page Handler Customization Wizard assists you in customizing your WebEnterprise Designer applications. When you start the Page Handler Customization Wizard on a business class page, you are presented with a list of categories that contain common customizations. When you double-click on one of the customizations, the Page Handler Customization Wizard automatically opens a Method Workshop window, displaying the appropriate method where you will place your customization code, as shown in [Figure 2-2](#). The initial method code contains documentation of the method parameters and return code, and, if required, a super.*method* statement and a return statement.

Figure 2-2 Page Handler Customization Wizard Opening the Appropriate Method Workshop



In addition to locating your customization code for you, the Page Handler Customization Wizard has online help associated with each customization. You can access help by clicking the Help button while a customization is selected, as shown in Figure 2-3, or by browsing through the Help system Index to the List of all Page Handler Customizations.

Figure 2-3 Page Handler Customization Wizard's Help on a Customization Topic

You also use the Page Handler Customization Wizard to delete customizations. You can delete specific customizations, or entire subclasses. (You delete a subclass by deleting all its customizations, an operation that deletes the subclass, renames the Base class, and collapses the hierarchy to the "Before" structure in [Figure 2-1 on page 65](#).)

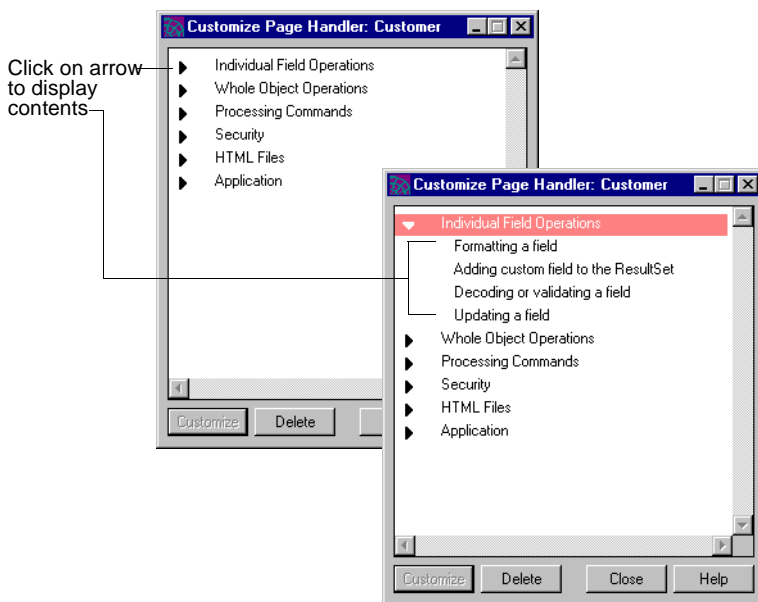
The following sections describe how to use the Page Handler Customization Wizard.

Customizing a Page Handler Class

You use the Page Handler Customization Wizard to customize the page handler classes that underlie the business class pages in your model. When you start the Wizard for a previously uncustomized page handler class, a customizable subclass is created for the page (this action is illustrated in the following procedure).

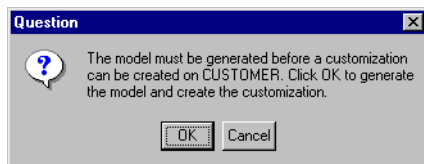
► **To customize a page handler class using the Page Handler Customization Wizard**

1. Select the business class page you wish to customize.
2. Choose the Component > Customize... command.
The Page Handler Customization Wizard appears.
3. Click the arrow to the left of the categories to view specific customizations.



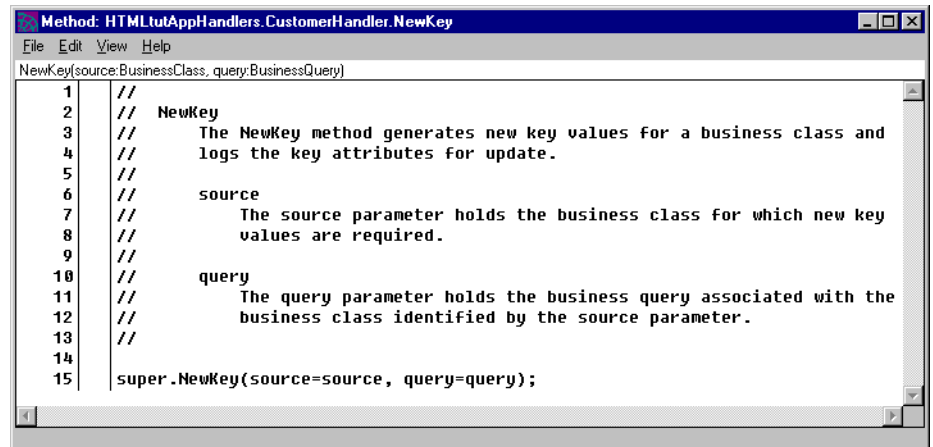
4. (Optional) Select the customization you wish to perform and press the Help key.
A help screen appears as shown in [Figure 2-3 on page 69](#).
5. Double-click the specific customization you wish to make (or click Customize).

If this is the first customization you are making to this class, WebEnterprise Designer displays the following dialog:



6. Click OK.

WebEnterprise Designer automatically expands the class hierarchy to include a customizable class for the selected business class page, and the Method Workshop opens, displaying an override of the appropriate method. The example shown here is from the “Creating a new key” customization point (under the category/subcategory Whole Object Operations/Inserting a new record).



```

Method: HTMLtutAppHandlers.CustomerHandler.NewKey
File Edit View Help
NewKey(source:BusinessClass, query:BusinessQuery)
1 //
2 // NewKey
3 // The NewKey method generates new key values for a business class and
4 // logs the key attributes for update.
5 //
6 // source
7 // The source parameter holds the business class for which new key
8 // values are required.
9 //
10 // query
11 // The query parameter holds the business query associated with the
12 // business class identified by the source parameter.
13 //
14
15 super.NewKey(source=source, query=query);

```

7. Modify or add code appropriate to the customization you wish to make.
8. Choose the File > Compile command to compile the method.
9. (Optional) Return to Step 3 and add more customizations to the class.
10. Close the Method Workshop and the Page Handler Customization Wizard window.

Note that the Page Handler Customization Wizard window is not modal. In other words, you can view customization information about a class, leave the window open and select another window or business class, and the Page Handler Customization Wizard will display the appropriate information for the newly selected class.

Customizing a Generated HTML Template

iPlanet UDS generates the application's HTML templates files (described in [Chapter 3, "Customizing Generated HTML Templates"](#)) into a subdirectory of the document root directory. You can edit these files by opening them with an HTML editor or text editor, or you can use the Page Handler Customization Wizard. The organization of the Wizard's customization points guides you conveniently to the correct location for your customization. For example, find the file you wish to customize in the following Wizard categories:

- HTML Files category—for page mode templates of the selected page
- Application/HTML Files—for application-wide customizations, including customizing the logon page, the start page, or the page responsible for validations

Before you can edit generated files with the Customization Wizard, you must set the FORTE_WW_HTMLEDITOR and FORTE_WW_EDITOR environment variables with the full pathname of the editor of your choice. For example:

```
FORTE_WW_HTMLEDITOR  
c:\PROGRA~1\WINDOW~1\Accessories\wordpad.exe
```

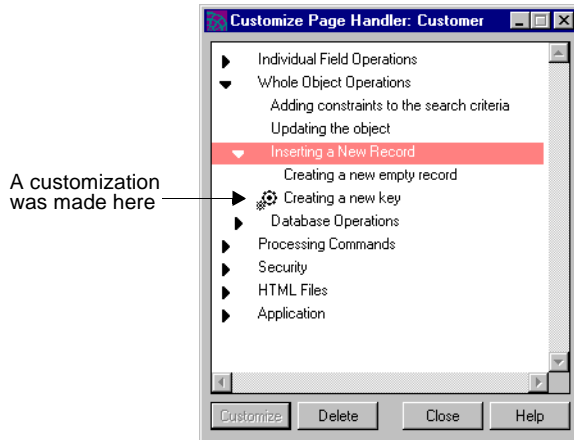
For more information on setting these variables, see the online help topic for the variable. For information on using shortnames for file specification on Windows NT, see the help topic, "Use a shortname for the Default Browser field."

Deleting Customizations

Use the Page Handler Customization Wizard to delete specific customizations or all customizations in a class.

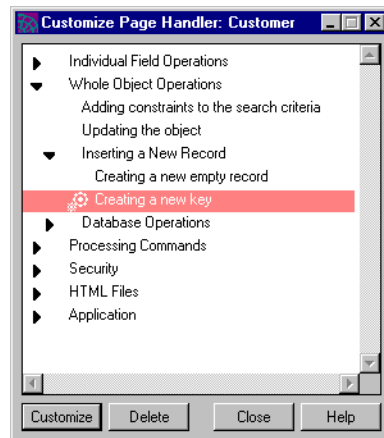
Deleting Specific Customizations

A customization point that has been implemented has a method symbol next to it:

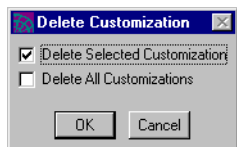


► To delete a customization

1. Select the class that contains the customization and choose Component > Customize... to display the Page Handler Customization Wizard.
2. Select the customization you wish to delete and click the Delete button.



The Delete Customization dialog appears.



3. Check the Delete Selected Customization option and click OK.

WebEnterprise Designer deletes the selected customization.

Deleting All Customizations in a Class

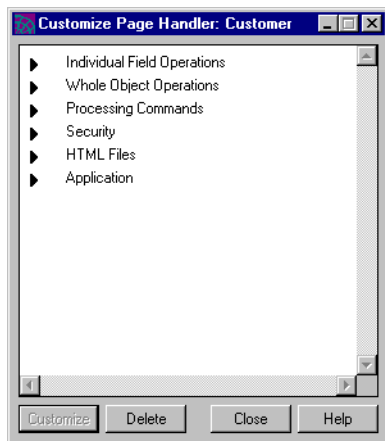
Use the Page Handler Customization Wizard to delete all customizations in a class. When it deletes all customizations, the Customization Wizard also deletes the customizable class, renames the base class to its original name, and collapses the class hierarchy (see the “Before” structure in [Figure 2-1 on page 65](#)).

► To delete all customizations (entire class)

1. If the Always Generate Custom Classes option is enabled, disable it.

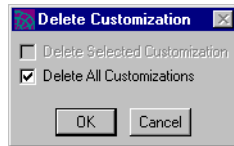
Because the Customization Wizard will delete the customizable class after it deletes all its customizations, you must disable this option so that the Wizard will be able to delete the class.

2. Select the class you wish to delete and choose Component > Customize... to display the Page Handler Customization Wizard.



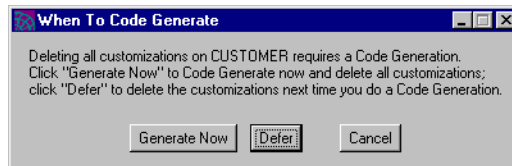
3. Click the Delete button.

The Delete Customization dialog appears, with only the Delete All Customizations toggle active. (This happens only when you click Delete without selecting a specific customization to delete.)



4. Click OK.

The following dialog appears.



5. Click Defer or Generate Now, depending on whether you have more classes to delete (Defer) or not (Generate Now).

Each time you delete all customizations in a class, WebEnterprise Designer regenerates the model. If you plan to delete all customizations on several classes, choose Defer to defer the model regeneration until you specifically request it.

When you choose Generate Now, WebEnterprise Designer removes all customizations (if any) in the leaf-level class, and returns the class hierarchy to its original “Before” structure shown in [Figure 2-1 on page 65](#).

Alternatively, you can defer all deletions and regenerate the model with the File > Generate Web Application Server Code command.

NOTE If you exit the workshop without regenerating the model, you will be prompted that the Delete All Customizations that you specified will not occur. If you ignore this reminder and exit without regenerating, the customized classes will *not* be deleted.

Making Application-Wide Customizations

The Page Handler Customization Wizard has an “Application” customization category. Application customizations are customizations that affect the application as a whole, rather than a specific page in the model, such as logon.

Deleting application customizations When you delete all customizations for business class pages, application customizations are not deleted. You must select each application customization specifically and then delete it.

A Roadmap to Customization Examples

WebEnterprise Designer provides many customization tools to help you customize your application. These are accessible through the Page Handler Customization Wizard.

Customization tools fall into these categories:

- Page Handler Customization Wizard customization points

For the complete list, see “[Page Handler Customization Wizard Customizations](#)” on page 77.

- customization examples

The main customization examples are in [Chapter 7, “Customization Examples,”](#) but other chapters also provide examples, according to their subject. Use the Table of Contents to locate them.

- customization techniques

Use the Table of Contents in this manual to help locate the sections in this chapter that describe the type of customization you wish to make.

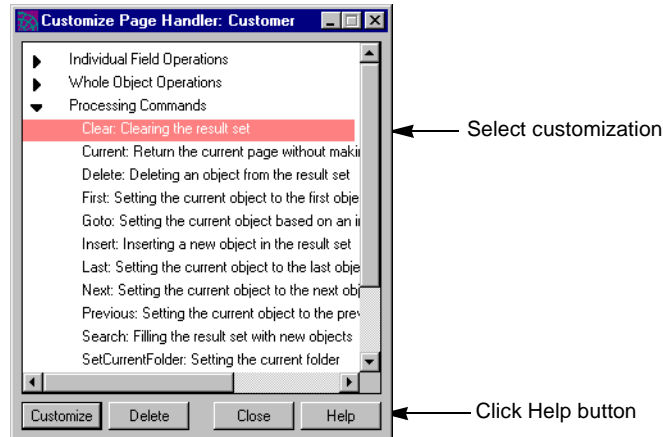
- WebEnterprise Designer example applications

For a complete description of example applications shipped with WebEnterprise Designer, see *Getting Started with WebEnterprise Designer*.

Page Handler Customization Wizard Help Files

At any point while you are using the Page Handler Customization Wizard, you can press the Help key to display online information about the currently selected customization.

Figure 2-4 Page Handler Customization Wizard Help



You can also access these item-specific customization examples directly by their titles in the List of all Page Handler Customizations help topic.

Page Handler Customization Wizard Customizations

The customizations available from the Page Handler Customization Wizard are:

Customization Section	Customization
Individual Field Operations	Formatting a field
	Adding custom field to the ResultSet
	Decoding or validating a field
	Updating a field

Customization Section	Customization
Whole Object Operations	Adding constraints to the search criteria
	Updating the object
	Inserting a new record
	Creating a new empty record
	Creating a new key
Database Operations	Delete: Before sending a delete object request
	Insert: Before sending an insert object request
	Search: Before sending a retrieve object request
	Update: Before sending an update object request
Processing Commands	Clear: Clearing the result set
	Current: Return the current page without making any changes
	Delete: Deleting an object from the result set
	First: Setting the current object to the first object in the result set
	Goto: Setting the current object based on an index
	Insert: Inserting a new object in the result set
	Last: Setting the current object to the last object in the result set
	Next: Setting the current object to the next object in the result set
	Previous: Setting the current object to the previous object in the result set
	Search: Filling the result set with new objects
	SetCurrentFolder: Setting the current folder
	Update: Updating the current object
	Custom Commands

Customization Section	Customization
Security	Restricting access to all page modes
	Restricting access to the search page mode
	Restricting access to the display page mode
	Restricting access to the insert page mode
	Restricting access to the update page mode
HTML Files	Under this category are all HTML file templates for the page
Lookup Files	Under this category are the lookup files for drop list and radio list validation for the page
Logon Validation	This customization point opens the HTTPAccess.LogonSession method for adding security logic to your Logon page
Application	Define HTML template for exceptions
	Define subsidiary applications
	Is subsidiary application
	Modify session timeout
HTML Files	AccessError page (AccessError.html)
	Start page (Start.html)

Customizing Manually

You can perform customizations that do not appear in the predefined list of common customizations in the Page Handler Customization Wizard. To do so, open the project that contains the class you wish to customize and then override the appropriate method or methods in that class. However, before you can customize a class, the class hierarchy must include the customizable leaf-level class. These are subclasses of the superclass whose name contains the word “Base.” These are not automatically generated by default. To create customizable classes, see either [“Creating a Single Customizable Page Handler Class” on page 66](#) or [“Creating a Full Set of Customizable Page Handler Classes” on page 66](#).

Customizable classes are never regenerated, so any changes you make to them will be preserved.

The most common way to customize the behavior of a WebEnterprise Designer application is to override a method inherited from a superclass with TOOL code of your own.

Locating Where to Customize

Making customizations through the Page Handler Customization Wizard automatically places them where you want them. If none of the customization points provided by the Customization Wizard suits your needs, then this section will help you locate common places to add customizations. If the type of customization you want to make is not covered here, follow these suggestions:

- Review the class interaction diagram in [Chapter 1](#) (“[Class Interaction Diagram](#)” on page 39). The interconnections between objects can be useful in figuring out where to make a customization. The class descriptions following each class diagram can also be useful.
- Look through the class function descriptions in [Chapter 1](#) (starting with “[ExpressHTTPAccess](#)” on page 44). If processing similar to what you need is covered in that section, then you will probably see which method you need to override.
- Run your application under the Debugger to just before where you want to change behavior. Set “Method Enter” breakpoints in the Debugger and continue execution of your application. Step or continue from there until you locate the best method to override.

Viewing inherited elements In addition, you can examine inherited class elements to help you determine what and where to customize.

Overriding Methods in a Superclass

Whenever this document instructs you to override a method, you must include a call to the method in the superclass (*super.method*) in your customized code. Omitting this call in most cases will cause the overridden method to fail to perform properly. In rare cases, you must **not** include a call to *super.method*—in these cases, the instructions will point this out explicitly. Customizations provided by the Page Handler Customization Wizard create initial code for the method that has a *super.method* method call, if one is required.

You *override* an existing method by creating a new method in the customizable class (for example CustomerOrderHandler) identical in name, parameters, and return values to the method in the superclass (CustomerOrderBaseHandler). Then, invoke the superclass method (to access its functionality) and add your own custom TOOL code.

When you want to create a method in a subclass that overrides a method defined in a superclass, drag the method from the Class Workshop for the superclass to the Class Workshop for the subclass.

► **To override a method**

1. Choose the View > Inherited command in the Class Workshop to help you find which methods are defined in a superclass.
2. Use the File > Open SuperClass command in the Class Workshop to open the Class Workshop for the superclass where the method is defined.
3. Drag-and-drop the method from the superclass to your customizable subclass.

This will create a method in the subclass with the correct name, parameters, and return type.

4. Open the method in the subclass and delete all its code, replacing it with the single statement:

```
return super.method_name(parameter_list);
```

For example, when overriding a method called NewSelectQuery, which has a parameter named assocID and returns a value, replace the code in the newly created subclass method with the following statement:

```
return super.NewSelectQuery(assocID=assocID);
```

5. Add your custom code in this overridden method.

NOTE Depending on your customization, you could add a call to super after this.

If you were to override a superclass method by dragging the method into the subclass and then modifying the code, you would not need to call super.method. However, do not customize in this manner, because future versions of WebEnterprise Designer may change the implementation of the original method, causing your customized method to fail to compile or to execute improperly. Your call to super.method encapsulates the method's behavior, making WebEnterprise Designer upgrades simpler.

Local and Global Customizations

Most customizations affect a particular business class page and are thus “local” in nature. However, you might need to make some customizations that will affect all business class pages (for example, log certain actions). The coding techniques for making local and global customizations are similar, but special steps must be taken to cause a change to affect all future generated business class pages. These steps are described in “[Global Customization](#)” on page 90.

Error Reporting

You can handle errors and raise exceptions in your customizations as you would in a non-WebEnterprise Designer application (using the `GetTextData` method on the `MsgCatalog` class, the `AddError` method on the `ErrorMgr` class, and so on). See the *TOOL Reference Guide* and the Framework Library online Help for more information.

You will see that classes in the `ExpressHandlers` project make use of the `ExpressHandlerError` class. This class is not intended for you to customize or subclass; you call it directly in your custom code. Most uses of the `ExpressHandlerError` class are in `raise` statements, like the following:

```
-- Raise used by WebEnterprise Designer code,  
-- not intended for you to override.  
raise ExpressHandlerError(originator=self,  
    error=Error.GEN_UNIMPLEMENTED).GetException;
```

In the above example, a new `ExpressHandlerError` object is instantiated and its `GetException` method returns an exception object.

Working with Business Classes

This section discusses some basic concepts about working with business classes that you need to know if you want to change field values and add or remove rows from a result set.

Business Class Record Status

Records in the result set contained in the `ExpressPageData` object contain `InstanceStatus` attributes. The values of these attributes indicate what changes have been made to a record since it was loaded from the database. These status values are used to determine which queries to run on behalf of a record after it has been updated, inserted, or deleted. The following constants, defined in the `BusinessClass` class, describe each numeric value for `InstanceStatus`:

Constant	InstanceStatus Value	Meaning
<code>ST_READONLY</code>	2	Record is read-only and cannot be modified. No Update/Insert/Delete queries will be run on behalf of this record.
<code>ST_READWRITE</code>	4	Record was loaded from the database and is updateable, but has not been changed by the user. Once changed, state will become <code>ST_UPDATE</code> .
<code>ST_UPDATE</code>	8	Record has been modified since being selected from database. Update statement will be run.
<code>ST_INSERT</code>	16	Record is newly created, contains values entered by the user, and is not yet in database. Insert statement will be run.
<code>ST_DELETE</code>	32	Record has been deleted (not yet deleted from database). Delete statement will be run.
<code>ST_EMPTY</code>	1	Empty record to be filled in by user—user has not yet typed any values into the record. (When values are entered, state will become <code>ST_INSERT</code>).

BusinessClass Attribute IDs (ATTR_)

Many methods require an integer parameter to indicate the attribute ID of the business class. The attribute IDs for each business class attribute are generated as constants in class *business_classBaseQuery*. The constants are named the same as the *business_classClass.attribute* name, but with the prefix "ATTR_". For example, the attribute *CustomerClass.CustomerNumber* has a corresponding constant *ATTR_CUSTOMERNUMBER* defined in *CustomerBaseQuery*. (These classes are found in the *business_modelService* project generated from the supplier business model.)

Note that when two business classes contain an identically named field (often it is a database join field), the values of the generated ATTR_ constants in the two BaseQuery classes will not necessarily be identical. For example, do not assume the following have the same value:

```
CustomerBaseQuery.ATTR_CUSTOMERNUMBER
```

```
CustomerOrderBaseQuery.ATTR_CUSTOMERNUMBER
```

Since these constants are inherited, the examples will refer to them through the customizable leaf-level class. For example, the examples will refer to *CustomerQuery.ATTR_NAME*, rather than *CustomerBaseQuery.ATTR_NAME*, where it is defined.

Changing the Value of an Attribute

When you want to change the value of an attribute of a BusinessClass you should use the *UpdateAttr* method of the *ExpressPageHandler* class. When changing the value of an attribute of a BusinessClass there are several things you need to do. This logic is encapsulated in the *UpdateAttr* method.

Checking the Status of a BusinessClass Object

When a BusinessClass object is updated, its UpdateQuery attribute is set. The following example checks whether the BusinessClass object myCust has been updated and if its Address attribute has been modified (the object pageData is of type ExpressPageData):

```

myCust : CustomerClass = pageData.Data[1];

if (myCust <> NIL) then
  if ((myCust.UpdateQuery = NIL)
    or (myCust.UpdateQuery.Values = NIL)) then
    task.Part.LogMgr.PutLine('Business class has not been updated.');
  else
    task.Part.LogMgr.PutLine('Business class has been updated.');
    if (myCust.UpdateQuery.GetUpdateAttr(attr=CUSTOMERQuery.ATTR_ADDRESS)
      = NIL) then
      task.Part.LogMgr.PutLine('Address field has not been updated.');
    else
      task.Part.LogMgr.PutLine('Address field has been updated.');
    end if;
  end if;
end if;

```

Undoing Changes Made to a BusinessClass Object

It is possible to undo changes made to a BusinessClass object by using the BusinessClient's Revert method:

```

myCust : CustomerClass = pageData.Data[1];
if (myCust <> NIL) then
  Client.Revert(myCust);
end if;

```

Customization Techniques: ClassHandler Classes

WebEnterprise Designer provides a set of commands and behavior as a result of your specifications in the HTML Application Model Workshop. However, there may be times when you need to modify the generated behavior.

This section shows examples and describes coding techniques you should employ when customizing handler classes. In these examples, business class page names, class names, and attribute names are used for illustration. You will need to change these to be appropriate for your handler class. Also, the “method” statements below should not be typed into the Method Workshop; they are provided for your information.

Creating a New Instance of a Business Class

You can customize your application to insert data into the result set, but rather than using “new()” to create an object, you should use the `NewClass` or `NewClassForInsert` methods to create an object that is initialized for the current business class page.

For example, if business class page `CustomerOrder` is based on class `CustomerOrderClass`, then create a new instance of `CustomerOrderClass` using the following code in a handler method:

```
record: BusinessClass = NewClass();
```

The new `CustomerOrderClass` object will instantiate and log the attributes used by the associated business class page.

The differences between `NewClass` and `NewClassForInsert` are:

- `NewClass` instantiates attributes used by the associated business class page, `NewClassForInsert` does not
- `NewClassForInsert` uses `NewKey` to assign values to the key attributes of the business class

Getting the Result Set

Use the `GetPageData` method to get the current result set of the Web client issuing the request for the associated business class page.

Getting the Initial Query

Use the `NewQuery` method to get the query required to retrieve all the data that is requested by the associated business class page. This is an unrestrained query; therefore all rows in the database are returned unless constraints are applied on some of the attributes.

Customization Techniques: Business Rules

Business rules are special data requirements that you want your application to enforce automatically, or particular actions the application must perform based on the state of the data.

Where to Implement

You can choose to implement business rules in the following places:

- on the browser
- in the Web application server
- in the business services

If you implement your business rules on the browser, the user gets immediate feedback rather than after submitting the request. However, anything done on the browser can potentially be defeated by the user. Therefore to ensure integrity, you should always implement your business rules in either the Web application server or in the business service. You may want to implement them on the browser as well, to provide a better user interface.

NOTE

Whether you implement a rule in the Web application server or the business service will depend on the nature of the rule. When a rule is applicable to the business service (and could be reused by other applications, for example, enforcing that all user IDs be more than six characters), implement it in the business service. When the rule is specific to one specific application (for example, enforcing that user IDs for the Travel application start with the letter T), implement it in the Web application server. Usually, you will have a combination of rules in both locations.

Business Rules on the Browser

In general, when you implement business rules on the browser, you must custom code them for each HTML template. However, you can avoid this by creating a JavaScript function that gets included and invoked wherever it is needed. The [“Example: Validating a Whole Form” on page 206](#) (in [Chapter 7](#)) demonstrates how to use JavaScript to implement business rules on the browser.

Customization Techniques: Data

This section discusses a variety of customizations that concern processing data from a result set and presenting it on a page.

Formatting Fields

At the end of processing a request, all the data from the user’s result set that is to appear on a displayed HTML page is formatted into a WebEnterprise ResultSet object. To change the formatting of this data, use either of the following:

- Page Handler customization—Formatting a field (Fields category)

See the customization list in [“Page Handler Customization Wizard Customizations” on page 77](#).

- Override handler method—FormatValue

This method is invoked on each field to be displayed. Parameters passed to it include the DataValue object to be displayed and the field’s field index and association ID. For information on field indexes and association IDs, see [“Field Identification” on page 51](#).

Formatting Custom Fields

Custom fields are fields that you have added to the HTML template yourself. The generated application will not know about these fields; therefore, you have to add them to the WebEnterprise ResultSet object yourself. To do this, use either of the following:

- Page Handler customization—Adding data for a custom field to a page (Fields category)
- Override handler method—DisplayPage method

Add your data to the rset parameter. See *A Guide to WebEnterprise* for specifics on how to manipulate a ResultSet object like the rset parameter.

Decoding or Validating Fields

Processing a request from a Search, Update, or Insert form includes decoding the fields on the page that represent business class attributes. To change the format that data is accepted in these fields, or to perform data validation, use either of the following:

- Page Handler customization—Decoding or validating a field (Fields category)
- Override handler method—DecodeValue method

This method is invoked on each field. Parameters passed to it include the TextData object with the user-entered value and the field's field index and association ID. For information on field indexes and association IDs, see [“Field Identification” on page 51](#).

Processing Custom Fields on an HTML Form Submission

The generated application will not know about fields you add to the HTML template yourself. You have to decode such custom fields from the WebEnterprise HttpRequest object yourself.

Processing an Insert or Update Form

To process values specified on an Insert or Update form, use either:

- Page Handler customization—Insert or Update (Database Operations category)
- Override handler method—BeforeInsert or BeforeUpdate methods

Values will be found on the request parameter. See *A Guide to WebEnterprise* for specifics on how to manipulate an HTTPRequest object like the request parameter.

Processing a Search Form

There are two customization points you can use when processing custom fields on a Search form.

If you want the custom fields to affect the generated query, then use either:

- Page Handler customization—Search (Database Operations category)
- Override handler method—BeforeSearch method

To change the way the request is processed, use either:

- Page Handler customization—Search (Actions category)
- Override handler method—DoSearch method

Global Customization

You can add global customizations by subclassing directly from the ExpressHandlers project. Global customizations affect all classes generated from that point forward. You make global customizations when you wish to affect features throughout the application.

You can add global customizations by creating a project that contains subclasses of specific classes in ExpressHandlers (the specific classes are listed below). Using options in the HTML Application Model Workshop, you specify that WebEnterprise Designer use these projects to provide the superclasses for the generated classes.

► **To customize ExpressHandlers classes**

1. In the Repository Workshop, create a new project to contain your customized classes (CustomProject).
2. In the Project Workshop for CustomProject, choose the File > Supplier Plans command and add ExpressHandlers and HTTP as suppliers to the project.
3. Create subclasses of the following superclasses in CustomProject:

ExpressClassHandler
 ExpressHTTPAccess
 ExpressLogonHandler
 ExpressLookupInfo
 ExpressPageData
 ExpressScanner
 ExpressTestRunner
 ExpressValueGenerator

4. Name each subclass with the same custom prefix (for example, “New,” as in NewClassHandler).

Each subclass must use the same suffix as its superclass. You must create subclasses of *all eight* of the above classes, even if you do not plan to customize all of them.

5. Using the Class and Method Workshops, add your customizations to any or all of these classes.
6. In the HTML Application Model Workshop, choose the File > Custom Generation Options command.
7. Enter your custom prefix name (for example, “New” in [Step 4](#)) in the Superclass Prefix for Global Customization property.

This property will initially be set to “Express.”

8. Click OK to apply your change and dismiss the dialog.
9. Choose File > Supplier Plans and add CustomProject as a supplier plan to the model.

Now when you generate code for the HTML application model, WebEnterprise Designer searches the supplier projects of the application and looks for classes with the prefix “New.” The generated HTML model classes will be subclasses of the CustomProject classes.

To use your customizations in future HTML application models, just perform steps 6-9, as described above.

NOTE Supplier plans added to the HTML application model will be added to the supplier plans for the generated project. These supplier plans are not automatically removed from the generated project if you remove the supplier plan as a supplier to the HTML application model. For example, if you change your model supplier plans to pick up customizations from a different project, then you must manually change the supplier plans in the generated project.

Customizing Generated HTML Templates

This chapter provides information on how to customize the HTML files generated from an HTML application model. These files are referred to as HTML templates.

HTML templates are .html files that contain embedded WebEnterprise tags. WebEnterprise Designer processes HTML templates to produce HTML documents displayed in a browser. Processing removes the tags, substitutes actual data values for dynamic data references, and sometimes includes other HTML templates.

Topics covered in this chapter include:

- how WebEnterprise Designer uses HTML templates
- what and where to customize (and what not to customize) in the HTML templates
- how your customizations are preserved after regeneration

At the end of the chapter are two annotated, customized HTML files that illustrate principles of good customization.

How WebEnterprise Designer Uses HTML Templates

When you generate code for an HTML application model, WebEnterprise Designer generates HTML templates into a subdirectory of the document root directory that has the same name as the HTML model. For example, the HTML templates for the HTMLtutApp tutorial reside in the `%{FORTE_ROOT}/html/docs/HTMLtutApp` directory.

The document root directory is the top-most directory of HTML template files used by WebEnterprise Designer, and is specified in a number of ways, the most common of which is with the FORTE_WW_DOCUMENT_ROOT environment variable. For more information on specifying the document root, see [“Setting Document Root on the ScannerService Service Object Partition” on page 259](#).

For each logical page in the HTML application model, WebEnterprise Designer generates a set of related HTML templates to support it. These generated HTML templates are design-specific, but some are common to all designs.

Common Templates

This section describes the HTML templates common to all page designs, organized by different page types, including:

- business class pages
- link pages
- logon pages

Business Class Page Templates

The following table lists the templates common to all business class pages. Templates you generally customize have a “Yes” in the Customize column.

File Name	Customize	Description
Main_bus_class_page.html	Usually not	Top-level HTML definition of the page.
Data_bus_class_page.html	Yes	Provides the tags to be displayed for the page (but not nested data).
Display_bus_class_page.html	No	Display mode of the page, which includes data for this page and nested pages.
Insert_bus_class_page.html	Yes	Insert mode of the page.
Search_bus_class_page.html	Yes	Search mode of the page.
Update_bus_class_page.html	Yes	Update mode of the page.
Scripts_bus_class_page.html	No	Scripts for the page.

The Main template (for example, Main_Customer.html in the HTMLtutApp application) has two main functions:

- it is the template referenced by other pages
When a page calls another page, it references the URL of the Main template. The other templates are considered internal.
- it includes other templates as appropriate for the function being performed
If the function is to display data, the Main template includes the page's Display template. The Display template in turn includes the Data template. If there are nested pages, the Display template includes the Data templates of all nested pages.

The Scripts template defines any JavaScript functions required by the page.

Generated Maintenance Files and Directory

In addition to HTML templates, two text files are generated into the document directory that are used to keep track of important information regarding the generation:

File Name	Customize	Description
manifest.txt	No	Lists each generated file, one per line.
report.txt	No	Describes details of the generation with respect to the customization.

A copy of the entire set of generated files is generated into the .base subdirectory, which maintains these copies throughout successive regenerations. See the section [“Regenerating After Customizing” on page 102](#) for more details concerning the report.txt file and the .base directory.

NOTE You should not modify the files in the .base directory.

Lookup Files

When you specify drop list or radio list validation for a page's field, you can also specify a Lookup class, and then further specify which of the class's attributes is to be the displayed field and which is to be the field where user input is stored (the same attribute can serve as both). (For more information on setting up a drop or radio list, see the Forte online Help.)

For every field that uses validation, WebEnterprise Designer generates a file that contains the legal stored and displayed values:

File Name	Customize	Description
<i>bus_class_page_qq_field_name.inc</i>	Yes	Contains stored and displayed values of the field.

If you specified a lookup class, then the lookup file is automatically populated from the database when the application starts. If you did not specify a Lookup class, then the file is generated without values and must be populated with data externally. In this case, you may use the Page Handler Customization Wizard to edit the file. Be sure to save a copy of this file, because it will be overwritten if you change the model and have to regenerate. Alternatively, you can populate this file at runtime.

For information on how to populate the generated file with data, see [“Drop List or Radio List Example: Entering Lookup Information Manually”](#) on page 217.

Start Page

The final file generated by WebEnterprise Designer is a start page for the application:

File Name	Customize	Description
Start.html	Usually not	Defines a shortcut for a string used in deployment.

The start page allows the administrator to publish a simpler URL for the application.

Link Page Templates

If a link page is defined for the application, WebEnterprise Designer generates the following HTML templates:

File Name	Customize	Description
Main_link_page.html	No	Top-level HTML definition of the page.
Display_link_page.html	Yes	Layout of the data display portion of the page.

Logon Page Templates

If a logon page is defined for the application, WebEnterprise Designer generates the following HTML templates:

File Name	Customize	Description
Logon_logon_page.html	Yes	The page displayed when a non-validated user attempts access to any page in the application.
LogonFailed_logon_page.html	Yes	The page displayed when a user is denied access to the application.
Validate_logon_page.html	No	The page used to validate the logon. This page is never displayed; it redirects to the first page of the application or the logon failed page as appropriate.

Page Design Templates

Different HTML templates are generated for different page designs. This section describes the template sets generated for two of the provided designs: simple and fancy. The other provided page designs (fancyMenu, fancyNoBorder, fancyNoCaption, and fancyNoFooter) are variations on the fancy page design. How they differ from the fancy page design will be made clear in the section on the fancy design.

Simple Page Design Templates

The simple page design creates a single HTML page and supports four modes of operation: search, display, insert, and update. When the current page is to be displayed in one of these modes, the Main template includes the template corresponding to that mode.

Files generated for the Simple page design are:

File Name	Customize	Description
Main_ bus_class_page.html	Usually not	Top-level HTML definition of the page.
Data_ bus_class_page.html	Yes	Provides the tags to be displayed for the page (but not nested data).
Display_ bus_class_page.html	No	Display mode of the page, which includes data for this page and nested pages.
Insert_ bus_class_page.html	Yes	Insert mode of the page.
Search_ bus_class_page.html	Yes	Search mode of the page.
Update_ bus_class_page.html	Yes	Update mode of the page.
Scripts_ bus_class_page.html	No	Scripts for the page.

Fancy Page Design Templates

The fancy page design extends the simple design by using frames to divide the canvas into four sections:

- a header frame at the top
- a display frame in the middle divided into:
 - a menu frame on the left
 - a data frame on the right
- a footer frame at the bottom

The header, footer, and menu are defined statically. The data frame contains dynamic data substituted in at runtime.

Files generated for the fancy design are:

File Name	Customize	Description
Main_bus_class_page.html	Usually not	Top-level HTML definition of the page.
Data_bus_class_page.html	Yes	Provides the tags to be displayed for the page (but not nested data).
Display_bus_class_page.html	No	Display mode of the page, which includes data for this page and nested pages.
Insert_bus_class_page.html	Yes	Insert mode of the page.
Menu_bus_class_page.html	Yes	Menu area for the page.
Search_bus_class_page.html	Yes	Search mode of the page.
Update_bus_class_page.html	Yes	Update mode of the page.
Header_bus_class_page.html	Yes	Header of the page.
Footer_bus_class_page.html	Yes	Footer of the page.
Scripts_bus_class_page.html	No	Scripts for the page.

Fancy Page Design Variations

All the other fancy page designs are based on the files just described, with certain modifications. These designs are:

- fancyMenu has no header or footer templates
The `Header_bus_class_page.html` and `Footer_bus_class_page.html` templates are not included.
- fancyNoBorder has no border around the panes
- fancyNoCaption has no page titles
The fancy page design places a caption in the display pane over the data. The fancyNoCaption page design removes the caption and has only data. This is good for nested pages when the data is self-descriptive and needs no additional label.
- fancyNoFooter has no footer template
The `Footer_bus_class_page.html` template is not included. The logo normally in the footer is moved to the upper right corner of the header.

Customizing HTML Templates

You are free to make changes to the HTML templates generated by WebEnterprise Designer. You might want to customize the generated HTML for a variety of reasons, such as:

- to change the font or other design elements
- to change the layout or containment hierarchy
- to add graphics or other HTML elements

WebEnterprise Designer allows you to customize any of the generated HTML templates; it preserves customizations during any subsequent generation of the HTML application model. This section describes how this process works, and gives guidance as to how to avoid damaging your HTML templates through customization.

NOTE If your customizations are limited to font and style changes, you should first look at creating custom page designs or styles, which is usually easier and more reusable. See [Chapter 4, “Customizing Page Designs”](#) and [Chapter 5, “Customizing Page Styles,”](#) for more information.

Customization Types

You can customize HTML templates for a single application or for many applications.

Single application When a specific change is required for an individual application, you generate the HTML templates and then customize one or more of them. This type of customization is the subject of this chapter.

Multiple applications When a change is required throughout one or more applications, you effectively create a custom page design. This type of customization is the subject of [Chapter 4, “Customizing Page Designs.”](#)

Where to Customize

In general, when you are customizing the Simple or Fancy page design, you will customize one of the page mode templates (data, search, insert, or update).

Because there are separate templates for each mode of the logical page, it is possible to customize only a single mode. An example of this type of customization is when you do not want some field validations applied when the user is entering search criteria, but you do want them when the user is doing an insert or update. In this case, you would customize by removing the unwanted field validations from the page's Search template.

If, on the other hand, you want to make changes in a single logical page, you must make the same changes to all the templates in the set.

You can customize the Main and Display templates, but there is usually no need to do so. An exception to this rule is when you want to affect the frame definition in the Fancy design (for example, by changing where the frame boundaries are, removing the header or the footer, and so forth). In that case, you might want to customize the Main template.

You might also want to customize the Header, Footer, and Menu templates of the Fancy design.

Within a template you can add, remove, or modify almost any of the normal HTML tags. The next section, *“What Not to Customize,”* offers guidelines on what to customize.

What Not to Customize

There are some items you must leave alone when customizing HTML templates. WebEnterprise Designer templates are a combination of HTML tags, WebEnterprise tags, and plain text. The following table provides guidelines on customizing these three categories:

Type of Item	Modify or Add	Move	Remove
HTML tags	Always OK	Usually OK	Usually OK
Plain text	Always OK	Usually OK	Usually OK
WebEnterprise tags	Never OK	Never OK	Never OK
WebEnterprise variables	Add OK	OK	OK

WebEnterprise tags WebEnterprise tags are HTML-like tags that start with “<?forte ...” and are usually embedded in an HTML comment, as in:

```
<!--<?forte iterate listentry entry.list_Movie start="1" max="1">-->
```

NOTE In this manual, as in *A Guide to WebEnterprise*, the tags are referred to in text in abbreviated form (for example, “the FORTE EXECUTE tag”), while the actual syntax requires a preceding question mark (as above). For a full description of Forte HTML tag syntax, see *A Guide to WebEnterprise*

WebEnterprise variables WebEnterprise variables are plain text starting with “\$\$” and usually enclosed by parentheses, as in:

```
$$ (listentry.Movie_qq_MovieID)
```

WebEnterprise variables are described in “[WebEnterprise Designer HTML Template Elements](#)” on page 56.

Hidden fields When customizing HTML templates, you must never remove or modify any hidden fields on forms. Hidden fields are identified with “HIDDEN” for input type, such as:

```
<input type="HIDDEN" name="ServiceName" value="movieService">
```

Regenerating After Customizing

You will usually regenerate the HTML application model after you have made customizations to the HTML templates. When you do this, new HTML templates are generated. Your customizations are preserved in one of three ways:

- by keeping the customized file in preference to overwriting the newly-generated one
- by merging the customized file with the newly-generated one (when this can be done without conflicts)
- by merging the customized file with the newly-generated one, but leaving the customization commented out

Forte reports on what happened to each file during the regeneration by means of the report.txt file. The following scenarios describe the details of this process.

Scenario 1: HTML Changes Only

Consider the circumstance in which you have completely finished your model and decide to change one or more HTML templates. Perhaps you wish to change the background color of all the templates, or you might want to use a different font for several labels.

HTML template change Changes to colors, fonts, and so forth, on certain templates

HTML model change No changes that affect those pages

HTML code generation change Forte compares the newly-generated pages corresponding to the changed pages to the equivalent pages in the Base Level directory (the .base directory). It finds no differences between the new pages and the Base Level page equivalents. It leaves the Top Level pages unchanged (the HTML changes are preserved). The template is listed by name in the report.txt file, along with any other templates that fall into this category, following the heading:

NOTE: The following files had user customizations but no model changes. The user customizations have been left. No changes are needed.

In addition, the count of files treated in this way will be listed at the end of the report.txt file.

Scenario 2: HTML and Model Changes

Now consider the circumstance in which you have changed one or more HTML templates, and now you change the model.

HTML template change Changes to colors, fonts, and so forth, on certain pages

HTML model change Attributes added or removed, links added or removed, both field and command labels edited

HTML code generation change Forte compares and finds differences between the new templates and equivalent templates in both the Base Level directory, and detects user modifications in the Top Level files. Forte merges the new model changes with the HTML changes. Base Level files are updated to reflect the new HTML model. The template is listed by name in the report.txt file, along with any other templates that fall into this category, following the heading:

```
WARNING: The following files had user customizations and model changes
but were merged without conflicts. The previously generated file (i.e.,
the root ancestor) is left in <fileName>.old. The user customized
file based upon the previous generation is left in <fileName>.cust. The
generated file from the current generation before merging is left in
<fileName>.gen. The merged file is left in <fileName>.html. No changes
are needed but you may wish to inspect the file to confirm that the
merge produced the desired result.
```

In addition, the count of files treated in this way are listed at the end of the report.txt file. Pre-merged files are preserved and the following files are generated:

- *htmlTemplate.old* files—old Base Level files (not merged)
- *htmlTemplate.cust* files—old Top Level files (with HTML customizations, not merged)
- *htmlTemplate.gen* files—Newly-generated files (not merged)
- *htmlTemplate.html* files—Newly-generated files with user customizations merged in

Scenario 3: Conflicting HTML and Model Changes

Now consider the circumstance in which you have changed one or more HTML templates, and also changed the model in some conflicting way. For example, perhaps you changed all Part Number labels to “Part No” in the model, but to “PN” in the HTML files.

HTML template change Changes to labels

HTML model change Changes to labels, but different from the HTML changes

HTML code generation change Forte compares and finds differences between the new templates and equivalent Base Level templates and detects user modifications in the Top Level templates. Forte attempts to merge the files, but because there are conflicting changes, the merge results in an error. The template is listed by name in the report.txt file, along with any other templates that fall into this category, following the heading:

```
ERROR: The following files had user customizations and model changes
that were merged with conflicts. These files will need to be reconciled.
The previously generated file (i.e. the root ancestor) is left in
<fileName>.old. the user customized file based upon the previous
generation is left in <fileName>.cust. The generated file from the
current generation before merging is left in <fileName>.gen. The
merged file is left in <fileName>.html with comments surrounding the
conflicting lines. These conflicts will need to be resolved manually.
```

The count of files treated in this way are listed at the end of the report.txt file, and the same files are generated as listed in **“Scenario 2: HTML and Model Changes.”**

After generation an error dialog alerts the user to the presence of merge conflicts and directs the user to the report.txt file. The conflicts are indicated in the .html file with blocks of text as shown below:

```
<!-- >>>>> BEGIN CONFLICT BLOCK -- GENERATED TEXT: -->
... generated information ...
<!-- ===== CUSTOMIZED TEXT: (left as comment)
... example of user-entered information ...
<!-- <<<<< END CONFLICT BLOCK -->
```

The first comment line points out the conflict block. The generated text follows as regular HTML. The second comment line introduces a comment block that ends only the end of the user-customized block (that is, the user-customized HTML is left in the file as a comment).

Customization Examples

This section contains two examples that illustrate where to put certain customizations, and what you should not modify or, in some cases, delete or even move.

The examples are:

- [“Example: Customizing a Field on a Search Page,”](#) which follows
This customization shows how to remove a JavaScript validation from a Search page and replace it with a simple input field. This example is fully described in [Chapter 7, “Customization Examples.”](#)
- [“Example: Customizing a Font Size on a Data Page”](#) on page 109
This customization shows how to make the font of a field on a Data page two points and bold.

Conventions Used with the Examples

In these examples, the following conventions are followed:

- **bolded** lines are customized lines
- unshaded lines are lines that should not be modified

Unless otherwise noted, you can delete or comment out the unshaded lines (or move the whole block containing the line), but you should not edit the text of the code.

For a description of WebEnterprise variables and HTML URL parameters and links, see [“WebEnterprise Designer HTML Template Elements”](#) on page 56.

Example: Customizing a Field on a Search Page

This example shows a sample Search page from the tutorial application (HTMLtutApp), which was customized to apply an IsAlphabetic validation to the Customer Name field of the Customer page. This validation was inconvenient for the Search page, where the user might want to use a wildcard. Therefore, on this Search page, the validation has been commented out and a simple input field inserted.

This customization is described in “[Example: Removing a JavaScript Validation from a Page Mode](#)” on page 198.

```

<!-- Forte WebEnterprise Designer Search form definition for model
      HTMLtutApp using page design fancy and style cool.  Generated on
      20-Feb-1999 13:05:14 -->
<!--<?forte if CustomerHandler.RestrictSearchAccess>-->
<!--<?forte redirect "HTMLtutApp/AccessError.html&PageName=Customer
      &PageMode=Search">-->
<!--<?/forte if>-->
<html>
  <head>
    <title>
      Search Customer
    </title>
    <link href="/forte/styles/cool.css" rel="stylesheet" type="text/css">
  </head>

  [Do not move or delete the following line.]
  <!--<?forte include "HTMLtutApp/Scripts_Customer.html">-->
  <body class="modify">
    <caption align="left">
      <div class="captionform">
        Find Customer
      </div>
    </caption>

  [Do not move or edit the following three lines.]
  <form method="POST" action="$$ (FORTE.ExecURL) ">
    <input type="HIDDEN" name="ServiceName" value="HTMLtutAppService">
    <input type="HIDDEN" name="TemplateName" value="$$ (FORTE.ReturnTemplate) ">
    <input type="HIDDEN" name="Action" value="CustomerHandler.Search">
    <br>
    <table border="0" cellspacing="3" cellpadding="5">
      <tr>

        <th class="labelmodify" style="width:20%">
          Customer Number
        </th>
        <td class="dataform" style="width:80%">
          <input type="text" name="Customer_qq_CustomerNumber" size=40>
        </td>

      </tr>
      <tr>

        <th class="labelmodify" style="width:20%">
          Name
        </th>
        <td class="dataform" style="width:80%">

  [Start comment.]
    <!--

```

```

[Call to JavaScript validation.]
<script language="JavaScript">
Select_Customer_qq_Name('Customer_qq_Name','Name','qqNone');
</script>

[End comment.]
-->

[Add input field.]
<input type="text" name="Customer_qq_Name" size=40>
</td>

</tr>
<tr>

<th class="labelmodify" style="width:20%">
Address
</th>
<td class="dataform" style="width:80%">
<input type="text" name="Customer_qq_Address" size=40>
</td>

</tr>
<tr>

<th class="labelmodify" style="width:20%">
Phone
</th>
<td class="dataform" style="width:80%">
<input type="text" name="Customer_qq_Phone" size=40>
</td>

</tr>
</table>
<br>
<br>
<table border="0" width="1" cellspacing="3" cellpadding="5">
<tr>
<td class="buttons">
<INPUT TYPE="submit" VALUE="SEARCH">
</td>
</tr>
</table>
</form>
</body>
</html>

```

Example: Customizing a Font Size on a Data Page

This next example is a sample Data page from the Movie application provided in the WebEnterprise examples directory. This example shows how you would make the Movie field a larger font than the default and also bold, rather than regular.

```

<!-- Forte WebEnterprise Designer Display form definition for model
movieApp using page design fancy and style cool. Generated on
16-Feb-1999 22:50:51 -->

[Do not move the following line.]
<!--<?forte iterate listentry entry.list_Movie start="1" max="1">-->

[Begin data display.]
<table width="100%" border="0" cellspacing="3" cellpadding="5">
  <caption align="left">
    <div class="captionform">
      Movie
    </div>
  </caption>
  <tr>

[Start definition of Movie field.]
    <th class="labelform" style="width:20%">
      MovieID
    </th>
    <td class="dataform" style="width:80%">

[Begin modification.]
      <font size="+2"> <bold>
        $$ (listentry.Movie_qq_MovieID)

[End modification.]
      </bold> </font>
    </td>

[End definition of Movie field.]
  </tr>

[Start definition of Title field.]
  <tr>

    <th class="labelform" style="width:20%">
      Title
    </th>
    <td class="dataform" style="width:80%">
      <bold>
        $$ (listentry.Movie_qq_Title)
      </td>

[End definition of Title field.]
  </tr>

```

[Start definition of Rating field.]

```
<tr>
    <th class="labelform" style="width:20%">
        Rating
    </th>
    <td class="dataform" style="width:80%">
        $$ (listentry.Movie_qq_MPRating)
    </td>
```

[End definition of Rating field.]

```
</tr>
```

[Start Critic Rating field.]

```
<tr>
    <th class="labelform" style="width:20%">
        Critic Rating
    </th>
    <td class="dataform" style="width:80%">
        $$ (listentry.Movie_qq_CriticRating)
    </td>
```

[End Critic Rating Field.]

```
</tr>
```

[Start Description field.]

```
<tr>
    <th class="labelform" style="width:20%">
        Description
    </th>
    <td class="dataform" style="width:80%">
        $$ (listentry.Movie_qq_Description)
    </td>
```

[End Description field.]

```
</tr>
```

[Start Web Address field.]

```
<tr>
    <th class="labelform" style="width:20%">
        Web Address
    </th>
    <td class="dataform" style="width:80%">
        $$ (listentry.Movie_qq_webaddress)
    </td>
```

[End Web Address field.]

```
</tr>
```

[End data display table.]

```
</table>
```

```

[Do not move the following line.]
<?forte if $$ (USER.TopPage)Handler.IsNestedPage(PageName="Movie")>

[Begin commands for nested page (because they can not appear in the command panel of the main page).]
  <table width="40%" align="center" border="0" cellspacing="3"

      cellpadding="5">
    <tr>
      <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

[Begin Search command.]
    <td>
      <a class="alink"
    HREF="$$ (FORTE.ExecURL)?ServiceName=movieAppService&TemplateName=movieApp/Sear
    ch_Movie.html&ReturnTemplate=movieApp/Display_$$ (USER.TopPage).html&Uniquifier
    =$$ (FORTE.UniqueID)" target="_fortedisplay">
        Search</a>

[End Search command.]
    </td>
    <br>
    <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

[Begin Update command.]
    <td>
      <?forte if MovieHandler.HasCurrentRow>
        <a class="alink"
    HREF="$$ (FORTE.ExecURL)?ServiceName=movieAppService&TemplateName=movieApp/Upda
    te_Movie.html&ReturnTemplate=movieApp/Display_$$ (USER.TopPage).html&Uniquifier
    =$$ (FORTE.UniqueID)&Action=MovieHandler.Current" target="_fortedisplay">
        Update</a>
      <?forte else>
        Update
      <?forte if>

[End Update command.]
    </td>
    <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

[Begin Insert command.]
    <td>
      <a class="alink"
    HREF="$$ (FORTE.ExecURL)?ServiceName=movieAppService&TemplateName=movieApp/Inse
    rt_Movie.html&ReturnTemplate=movieApp/Display_$$ (USER.TopPage).html&Uniquifier
    =$$ (FORTE.UniqueID)" target="_fortedisplay">
        Insert</a>

[End Insert command.]
    </td>
    <td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

```

[Begin Delete command.]

```

<td>
  <?forte if MovieHandler.HasCurrentRow>
    <script language="JavaScript">
      function confirm_delete(that,url,action)
      {
        ret = confirm("Delete this record?");
        if (ret)
          that.href = url + action;
        else {
          that.href = url;
        }
      }
    </script>
    <a class="alink" HREF="#1"
NAME="1"onclick="confirm_delete(this, '$$(FORTE.ExecURL)?ServiceName=movieAppSe
rvice&TemplateName=movieApp/Display_Movie.html&ReturnTemplate=movieApp/Display
_$$$(USER.TopPage).html&Uniquifier=$$$(FORTE.UniqueID)', '&=ActionMovieHandler.De
lete' )" target="_fortedisplay">
      Delete</a>
    <?forte else>
      Delete
    <?forte if>

```

[End Delete command.]

```

</td>
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

```

[Begin First command.]

```

<td>
  <a class="alink"
HREF="$$$(FORTE.ExecURL)?ServiceName=movieAppService&TemplateName=movieApp/Disp
lay_$$$(USER.TopPage).html&Action=MovieHandler.First&Uniquifier=$$$(FORTE.Unique
ID)"target="_fortedisplay">
    First </a>

```

[End First command.]

```

</td>
<td>&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>

```

[Begin Previous command.]

```

<td>
  <a class="alink"
HREF="$$$(FORTE.ExecURL)?ServiceName=movieAppService&TemplateName=movieApp/Disp
lay_$$$(USER.TopPage).html&Action=MovieHandler.Previous&Uniquifier=$$$(FORTE.Uni
queID)" target="_fortedisplay">
    Previous </a>

```


[Displays requested row.]

```
<div id="reqstatus">$$ (entry.RequestStatus_Movie)</div>
</th>
</tr>
</table>
```

[Do not move the following line.]

```
<!--<?/forte iterate listentry>-->
```

Customizing Page Designs

WebEnterprise Designer contains a set of page designs—simple, fancy, and several variations of fancy—that represent several common Web page layouts. You use these to determine the layout of pages in your WebEnterprise Designer applications.

This chapter provides information on WebEnterprise Designer page designs and how to customize them, including:

- the internal structure of page designs
- strategies and considerations for customizing page designs
- a customization example

For information on WebEnterprise Designer styles and how to customize them, see [Chapter 5, “Customizing Page Styles.”](#)

About Page Designs, Templates, and Pages

The production of a Web page by WebEnterprise Designer involves a number of steps and several intermediate representations of the page. The ultimate source that determines the layout the Web page when it is displayed in a browser is the *page design*.

Page designs determine:

- whether HTML frames are used
- whether headers and footers are displayed
- how navigation menus are represented
- the basic structure of all documents

The scope of a page design is the entire iPlanet UDS installation. Changes to a design affect every HTML application that is subsequently generated from it. Designs reside in subdirectories of the Express installation, under the `${FORTE_ROOT}/userapp/express/clx/designs` directory.

Page Designs and Web Page Production

WebEnterprise Designer uses the following steps to render an HTML page:

1. When the page is defined in the HTML Application Model Workshop it is associated with a page design.

This design specification is fundamentally a pointer to a directory. For example, when you specify that the application will use the simple design, this directs WebEnterprise Designer to use the files in the `${FORTE_ROOT}/userapp/express/clx/designs/simple` directory for generation.

2. When you generate code for the HTML Application, the metadata for each page in the model, as well as the model itself, is input to the HTML Template Generator.

The metadata, as defined in the HTML Application Model, includes attributes that the generator uses to determine:

- o which HTML templates to generate
- o the effect of Code Generation Directives within the design files

The output of this phase is a set of application-specific HTML templates stored under the HTML document root directory. (For information on the HTML document root and its setting, see [“Setting Document Root on the ScannerService Service Object Partition” on page 259.](#))

3. When you run the HTML Application, the client requests a particular HTML template, which the WebEnterprise runtime system reads and scans for WebEnterprise template directives.

The directives represent both methods to be called and references to dynamic application data. WebEnterprise resolves these references, substitutes data into the template, and returns a “pure HTML” document to the Web server and the user’s browser screen.

When to Customize Page Designs

Sometimes you might want to customize an application's generated HTML template. Such a customization affects only that one page in that particular application. This type of customization is covered in **Chapter 3, "Customizing Generated HTML Templates."** If, however, your company wants all of its Web applications to possess a similar structure and "look and feel," you can best accomplish this with WebEnterprise Designer by creating a custom page design, to be used as the basis for pages in all applications.

Creating a custom design is, however, not a trivial task. Designs contain a mixture of code generation directives, WebEnterprise template directives, and HTML. You must take care to ensure that changes to page designs result in valid HTML pages. This chapter offers guidelines and examples that will help you create custom designs.

Page Design Elements and HTML Template Generation

When you generate an HTML application, iPlanet UDS first generates the TOOL project and its contents, and then creates the required HTML templates. During the generation of HTML templates, the iPlanet UDS generator processes each component of the HTML model sequentially, in the following order:

- the model object itself
- each page defined in the model
- any input validations or output-formatting JavaScript procedures

Design files are the main tools used to process the pages of the HTML model. Not only do the contents of the files direct the processing, but the file names themselves play a role in the process.

This section begins with a description of the elements of page design file names, followed by a description of the functions of code generation directives that are the contents of the files.

Page Design File Names and Selectors

Design file names (that is, the names of files that reside in the specific design directory) play a crucial role in HTML template generation. Design file names have two important functions:

- determining which design files are selected for generation
- defining the names of the generated HTML template files

For a description of generated templates, refer to [“How WebEnterprise Designer Uses HTML Templates” on page 93](#).

The best way to understand the design filename structure is to use an example, in this case, the fancy page design files, found in `${FORTE_ROOT}/userapp/express/clx/designs/fancy`. The following sections describe each file in this directory, including descriptions of the different elements of the filenames. These elements are called *selectors*, because they select elements of the application.

`_RC_IsAabModel_Start.html`

`_RC_IsAabModel_AccessError.html`

These files select the Start and AccessError pages for processing. The selectors are:

Selector	Use
IsAabModel	Refers to the entire model and selects the file for generation.
RC	Refers to one component (rather than an array of components to be iterated); thus, every generated model results in a <code>Start.html</code> and <code>AccessError.html</code> template file.
Start	Selects the Start page, which is the page users normally request when they enter the application, and which redirects them to the application’s starting page, as defined in the model.
AccessError	Selects the page that is returned if the application is customized to restrict access to certain pages and that access is denied. For information on customizing security in HTML applications, see Chapter 8, “Customizing Application Security.”

_RC_IsFormWindow_Data__RR_Name_.html

_RC_IsListWindow_Data__RR_Name_.html

These files select each form and list page in the application for processing. The selectors are:

Selector	Use
<i>_RC_</i>	Refers to one component (rather than an array of components to be iterated).
<i>IsFormWindow</i>	Selects the form pages in the model.
<i>IsListWindow</i>	Selects the list pages in the model.
<i>Data</i>	Selects the Data template, which governs the layout and display of a page's data, whether the page is the only page displayed, a master page, or a detail page.
<i>_RR_Name_</i>	Is replaced by the name of the page in the model, for example, a list page named Customer generates a template file named <i>Data_Customer.html</i> .

_RC_IsDataWindow_Main__RR_Name_.html

_RC_IsDataWindow_Header__RR_Name_.html

_RC_IsDataWindow_Footer__RR_Name_.html

_RC_IsDataWindow_Menu__RR_Name_.html

_RC_IsDataWindow_Display__RR_Name_.html

_RC_IsDataWindow_Search__RR_Name_.html

_RC_IsDataWindow_Insert__RR_Name_.html

_RC_IsDataWindow_Update__RR_Name_.html

These files select all data pages for processing. The selectors are:

Selector	Use
<i>_RC_</i>	Refers to one component (rather than an array of components to be iterated).
<i>IsDataWindow</i>	refers to Data pages. Each Data page (which includes both form and list pages) generates templates from each of these design files.
<i>_Main_</i>	The fancy design employs HTML frames in which the Main file defines the frame layout.
<i>_Header_</i>	Defines the frame positioned across the top of the page.

Selector	Use
<code>_Footer_</code>	Defines the frame positioned across the bottom of the page.
<code>_Menu_</code>	Defines the frame on the left side of the page.
<code>_Display_</code>	Defines the large center frame, depending upon the page mode.
<code>_Search_</code>	
<code>_Insert_</code>	
<code>_Update_</code>	
<code>_RR_Name_</code>	Is replaced by the name of the page in the model, for example, a list page named Customer generates a template file named <code>Menu_Customer.html</code> .

`_RC_IsLinkWindow_Main__RR_Name_.html`

`_RC_IsLinkWindow_Display__RR_Name_.html`

These files select link pages for processing. The selectors are:

Selector	Use
<code>_RC_</code>	Refers to one component (rather than an array of components to be iterated).
<code>IsLinkWindow</code>	Selects link pages. Each link page generates two HTML templates.
<code>_Display_</code>	Defines the page that displays the link page (for example, the <code>Display_Home.html</code> template for the Home page in the HTMLtutApp tutorial).
<code>_Main_</code>	Defines a shorthand page that simply includes the Display page (in the same application, the <code>Main_Home.html</code> template).
<code>_RR_Name_</code>	Is replaced by the name of the page in the model, for example, a list page named Customer generates a template file named <code>Main_Customer.html</code> .

_RC_HasJavaScripts_Scripts__RR_Name_.html

This file defines the scripts file. The selectors are:

Selector	Use
<u>_RC_</u>	Refers to one component (rather than an array of components to be iterated).
HasJavaScripts	Selects each page that has a JavaScript script defined for it. There are two types of scripts: input validation scripts and output-formatting scripts.
Scripts	Defines the page that displays the link page (for example, the <code>Display_Home.html</code> template for the Home page in the HTMLtutApp tutorial).
<u>_RR_Name_</u>	Is replaced by the name of the page in the model, for example, <code>Scripts_Customer.html</code> . The specified JavaScript files are included by the <code>Scripts_Customer.html</code> file.

_RI_LookupFields_UniqueName_.inc

The file defines input validation lookup fields. Lookup template files are generated for each input validation lookup field defined in the application. The selectors are:

Selector	Use
<u>_RI_</u>	Refers to an array of components. It is iterated, with each element's <code>_UniqueName_</code> attribute forming an output file.
LookupFields	Refers to the lookup attributes defined for the validation.
<u>_UniqueName_</u>	Refers to the unique naming scheme of the lookup field's file name (<i>pagename_qq_field</i>).

NOTE The files produced by code generation are merely “stubs,” without any of the actual Express Business Class data. When the application is actually executed, the database is queried and the actual lookup files are created and written to the application's HTML template directory.

Page Design Code Generation Processing

Page design files contain a mixture of code generation directives, WebEnterprise HTML template tags, and standard HTML V4.0 tags. When you generate code for an HTML application, the code generator selects and processes design files (as described in [“Page Design File Names and Selectors” on page 118](#)). This processing involves two activities that result in an HTML template:

- code generation attribute references are replaced with the actual values of those attributes
- code generation conditional and looping directives are executed

All code generation attributes and directives are enclosed within a pair of “curly braces” (for example, `{{attributes-and-directives}}`). The code generator scans the page design file for these and then processes the enclosed directive. The code generation processor essentially has access to all attributes and properties that are defined within the HTML application model. For example, consider the following lines in the `_RC_IsDataWindow_Display__RR_Name.html` file:

```
<title>
  Display {{%R(page)[%P(Name)]}}
</title>
```

These lines direct the code generator to fetch (%R) the current page object and print (%P) its Name attribute. The result could be:

```
<title>
  Display Movies
</title>
```

This example illustrates the simplest of code generation directives; many are much more involved. The most powerful directives are those involving conditional and looping logic. These directives span multiple lines and, in the case of page design files, surround blocks of HTML template tags, HTML tags, and other code generation directives.

Consider this block from `_RC_IsListWindow_Data__RR_Name.html`:

```
<!--{{F(NestedLinks)}}-->
<center>
  <table width="80%" cellspacing="3" cellpadding="5" align="CENTER">
    <!--<?Forte include
  "{{R(page)[E(Parent)[P(Name)]]}}/Data_{{E(Node)[P(Name)]}}.html">-->
    </table>
  </center>
<!--{ }-->
```

This code block contains the following notable items:

- The code generation directives are enclosed within standard HTML 4.0 comments (that is, “<!-- comment -->”).
This means that page design files can be edited with standard HTML editors.
- The “%F(NestedLinks)[” directive on the first line actually is not completed until the closing “]” on the last line.

The effect of this block is to iterate through all (if any) of the nested links defined for the ListWindow, reproducing the lines between the <center> and </center> HTML tags.

The generated HTML template block might be:

```
<center>
  <table width="80%" cellspacing="3" cellpadding="5" align="CENTER">
    <!--<?Forte include "Movie/Data_Showings.html">-->
  </table>
</center>
<center>
  <table width="80%" cellspacing="3" cellpadding="5" align="CENTER">
    <!--<?Forte include "Movie/Data_Reviews.html">-->
  </table>
</center>
```

Guidelines for Customizing Code Generation Directives

The general rule is: do not customize code generation directives! Code generation directives are complex entities and not easily modified. However, if you understand the purpose of a code generation directive, it may be possible to move or delete it. Take great care to identify the entire scope of a directive and deal with it as a whole, rather than treating a block within curly-braces as a unit.

If your customization requires change or movement of code generation directives, the best practice is to make incremental changes, regenerate the application, compare the new HTML templates with the old and, if acceptable, continue.

Example: Customizing a Page Design

In page design customizations, the most important consideration is whether the design will use HTML frames and, if so, what that frame layout will be. Begin your customization by selecting the existing design that most closely matches the new design you wish to create. If your design is not based on frames, use the simple design as a starting point. If your design is to be frame-based, use fancy or one of its variants.

This section illustrates techniques for creating customized page designs with an example that creates a design that does not use frames, but achieves a framelike appearance by using HTML tables. This design consists of two panes: one for the display of data, the other for a menu. We will name the design “brilliant.”

The remainder of this chapter describes how to create the brilliant page design. We will not describe every detail of the complete customization, but only the first major part of it, and then give guidelines for the rest. We will generate the HTMLtutApp tutorial application with the newly-created brilliant design.

► The general steps for creating a customized page design are

1. Create a new design directory for the design files and populate it with copies of files of the closest style.
2. Create the bitmap and text that identify and describe the design in the HTML Application Model Wizard.
3. Clear the Document Directory of all existing generated HTML templates.

This is to prevent the code generator from merging the customizations with existing designs.

4. Make the necessary customizations.

In this simple example, we describe how to customize the display of form pages.

5. Generate the HTML and inspect it for errors.
6. Fine-tune the customized design.

Create a New Design Directory

This section describes how to create a directory for the new design and how to populate it.

► To create and populate a new design directory

1. Create a brilliant directory under the `${FORTE_ROOT}/userapp/express/clx/designs` directory.
2. Copy all the files in `${FORTE_ROOT}/userapp/express/clx/designs/simple` to the new brilliant directory.

Use the simple directory files as a starting point, because simple is not frame-based, and therefore closest of all the available designs to our proposed design.

Identify the Design with a Bitmap and Text

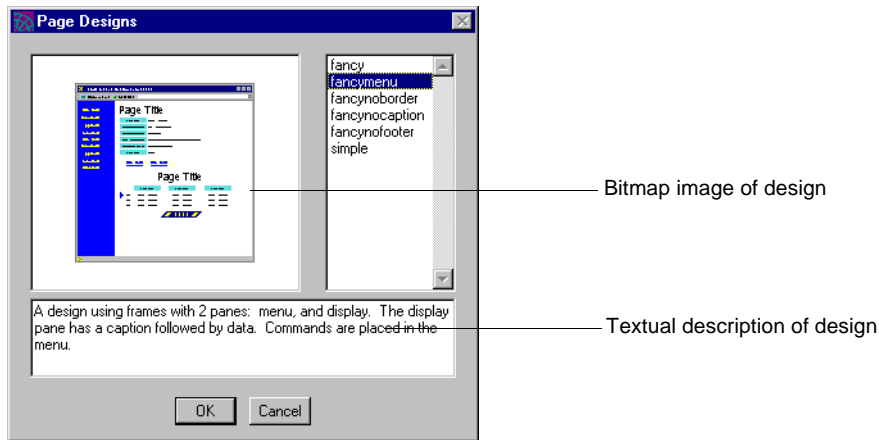
Two files within the design directory serve to identify the design in the Page Design Properties dialog:

- a bitmap image of a thumbnail picture of a page created with the design
- a brief textual description of the design

(You access the Page Designs properties dialog by clicking the Page Design browser button on either the HTML Application Properties dialog or on the Page Options page of the Page Wizard.)

For our example, even though the definition of the simple design is closest to the brilliant design, we want it to look more like the fancyMenu design. Therefore, we will use the fancyMenu bitmap for our image identifier. The identifiers of the fancyMenu design are shown in [Figure 4-1](#):

Figure 4-1 Page Design Identifiers



➤ **To identify a page design**

1. Copy the `fancyMenu.bmp` file from the `fancyMenu` directory to the `brilliant` directory.
2. Rename the `fancyMenu.bmp` file `brilliant.bmp`.
3. Create a text file named `brilliant.txt` in the `brilliant` directory.
4. Add the following text to the `brilliant.txt` file:

```
A design using tables with two panes: menu and display. The display  
pane has a caption followed by data. Commands are placed in the menu.
```

5. Save and close the `brilliant.txt` file.

Clear Existing Generated HTML Templates

WebEnterprise Designer's HTML template generator is designed to identify and merge customizations made to the HTML templates. However, the process of customizing a page design not only defeats the generator's merge logic, but can result in a large number of spurious errors. Therefore, before generating an HTML application that uses a modified page design, first delete the entire contents of the application's HTML template directory.

► To clear generated HTML templates

1. Delete the `${FORTE_ROOT}/html/docs/HTMLtutApp` subdirectory.

This includes the `.base` subdirectory.

2. During the customization process, each time you modify a design used by an HTML application, clear the old generated HTML templates before generating a new set.

Once the design is stable, you can omit this step.

Customize the Design Files

Depending upon the nature of the customization, this step can be straightforward or extremely complex. For example, removing the caption from the main pane is a simple customization that requires only that you delete several lines from each design file that displays captions (the `Display`, `Update`, `Insert`, and `Search` files).

In this section, you will customize the display of form pages. This is only the first part of the task of customizing an entire design. Customization of the other parts of this page design is left as an exercise for the reader.

To customize the display of form pages, you will:

1. Extract the navigation menu block from the form's `Data` design file.
2. Place the extracted menu block in a new `Menu` file.
3. Modify the `DataWindow Display` design file, defining the pane table structure, and including the new `Menu` design file.

Remove the Menu From the Data File

If you compare the list of files in the simple design directory with that of the fancy design (and its variants), you will notice that, unlike the fancy designs, simple has no `Menu` file (`_RC_IsDataWindow_Menu__RR_Name_.html`). This is because the navigation menu for the simple design is in the `Data` file (the `_RC_IsFormWindow_Data__RR_Name_.html` file).

In this example, you will create a `Menu` file for the brilliant design. One way to do this is to take the menu block from the `Data` file and put it into a `Menu` file that you create. But this menu is not in the correct form. The simple menu is a horizontal line of menu items placed immediately below the form data. Our new design is modeled on the fancy design family, where the menu is a vertical array of items.

3. Remove this entire block from the file.
4. Save and close the `_RC_IsFormWindow_Data__RR_Name_.html` file.

Modify the Menu Design File

In this section, you will create a Menu file for the brilliant design by copying the Menu file from the fancyMenu directory to the brilliant directory. You will then change the file from an HTML frame to an included part of a frameless page. This requires two changes. You will:

- strip off all lines at the top and bottom that are required for frames
- eliminate all the frame-related “target” attributes

► To create and modify the Menu design file

1. Copy the `_RC_IsDataWindow_Menu__RR_Name_.html` file from the fancyMenu design directory and paste it in the brilliant directory.
2. Open the `_RC_IsDataWindow_Menu__RR_Name_.html` file in a text editor.
3. Strip off lines at the top and bottom that are required for frames.

Include everything from the first line through the `<body>` tag at the top, and the `</body>` and `</html>` tags from the end of the file.

Remove the **bolded** lines:

```
<!-- Forte WebEnterprise Designer Menu form definition for model
{{%T(IsAabModel)[%P(Name)]%O[%E(Parent)[%P(Name)]]}} using page
design fancyMenu and style {{%P(HTMLStyle)}}. Generated on
20-Feb-1999 13:06:17 -->
<!--{{%A(page)}}-->
<!--<?forte assign USER.TopPage="{{%R(page)[%P(Name)]}}">-->

<html>
  <head>
    <link href="/forte/styles/{{%R(page)[%P(HTMLStyle)]}}.css"
rel="stylesheet" type="text/css">
  </head>

[Note the following line for later.]
  <body class="menu">
    <p class="captionmenu">
      {{%R(page)[%P(Name)]}} Menu:
    </p>
      .
      .
      .
  </body>
</html>
```

4. Eliminate all the "target="_fortedisplay" attributes on the menu's anchor tags (<A>).

These are all frame-related and will have unpleasant effects in a frame-less application. For example, remove the **bold** text from all lines like this:

```
<a class="amenu" HREF="{{$FORTE.ExecURL}?ServiceName={{%R(...
&Uniquifier={{FORTE.UniqueID}} " target="_fortedisplay">
  {{%P(Label)}}</a>
```

5. Save and close the `_RC_IsDataWindow_Menu__RR_Name_.html` file.

Modify the Display Design File

The Display HTML template (the `_RC_IsDataWindow_Display__RR_Name.html` file) is the main HTML page for displaying data. It creates the panes in the browser's page and places the menu on the left and the data in the rest. In this section, you will create an HTML table for the entire page, a table row that is the entire table, and then two table data elements (defined by the HTML `<td>` tag) within that row.

The first `<td>` in the table you will create contains the menu. We'll use the `class="menu"` attribute (noted in the code of the fancyMenu menu's `<body>` tag, which you excised in the previous section) to get the proper menu style elements (colors and font).

► **To modify the Display file to create the required display and call the Menu file**

1. Open the `_RC_IsDataWindow_Display__RR_Name.html` file in a text editor.
2. Find the display block.
3. Remove the line that includes the Menu.

Remove the **bold** line:

```
[Begin display block]
<body class="display">

  <!--<?forte execute {{%R(page)[%P(Name)]}}Handler.ProcessAction
    resultset="entry">-->
  <!--<?forte assign USER.TopPage="{{%R(page)[%P(Name)]}}">-->
  <!--<?forte include "{{%R(page)[%E(Parent)][%P(Name)]}}/Data_
    {%R(page)[%P(Name)]}.html">-->

End display block
</body>
```

4. Replace the deleted line with the following code:

```
<table border=0 width="100%" cellspacing=0 cellpadding=0>
  <tr>
    <td valign=top width=155 rowspan=3 class="menu">
      <!--?Forte include
"{{%R(page)[%E(Parent)[%P(Name)]}}/Menu_{{%R(page)[%P(Name)]}}.html">-->
      </td>
    <td>
      <!--?Forte include
"{{%R(page)[%E(Parent)[%P(Name)]}}/Data_{{%R(page)[%P(Name)]}}.html">-->
      </td>
  </tr>
</table>
```

5. Save and close the `_RC_IsDataWindow_Display__RR_Name.html` file.

Generate and Inspect

The next step is to generate the application and inspect for errors. Generation errors are written to the Launcher log, and are also noted in the generated HTML template, with syntax like:

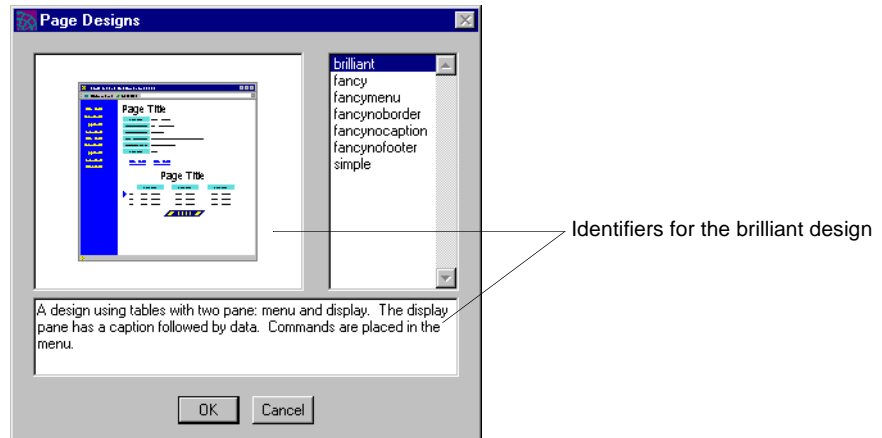
```
*** BAD SELECTOR USED IN VARIABLE REFERENCE
```

► To verify your work by generating code

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Choose File > Properties to open the HTML Application Properties dialog.

3. Click the browser on the Page Design field to display the available page designs.

The brilliant design appears in the list:

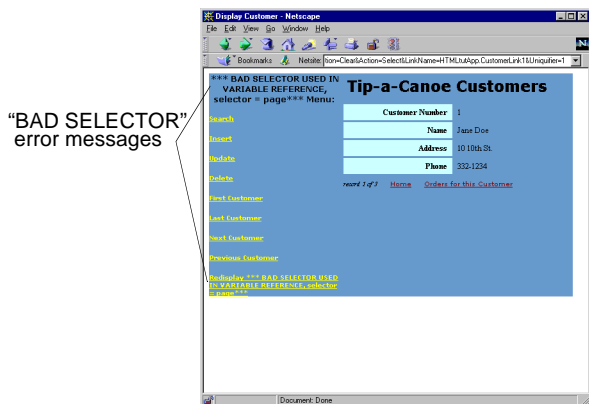


4. Select brilliant and click OK.
5. Click OK to apply the changes and close the HTML Application Properties dialog.
6. Generate the code for the application.
7. Inspect for errors by opening the Launcher window and running the application and looking for "BAD SELECTOR" messages.

Our example does, in fact, have errors. The launcher window has a series of error messages:

The template variable page is undefined.

And when you run the HTMLtutApp application, the windows contain a number of “BAD SELECTOR” messages:



The problem is that the top of the `_RC_IsDataWindow_Menu__RR_Name_.html` file (which you copied from fancyMenu in “[Modify the Menu Design File](#)” on [page 129](#)) contains a code generation directive that is essential to the directives contained within the menu block. This directive assigns a value to the code generation page object. The menu uses this object, but you deleted it from the menu design file, so you need to replace it.

➤ **To fix the error in the Menu file**

1. Open the `_RC_IsDataWindow_Menu__RR_Name_.html` file in a text editor.
2. Insert the following line (the assignment of the page object) at the top of the file.

```
<!--{ { %A(page) } }-->
```

(This line is also at the top of `_RC_IsFormWindow_Data__RR_Name_.html`, if you want to copy it from another file.)

3. Delete the `#{FORTE_ROOT}/html/docs/HTMLtutApp` subdirectory, including the `.base` subdirectory.

Remember, you have to do this every time you change a design template. See “[Clear Existing Generated HTML Templates](#)” on [page 126](#) for information.

4. Attempt to regenerate and run the HTMLtutApp application.

When you click on the Generate Web Application Server Code button, nothing happens! This is because the code generator is sensitive to changes in the WebEnterprise Designer definition of the model, but is unaware of the external state of HTML templates and page designs. In order to force a code generation, you must change something in the HTML Application.

5. Change the Title field of the HTML application in the property dialog, and then change it back again.

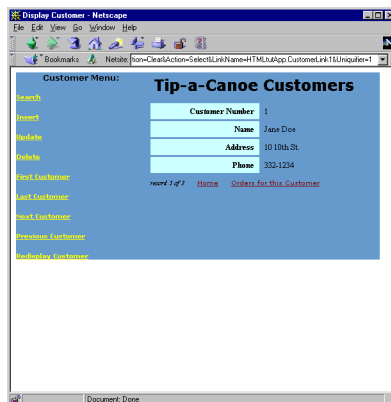
For example, change the title from “HTML Tutorial” to “HTML Tutorial1” and then change it back to “HTML Tutorial.”

6. Regenerate the application.

New HTML template files are created. The launcher window should show no errors.

7. Run the HTMLtutApp application.

The application is displayed in the new brilliant design, that should also have no errors:



Fine-Tune the Customized Design

The following steps are the minimum ones you must do to complete the new design:

- using the Display page as a model, convert the other page modes (Search, Update, and Insert) to panes using tables
- modify also the layout of list pages (this example only modified form pages)

Further fine-tuning could include incorporating the common features of your company's Web applications, for example, by adding the company logo to headers and footers.

Testing a customized design requires a fairly rich HTML application, exercising all the features of WebEnterprise Designer applications. Over time, as your own applications evolve in complexity and power, your customized design will improve until it is exactly what you want.

Customizing Page Styles

WebEnterprise Designer contains two HTML style sheets—cool and steel—that govern the presentation aspects of application pages. In the WebEnterprise Designer Application Model Workshop, these are referred to as styles. This chapter provides information on these styles and how to customize them, including:

- what HTML style sheets are
- the characteristics of WebEnterprise Designer native styles
- how to customize styles and add them to the WebEnterprise Designer application environment

HTML 4.0 and Style Sheets

WebEnterprise Designer creates Web pages that conform to the HTML 4.0 standard defined by the World Wide Web Consortium (W3C). The HTML 4.0 specification is available at <http://www.w3c.org/TR/REC-html40>.

A recent aspect of the evolution of HTML is the separation, in HTML 4.0, of the presentation aspects of Web documents (which include such properties as font information, colors, and alignment) from the layout of the page. In HTML 4.0, the presentation aspects are contained in style sheets, which are combined with HTML documents to produce fully rendered pages in a browser. The tags and attributes of earlier versions (HTML 2.0 and HTML 3.2), including “” and “bgcolor=”, are now listed as “deprecated” features, in favor of style sheets as the recommended mechanism for describing presentation.

The browsers supported by WebEnterprise Designer, Netscape Navigator (version 4.0 and above) and Microsoft Internet Explorer (version 4.0 and above), both support HTML 4.0 documents, including style sheets conforming to the W3C Cascading Style Sheets, level 1 specification (CSS1). The HTML templates generated for WebEnterprise Designer applications are designed to work in concert with CSS1 style sheets. This specification is available at <http://www.w3c.org/pub/WWW/TR/REC-CSS1>.

WebEnterprise Designer and Style Sheets

Each page defined in the HTML Application Workshop is associated with a page design and a style. The design governs the layout of the application's page. When you generate the application's Web server code, pages are processed through this design, resulting in HTML templates. This process and its opportunities for customization are the subject of [Chapter 4, "Customizing Page Designs."](#)

WebEnterprise Designer styles are native style sheets used by the generated HTML templates to format their HTML elements. There are two native styles, *cool* and *steel*. Cool defines a colorful presentation using both serif and sans-serif fonts. Steel creates pages devoid of color that use only sans-serif fonts.

The style sheets themselves are part of the WebEnterprise Web server installation and are installed under the Web server root. When you specify them, you use URLs relative to the Web server document root. For example, you specify the cool style with `/forte/styles/cool.css`, and the steel style with `/forte/styles/steel.css`. The actual locations of these files might be one of the following:

- `c:\Netscape\Suitespot\forte\styles\cool.css`—location of the cool style on the Netscape Web server root directory on NT
- `/usr/netscape/suitespot/forte/styles/steel.css`—location of the steel style on the Netscape Web server root directory on UNIX
- `c:\InetPub\forte\styles\steel.css`—location of the steel style on Microsoft IIS on NT

The rest of this chapter provides a description of style elements used in WebEnterprise Designer HTML templates and offers guidelines for creating your own styles to use with WebEnterprise Designer applications.

Using HTML Style Elements with WebEnterprise Designer

An HTML template generated by WebEnterprise Designer refers to style sheets in two ways:

- to name the style (CSS1) file to use to process the page
- to use elements defined in the style sheet to format HTML elements of the page

Identifying the Style Sheet to Use

Use the HTML link tag in the document header to identify the style file to use for processing the page. For example:

```
<link href="/forte/styles/cool.css" rel="stylesheet" type="text/css">
```

This line specifies the cool.css style sheet file. (File specification for style sheets are relative to the Web server root directory; see [“WebEnterprise Designer and Style Sheets” on page 138](#) for a discussion of this.) The browser asks the Web server to fetch the style sheet (unless it is already in the browser’s cache) and then uses its contents to format the HTML page.

Using HTML Attributes

The HTML 4.0 and CSS1 specifications provide several ways to apply styles to a document. WebEnterprise Designer employs two HTML 4.0 style sheet selectors:

- the class attribute, which assigns the style to all elements defined by the specified tag
- the id attribute, which assigns the style to a single element

In HTML, the class and id attributes are element identifiers, and in particular, document-wide element identifiers.

The class Attribute

You can use the class attribute with many HTML tags to specify a CSS1 style element. For example, consider this class attribute specification within an HTML

```
<body> tag:
```

```
<body class="display">
```

The class attribute in this case directs the browser to find the *display* element in the current style sheet and apply its attributes to the HTML `<body>` tag.

In this example, the current style sheet is the cool.css file. The display element in cool.css is specified as follows:

```
[HTML 4.0 specifications]
.display {

[background color]
    background-color: rgb(255,255,255);

[document font]
    font-family: "Times New Roman", Times, Georgia,
                "Century Schoolbook", "Bookman Old Style", serif;
}
```

In the above code, the document's background color is defined as white. A list of fonts is specified for the document; the browser will choose the first one it supports. In previous versions of HTML, you would specify these attributes with the following tags:

```
[Pre-4.0 background color]
<body bgcolor = "FFFFFF">

[Pre-4.0 document font]
<basefont face = "Times New Roman, Times, Georgia, Century Schoolbook,
Bookman Old Style, serif">
```

The Scope of the class Attribute

An important aspect of the class identifier is that the attributes of the style element identified by it (that is, display) apply to all HTML elements defined between the `<body>` and `</body>` tags (in this case, the entire document). If the class attribute were placed in the `<table>` tag, it would apply to all elements between the `<table>` and `</table>` tags.

Finally, one class may override another within its scope. Consider the following HTML tags:

<code><body class="display"></code>	WHITE
<code><p>Table of Contents</p></code>	WHITE
<code><table class="mytables"></code>	RED
<code><tr> . . . </tr>></code>	RED
<code></table></code>	RED
<code><table></code>	WHITE
<code><tr> . . . </tr></code>	WHITE
<code></table></code>	WHITE
<code></body></code>	WHITE

If the display class defines the background color as white and the mytables class defines it as red, the document's background color shifts from white to red and then back to white, as specified by the labels in the column at the right of the code.

The id Attribute

Like the class attribute, the id attribute is an element identifier. The difference between id and class is that id applies only to the current tag. For example:

```
<div id="statusline">Row 1 of 42</div>
```

In this line, id applies the statusline style elements only to the output string "Row 1 of 42." If other HTML tags are defined within the block defined within `<div>` and `</div>`, they are not governed by statusline.

Using HTML Style Elements

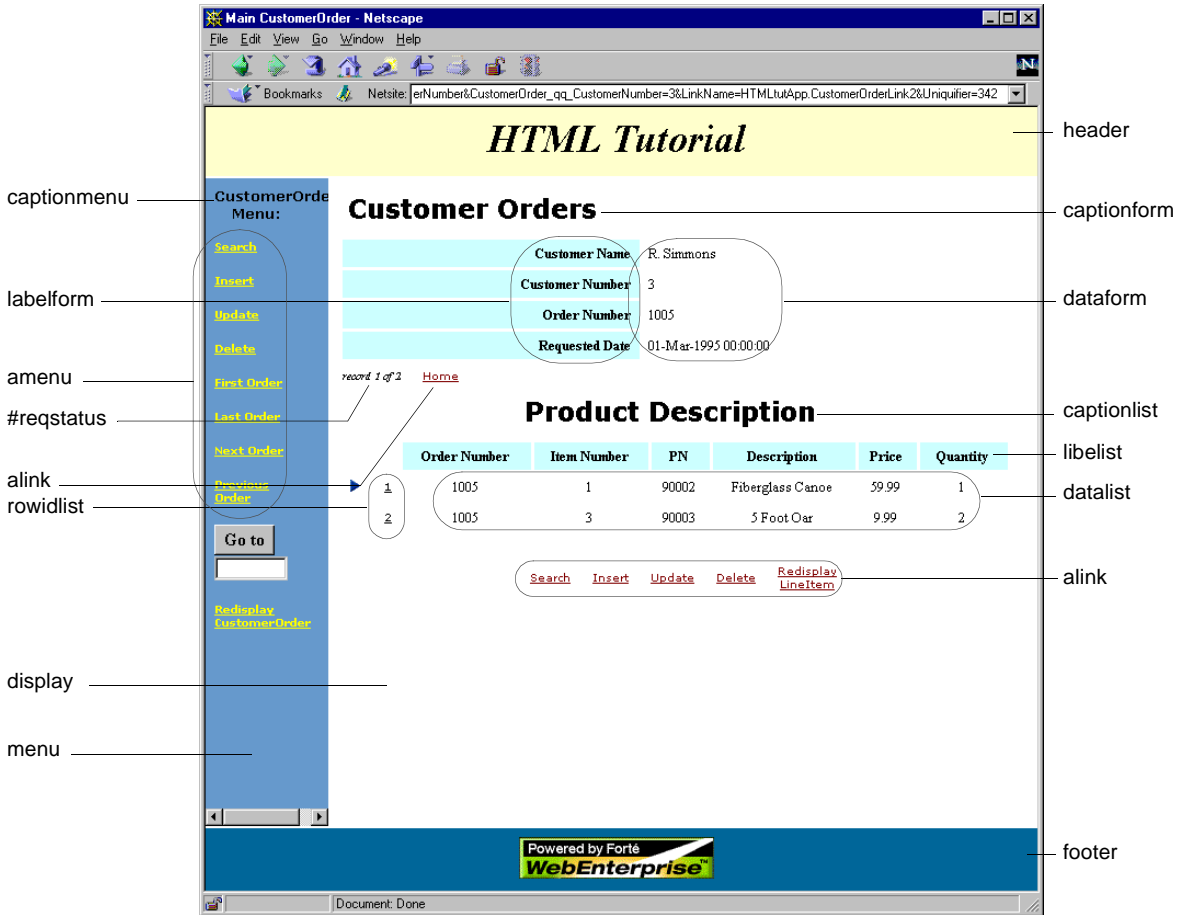
Both WebEnterprise Designer styles (cool and steel) use a common set of CSS1 elements. These elements are:

Style type	Style element	Use for formatting:
General page-level styles	display	The display frame body of the page's display mode.
	modify	The display frame body of the page's Insert and Update modes.
	menu	The body of the menu frame.
	footer	The body of the footer frame.
	header	The body of the header frame.
Anchors (links)	amenu	The anchors (links) in the menu frame.
	alink	The anchors (except for Data or Field Label links) in the display frame.
	adata	The anchors in the display frame that are attached to data items (links whose Activate Link On property is set to Data).
	alabel	The anchors in the display frame that are attached to the label of data items (links whose Activate Link On property is set to Field Label).
Page captions	captionform	The caption above the retrieved data on data pages of form pages.
	captionlist	The caption above the retrieved data on data pages of list pages.
	captionlogon	The caption above the data entry fields on logon pages.
	captionmenu	The caption above the commands in the menu frame.

Style type	Style element	Use for formatting:
Column labels	rowidlist	The row ids on list pages.
	labellist	The field labels above the data columns on the data templates of list pages.
	labelform	The field labels on the data templates of form pages.
	labelmodify	The input field labels on the data templates of insert, update, and search mode pages.
Column data	datalist	The retrieved data on the data templates of list pages.
	dataform	The retrieved data on the data templates of form pages.
Form buttons	buttons	All buttons on all page modes of data pages (for example, the Insert and Search buttons).
	buttonlogon	The button on logon pages.
Status line (accessed by "id=")	#reqstatus	The status line of the data display (Record 1 of n).

Figure 5-1 shows how some of these elements appear on a Web page.

Figure 5-1 Style Element Examples



Customizing Page Styles

The first step in customizing a page style is to assess the scope of your project. You should ask yourself the following questions:

- Will you be modifying elements only or adding new ones as well?

If you modify only existing elements, you only need make changes to the style sheet itself. If you add new elements, you must add them to both the generated HTML templates and to the style sheet they will use.

- Will the new style be available to all applications, or will it be restricted?

If the use of the new style is restricted to specific applications, then you can simply customize each application's HTML templates to use the new style. If any application is allowed to use the style, you must add the style to the list in the Style browser in the HTML Application Model Workshop. You might also want to add the elements to one or more page designs.

- Will the new style be usable with all available page designs or will it be restricted?

If you do not add support for the new style elements to all page designs, this can result in some invalid combinations of designs and styles. HTML templates using a particular style element will display correctly only if the style being used implements the element. You can either add the element to all styles or document any restrictions to design/style combinations in the textual descriptions that display in the Style and Page Design browsers.

The following sections provide customization guidelines under different scenarios arising from how you answer these questions. All discussions and examples refer to creating a new style, which is based on the cool style, and named the "marketing" style.

Outline of Basic Procedures

The general steps for creating and implementing a customized style are:

1. Create a new style sheet file by copying the most similar existing one, and rename it with the new style name.

Details are given in ["Creating the New Style Sheet File" on page 146](#).

2. Determine whether you will add new elements to the style sheet or only modify existing elements.

If you are only modifying existing elements, refer to ["Modifying Existing Elements" on page 146](#) for information.

If you are adding new elements, refer to ["Adding New Elements" on page 147](#).

3. Determine whether use of the new style is to be restricted or not.

If you are restricting the style to certain applications, refer to ["Adding New Elements" on page 147](#).

If the style is to be made generally available, you will want to identify it in the HTML Application Model Workshop. Refer to [“Identifying the Style with a Bitmap and Text” on page 148](#) for details.

4. Test the new style with different browsers and adjust accordingly.

Refer to [“Considering the Browser” on page 150](#) for important information and guidelines regarding browsers.

Creating the New Style Sheet File

The first step in creating the new marketing style is to create a `marketing.css` file in the WebEnterprise Designer styles directory. The styles directory depends on where your Web server directory is. See [“WebEnterprise Designer and Style Sheets” on page 138](#) for a description of several possibilities, or see your system administrator for the exact location in your installation.

► To create a new style sheet file

1. Find the `cool.css` file in the `Web_server_root/forte/styles` directory.
2. Copy the `cool.css` file to a new `marketing.css` file in the same directory.

Now application pages that use the marketing style will look just like cool style pages.

Modifying Existing Elements

When you make customizations that only modify existing elements, such customizations generally require changes only to the style sheet itself. Guidelines for this type of customization are as follows:

► To customize only existing elements in the new style sheet

1. Create a new style sheet, as described in [“Creating the New Style Sheet File,”](#) above.
2. Modify a few elements in the new style sheet.

Refer to [“Using HTML Style Elements with WebEnterprise Designer” on page 139](#) for information on the elements used in the WebEnterprise Designer style sheets. For a basic reference on HTML 4.0, see the HTML 4.0 specification at <http://www.w3c.org/TR/REC-html40>.

3. Test the style sheet by regenerating code and running the application.

If your changes do not appear, refer to [“Browser Caching” on page 150](#) for a solution.

4. Repeat [Step 2](#) and [Step 3](#) until your new style is perfected.

Remember to read [“Considering the Browser” on page 150](#) for information related to testing and use of your new style.

Adding New Elements

If you choose to add new style elements, you must add them to both the style sheet and the generated HTML templates that refer to it. How you do this depends on whether or not you want the style to be available to all applications or not.

► To customize a restricted style

1. Open the new CSS1 style sheet (marketing.css from [“Creating the New Style Sheet File,”](#) above).
2. Make your changes, including adding new elements.

For information on elements used by WebEnterprise Designer, refer to [“Using HTML Style Elements with WebEnterprise Designer” on page 139](#). For a basic reference on HTML 4.0, see the HTML 4.0 specification at <http://www.w3c.org/TR/REC-html40>.

3. Save and close the marketing.css file.
4. Open each of the application’s HTML templates and change the <link> tag to your new style.

Example

```
<link href="/forte/styles/marketing.css" rel="stylesheet"
      type="text/css">
```

5. Modify the HTML, as appropriate, to use your new style elements.

See [Chapter 3, “Customizing Generated HTML Templates,”](#) for details on customizing HTML templates.

If, however, multiple applications can use the new style elements, then you might want to follow these procedures instead:

► **To customize generally available styles**

1. Identify the new style to the HTML Application Model Workshop, as described in “[Identifying the Style with a Bitmap and Text](#),” below.

This automatically customizes the <link> reference in the HTML templates of any application that uses this style.

2. Customize your new style sheet, as in the previous procedure.
3. Instead of modifying the individual templates of an application, modify one or more page designs.

See [Chapter 4, “Customizing Page Designs,”](#) for details.

Bear in mind, however, that adding support for new style elements to a restricted number of designs or styles can result in some invalid combinations of designs and Styles. For example, if a new design uses a new style element that is only implemented in one new style, then application pages using the new design *must* use the new style. Obviously, HTML templates using a particular style element will only display correctly if the style in use implements the element. To avoid this restriction, you could add the element to all styles, or you can document the restrictions in the design and style description files (for example, the marketing.txt files described next in “[Identifying the Style with a Bitmap and Text](#)”).

Identifying the Style with a Bitmap and Text

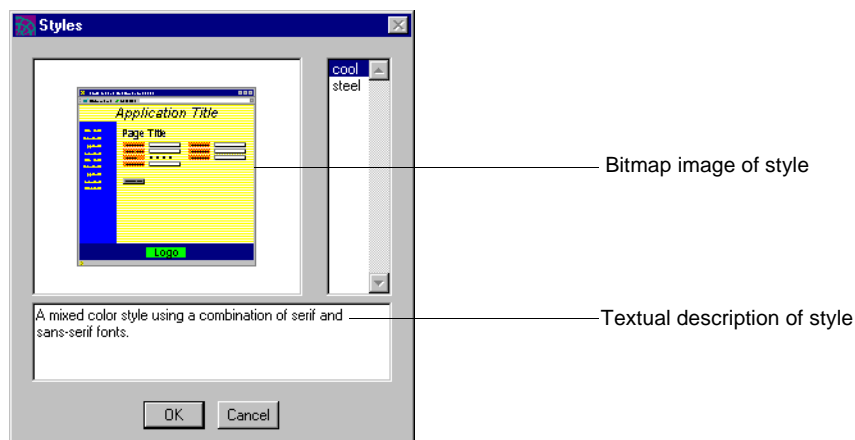
If you will be making the new style generally available to all applications, then you need to identify and describe the style in the HTML Application Model Wizard. Two files within the `#{FORTE_ROOT}/userapp/express/clx/styles` directory identify the style in the Style Properties dialog:

- a bitmap image of a thumbnail picture of a page created with the style
- a brief textual description of the style

(You access the Styles properties dialog by clicking the Style browser button on either the HTML Application Properties dialog or on the Page Options page of the Page Wizard.)

For example, [Figure 5-2](#) shows the identifiers of the cool style in the Style Properties dialog:

Figure 5-2 Style Identifiers



To make your new style available to the workshop, you must add two files to that directory.

➤ **To identify the marketing style**

1. Make a copy of the `cool.bmp` file in the `${FORTE_ROOT}/userapp/express/clx/styles` directory.
2. Rename the `cool.bmp` file `marketing.bmp`.
3. Create a text file named `marketing.txt` in the same directory.
4. Add the following text to the `marketing.txt` file:


```
The corporate standard style for external Web applications.
```
5. Save and close the `marketing.txt` file.

Now when you display the Style Property dialog, you will see three choices: `steel`, `cool`, and `marketing`. If you select the `marketing` style, the generated HTML templates will automatically contain a `<link>` tag referring to the `marketing.css` file.

Considering the Browser

This section discusses two important considerations regarding browsers and customized style sheets.

Browser Caching

One difficulty with implementing a new style sheet is that browsers tend to cache the style sheet in their local memory. Even though you have modified the style sheet, the browser can choose to use its local, unmodified copy, instead. Browser preference settings affecting cache behavior do not apply to style sheets in the same way they apply to HTML documents. If you make style sheet changes, but the appropriate changes are not displayed when you run the application, examine the Web server log to see if the browser is re-fetching the old style file. If so, shut down and restart the browser.

Browser Independence

An important objective of WebEnterprise Designer is that generated applications be browser-independent. That is, application pages should look much the same whether viewed with Netscape Navigator or Microsoft Internet Explorer.

The two style sheets included with WebEnterprise Designer achieve this objective. However, it has been our experience that the support for CSS1 in the browser products is not as robust as their support for HTML 4.0. Style sheet elements that appear correct might, in fact, not display correctly.

For example, CSS1 supports specification of colors by keyword (such as “white”, “red”, “blue”, and so forth). However, one of the browser products does not currently implement this correctly. For this reason, we have adopted the “rgb” (red, green, blue) syntax for color specification, which both products correctly implement. An example of this is from the cool.css file:

```
.display (  
    background-color: rgb(255,255,255);
```

It is vital that applications in general, and style sheets in particular, undergo testing with both browser products.

Customizing Error Pages

When errors are detected during the processing of WebEnterprise Designer applications, they are displayed in the browser as an error page. You can customize the appearance of this error page or create your own.

This chapter provides information on how to:

- modify the appearance of the default HTML exception page
- create customized error pages

Most of the information in this chapter applies to WebEnterprise applications, as well as WebEnterprise Designer applications.

WebEnterprise Exception Handling

WebEnterprise applications, including those created with WebEnterprise Designer, are based on a request-response model, in which the browser client requests iPlanet UDS services, which are returned in HTML form (Web pages in the browser). When errors occur during the processing of these requests, iPlanet UDS exceptions must be converted to an HTML representation and returned to the client.

WebEnterprise uses dynamic exception mapping to format exceptions. Standard template-driven HTML Scanner technology allows the developer to control the appearance of error pages.

In addition, the WebEnterprise Designer Page Handler Customization Wizard contains customization points that permit arbitrary application-specific processing of exceptions.

Default Exception Processing

The default WebEnterprise exception processing steps are as follows:

1. Exceptions raised within the WebEnterprise runtime system are caught and processed at two distinct points:
 - exceptions raised during the processing of an HTMLScanner template request are processed within the HTMLScanner service object
 - all other exceptions are processed within the HTTPAccess service object
2. When WebEnterprise catches an exception, it maps the exception class to an error HTML template file to format the exception's attributes.

The default error HTML templates all reside in the `${FORTE_ROOT}/html/errors` directory. Exceptions are mapped to templates as follows:

Error	Mapped to error template
HTMLScannerException	htmlscanner.html
HTTPAccessException	httpaccess.html
All other exceptions	generic.html

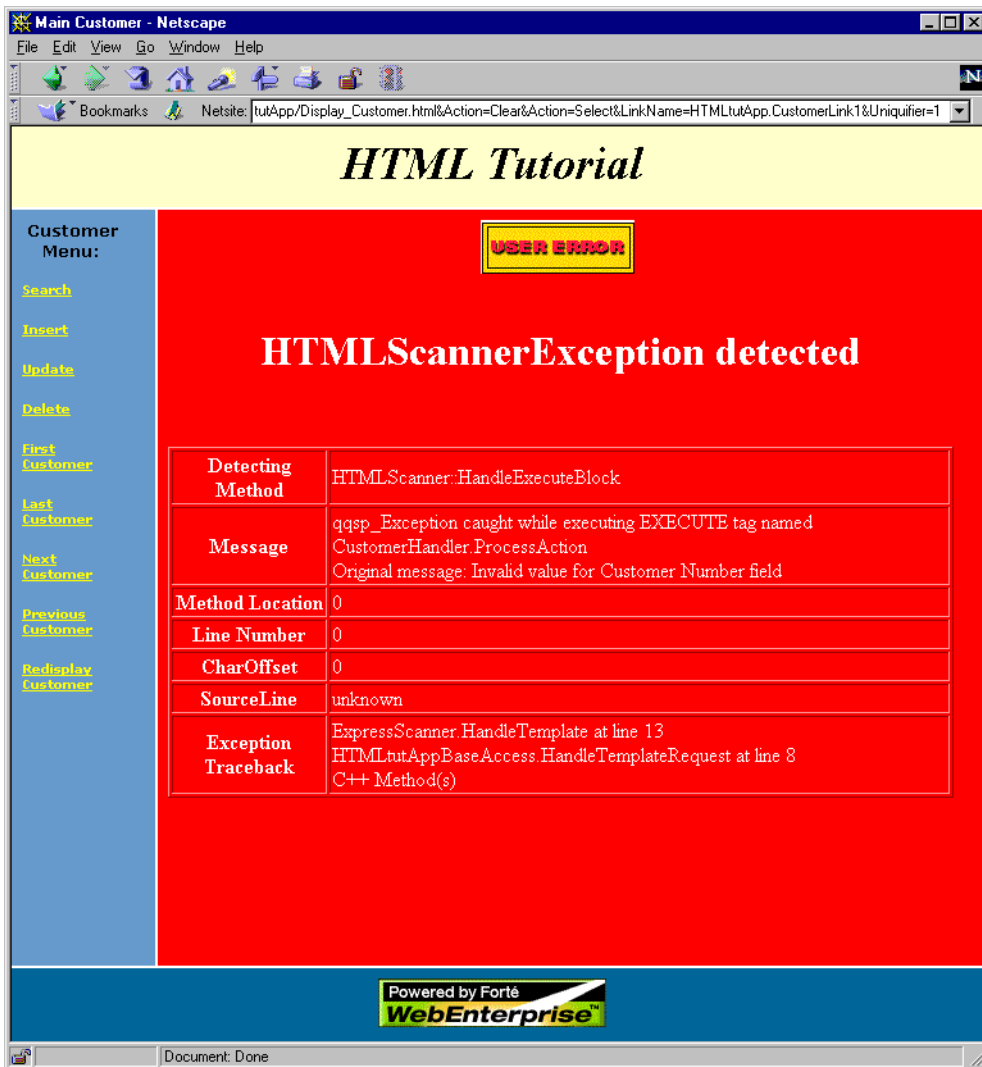
3. WebEnterprise then generates an internal HTTPRequest, with the appropriate error template file as the requested template.
4. WebEnterprise converts the exception's attributes to HTTPRequest parameters, which are accessible to the HTML template within the FORTE pseudo-result set.

Exception attributes are listed and described in the next section, ["WebEnterprise Exception Result Set Variables."](#)

5. WebEnterprise then processes the template file using standard HTMLScanner procedures.
6. The default error templates display all exception attributes.

An example error page is shown in [Figure 6-1](#).

Figure 6-1 Default Scanner Exception Page



WebEnterprise Exception Result Set Variables

As described in the previous section, when WebEnterprise processes exceptions, it converts exception attributes to HTTPRequest parameters, which are then accessible from the HTML templates by means of the `$(FORTE.parameter_name)` specification. (For information on the syntax of iPlanet UDS variables, see *A Guide to WebEnterprise*.)

The sections that follow provide a table listing each exception class's attributes and their associated HTMLScanner result set variable references. Attributes that are undefined are assigned a default value (generally "0" for a numeric attribute and "n/a" for text attributes).

HTMLScannerException Class Variables

The following table lists the attributes of the HTMLScannerException class and their corresponding result set variables. These are the variables used in the `htmlscanner.html` template.

Attribute	Result set variable	Description
Classname	<code>\$(FORTE.ExceptionClass)</code>	"HTMLScannerException"
Message	<code>\$(FORTE.Message)</code>	The error's text
DetectingMethod	<code>\$(FORTE.DetectingMethod)</code>	The method raising the exception
MethodLocation	<code>\$(FORTE.MethodLocation)</code>	Additional identifying information
Severity	<code>\$(FORTE.Severity)</code>	The severity text description
	<code>\$(FORTE.SeverityGIF)</code>	The severity image file
CharOffset	<code>\$(FORTE.Charoffset)</code>	The byte offset in the line of the error
LineNumber	<code>\$(FORTE.LineNumber)</code>	The line number in the HTML template
SourceLine	<code>\$(FORTE.SourceLine)</code>	The text of the HTML template line
ExceptionTraceback	<code>\$(FORTE.ExceptionTraceback)</code>	The method's traceback

HTTPAccessException Class Variables

The following table lists the attributes of the HTTPAccessException class and their corresponding result set variables. These are the variables used in the httpaccess.html template.

Attribute	Result set variable	Description
Classname	\$(FORTE.ExceptionClass)	“HTTPAccessException”
Message	\$(FORTE.Message)	The error’s text
DetectingMethod	\$(FORTE.DetectingMethod)	The method raising the exception
MethodLocation	\$(FORTE.MethodLocation)	Additional identifying information
Severity	\$(FORTE.Severity)	The severity text description
	\$(FORTE.SeverityGIF)	The severity image file
ExceptionTraceback	\$(FORTE.ExceptionTraceback)	The method’s traceback

Variables for All Other Exceptions

The following table lists the attributes and corresponding result set variables used for errors that are neither HTMLScannerException nor HTTPAccessException errors. These are the variables used in the generic.html template.

Attribute	Result set variable	Description
Classname	\$(FORTE.ExceptionClass)	The name of the exception class
Message	\$(FORTE.Message)	The error’s text
DetectingMethod	\$(FORTE.DetectingMethod)	The method raising the exception
MethodLocation	\$(FORTE.MethodLocation)	Additional identifying information
Severity	\$(FORTE.Severity)	The severity text description
	\$(FORTE.SeverityGIF)	The severity image file

Customizing Error Pages

This section provides customization guidelines for WebEnterprise error pages. The different levels of customization discussed include:

- modifying the default error pages
- creating a custom error page
- adding new result set variables
- using the error page customization point in the Page Handler Customization Wizard

All but the last level are available to all WebEnterprise users; the last level is available only to WebEnterprise Designer users.

Modifying Default Error Pages

The simplest way to customize WebEnterprise error pages is to modify the HTML template files in the `${FORTE_ROOT}/html/errors` directory. It is a straightforward task to adjust the HTML tags to produce an installation-specific or company-wide error page format.

For a basic reference on HTML 4.0, see the HTML 4.0 specification at <http://www.w3c.org/TR/REC-html40>.

Creating Custom Error Pages

If you want your application to log errors, delete sessions, or take other application-specific action beyond the default WebEnterprise exception handling procedure, you can use two additional levels of customization that are supported within WebEnterprise, namely:

- creating your own error HTML template files
You can use the variables described in “[WebEnterprise Exception Result Set Variables](#)” on page 154 and customize the exception class mapping, as well.
- creating your own result set variables

With your own variables, you can include application-specific dynamic data on the error page.

While these customization levels are available to any WebEnterprise application, WebEnterprise Designer provides customization points that simplify the creation of custom error handlers. These are described in the next section.

Customizing a WebEnterprise Designer HTML Application

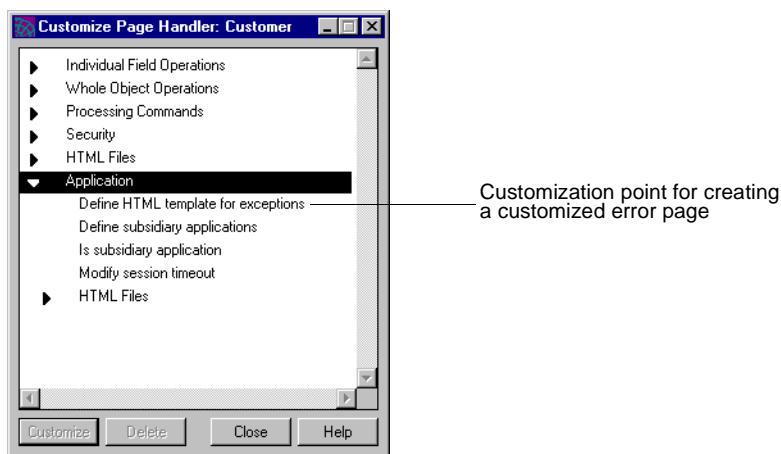
WebEnterprise Designer's Page Handler Customization Wizard contains a customization point for customizing an application's error pages. For complete information on the Customization Wizard, see ["Customizing With the Page Handler Customization Wizard"](#) on page 67.

► **To customize an application's error page using the Customization Wizard**

1. Open your application in the HTML Application Model Workshop.
2. Select the business class page you wish to customize.
3. Choose the Component > Customize... command to open the Wizard.

If this is the first customization you are making to this class, WebEnterprise Designer displays a dialog requesting permission to generate a customizable class for the page. Click OK.

4. Open the Application group by clicking on the arrow next to it.



5. Select the “Define HTML template for exceptions” customization point and click Customize.

The Method Workshop opens, displaying the `GetErrorTemplate` method. This method overrides the `HTMLScanner.GetErrorTemplate` method defined.

6. Add your code to implement application-specific error processing in `GetErrorTemplate`.

For information on `GetErrorTemplate`, click the Help button while this customization point is selected. Documentation is also provided as comments in the method body, as well. The Help topic description contains a link to a description of the `HTMLScanner.GetErrorTemplate` method.

The following sections provide an expanded version of the Help documentation.

The `GetErrorTemplate` Method

The `GetErrorTemplate` method provides a means of modifying the HTML template that should be used when an exception is generated during processing a user request.

GetErrorTemplate (*e=GenericException, request=HttpRequest, customParameter=array of NamedElement*)

Returns Framework.TextData

Parameter	Required	Input	Output
e	●	●	
request	●	●	
customParameters	●		●

The arguments to the `GetErrorTemplate` method are:

Argument	Description
e : <code>GenericException</code>	The actual exception object thrown by the application.
request : <code>HttpRequest</code>	The <code>HttpRequest</code> object being processed when the exception was thrown.

Argument	Description
customParameters : Array of NamedElement	An optional array of name-value pairs. If this is returned, the elements are added to the error's HTTPRequest parameters and are therefore available to the error HTML template as resultset variables.

Return Value

GetErrorTemplate returns a TextData object that contains the specification of the HTML template file to be used to format and return the error. As with all HTML template specifications, this file is assumed to reside within the WebEnterprise document root. This makes it simple to access an application-specific error page.

WebEnterprise also provides the ability to directly access HTML documents (including HTML templates) that reside under the installation's html directory. Any template specification that begins with the token `"/forte/"` is re-mapped to the `/${FORTE_ROOT}/html` directory.

For example, the template name `"/forte/errors/htmlscanner.html"` resolves to `/${FORTE_ROOT}/html/errors/htmlscanner.html`. Using this syntax, a site can easily create site-wide HTML template directories.

Example: Application-Specific Error Template

This example maps HTMLScanner exceptions to an application-specific (Movie) error template and all other exceptions to the standard generic error template.

```
// GetErrorTemplate(e: GenericException, request: HTTPRequest,
//   output customParameters: Array of NamedElement) : TextData
//
errorTemplate : TextData = new;
customParameters = nil;
//
if e.IsA(HTMLScannerException) then
    errorTemplate.SetValue('Movie/Error_Scanner.html');
else
    errorTemplate.SetValue('/forte/errors/generic.html');
end if;
//
return errorTemplate;
```

Example: Application-Specific Template with Custom Data

This example extracts the user name associated with the session (previously attached to the session during Logon) and adds it to the customParameters object. This will cause it to be added to the result set variables. A custom error HTML template (not illustrated here) will then display the user name on a standard application error page. (See [Chapter 8, "Customizing Application Security,"](#) for details on capturing and storing user authentication information.)

```
// GetErrorTemplate(e: GenericException, request: HTTPRequest,
//   output customParameters: Array of NamedElement) : TextData
//
// Every exception will be processed with the same template
//
errorTemplate : TextData = new;
errorTemplate.SetValue('Movie/Errors.html');
//
// Assume that the LogonSession method returned the username in the
// credentials object, where it was added to the SessionData.
//
customParameters = new;
p : NamedElement = new;
p.Name = 'ApplicationUsername';
if request.CurrentSession <> NIL then
  p.Object = TextData(request.CurrentSession.GetSessionData(
    TextData(value='Logon-Credentials')));
else
  p.Object = TextData(value='No session');
end if;
customParameters.AppendRow(p);
//
return errorTemplate;
```

The user name is available in the `Movie/Errors.html` template as the result set variable `$(FORTE.ApplicationUsername)`.

Customizing HTTPAccessExceptions

As noted above, the `HTMLScanner.GetErrorTemplate` method performs mapping for exceptions raised during HTML template processing. This phase includes exceptions raised in Business Services. Overriding `GetErrorTemplate` effectively customizes these exceptions.

Some WebEnterprise exceptions occur outside of the `HTMLScanner` template processor, for example, errors occurring during non-template requests (“pagename” requests), HTTP parsing errors, and communications failures. Such errors can raise exceptions that require a slightly different customization.

These exceptions are caught, mapped, and formatted within the `HTTPAccess` subclass service object. By default, the `HTTPAccess.GetErrorTemplate` method applies exactly the same mapping logic as `HTMLScanner.GetErrorTemplate`. However, customizing the `HTMLScanner` method has no effect on the exceptions processed within `HTTPAccess`. Unless you also customize `HTTPAccess.GetErrorTemplate`, `HTTPAccess` exceptions will be processed through the default `htmlscanner.html`, `httpaccess.html`, and `generic.html` templates.

You customize `HTTPAccess` exception processing by customizing the `GetErrorTemplate` method in the `HTTPAccess` subclass. Since this method has the same signature as the `HTMLScanner` method, you can customize in several ways:

- copy your customized `GetErrorTemplate` method from your `HTMLScanner` subclass to your `HTTPAccess` subclass
- override the `HTTPAccess.GetErrorTemplate` method and have it directly call your `HTMLScanner.GetErrorTemplate`

Example

```
// MovieAccess.GetErrorTemplate(e: GenericException, request:
//     HTTPRequest,
//     output customParameters: Array of NamedElement) : TextData
//
return MovieScannerService.GetErrorTemplate(e, request,
customParameters);
```

- override the `HTTPAccess.GetErrorTemplate` method, implementing different custom exception logic

As a suggestion, you might use a general purpose `HTTPAccess.GetErrorTemplate` method in every `WebEnterprise` application, but a different `HTMLScanner.GetErrorTemplate` method for each application.

CAUTION When customizing exceptions with `GetErrorTemplate`, check for the existence of objects before you use them. This is particularly true when the `HTTPAccess.GetErrorTemplate` calls a customized `HTMLScanner.GetErrorTemplate`. For example, exceptions caught in `HTTPAccess` might not have their `HTTPRequest.CurrentSession` attribute set (because the error occurred before the session was associated with the request). The `GetErrorTemplate` method that processes the exception must be careful to test `CurrentSession` for a `NIL` value before using it.

Errors in Error Customization

What happens if errors occur during the mapping and formatting of exceptions? What if the error template object returned by `GetErrorTemplate` is syntactically incorrect or refers to a file which does not exist? What if my customized `GetErrorTemplate` refers to the `request.CurrentSession` object, but it is `NIL` because no session exists?

In these cases, the “last chance exception processor” is invoked. The last chance exception processor catches and dismisses the exception raised during exception processing. It then creates an HTML error response containing the original exception’s attributes. The format of this error page is fixed.

If, after customizing your error pages, exceptions are not returned using customized pages (or, for that matter, the default pages in `${FORTE_ROOT}/html/errors`), then it is likely that there is an error in the error customization. Since the last chance exception processor’s mission is to hide these errors, you can diagnose the problem by tracing handled exceptions (that is, by enabling `trc:lo:25` in the partition where either or both the `HTMLScanner` and `HTTPAccess` service objects exist). The exception caused by customization should then appear in the trace log.

Customization Examples

This chapter describes a number of desirable customizations for WebEnterprise Designer applications. These examples illustrate how information described in the previous chapters of this manual might be used in real-life situations.

The customizations in this chapter include:

- adding a Lookup reference page
- passing data with a command link
- automatically populating data on an Insert page
- adding a drop list for entering and formatting dates
- removing a JavaScript validation from a page mode
- displaying the record just inserted
- validating a whole form
- making a single field mandatory
- entering lookup information for drop lists or radio lists manually
- removing <Not Selected> and <None> from drop lists and radio lists

Introduction

The examples in this chapter are based on Tech Notes for WebEnterprise Designer that are available on the iPlanet UDS Cybersupport Web site. Check this Web site frequently for new information.

All customizations in this book start with the HTMLtutApp tutorial application, described in *Getting Started with WebEnterprise Designer*. We recommend you create your own tutorial application. Alternatively, you can import a finished version of the tutorial by following the instructions provided in Appendix A of *Getting Started with WebEnterprise Designer*.

The examples are as follows:

- **“Example: Adding a Lookup Reference Page” on page 166**
Creates a link to a list of customers from the Insert mode of the Customers page. Selecting a customer from the lookup page passes the Customer Number value back to the Insert page.
- **“Example: Passing Data with a Command Link” on page 177**
Similar to the previous example, but instead of a lookup page, uses a variable to pass the record value of the selected record to the Insert page.
- **“Example: Automatically Populating Data on an Insert Page” on page 185**
Yet another way to pass a value to the Insert page. This example creates a new Customer page and uses a WebEnterprise Designer variable to pass the value to the Insert page.
- **“Example: Adding a Drop List for Entering and Formatting Dates” on page 191**
Creates three drop lists and uses stored values for date entry.
- **“Example: Removing a JavaScript Validation from a Page Mode” on page 198**
Removes an inappropriate field constraint from a Search mode template.
- **“Example: Displaying the Record Just Inserted” on page 203**
Moves the most recently inserted record to the top of the result set list, so that it is displayed immediately after it is entered.
- **“Example: Validating a Whole Form” on page 206**
Uses a JavaScript script to validate a form. Also provides a JavaScript boilerplate for further use.

- [“Example: Making a Field Mandatory” on page 213](#)
Uses TOOL customization to force a user to enter data in a field.
- [“Drop List or Radio List Example: Entering Lookup Information Manually” on page 217](#)
Shows how to enter displayed values and stored values in drop lists or radio lists without referring to a business class.
- [“Drop List and Radio List Example: Removing <Not Selected> and <None>” on page 223](#)
Shows how to remove unwanted default values from drop lists and radio lists and specify new ones.

Methods for Editing Generated Files

The Page Handler Customization Wizard provides convenient and organized access to any files generated for your HTML application model, including TOOL files and HTML template files. When the method you want to customize does not have a customization point defined for it in the Customization Wizard, you have to customize manually, using the Method Workshop.

Using the Page Handler Customization Wizard

Complete information on using the Wizard is provided in [“Customizing a Generated HTML Template” on page 72](#). Before you can edit generated files with the Customization Wizard, you must set the FORTE_WW_HTMLEDITOR environment variable with the full pathname of the editor of your choice. For example, if you were using the HTML editor FrontPage:

```
FORTE_WW_HTMLEDITOR D:\frontpage\bin\fpeditor.exe
```

For information on setting these variables, see the online Help topic for the variable. For information on using shortnames for file specification on Windows NT, see the help topic, “Use a shortname for the Default Browser field.”

Customizing TOOL Methods Manually

Chapters 1 and 2 of this manual offer general guidance concerning which methods to customize for many types of customizations. Once you know which method to override or customize, then you edit the method in the Method Workshop. If your customization is to a page handler class, you must first create a customizable version of the class. (Refer to [“Creating Customizable Classes” on page 65](#).)

For example, “[Step 2. Override the BeforeInsert Method](#)” on page 195 describes a TOOL method customization that creates a BeforeInsert method in the CustomerOrderHandler class that overrides the ExpressClassHandler.BeforeInsert method. You can use the following alternative to the Customization Wizard:

- **To customize a method of an ExpressHandlers class in a HTMLtutAppHandlers class**
 1. In the Project Workshop for ExpressHandlers, open the Class Workshop for the ExpressClassHandlers class.
 2. Locate the BeforeInsert method in the Class Workshop.
 3. In the Project Workshop for HTMLtutAppHandlers, open the CustomerOrderHandler class.
 4. Drag the BeforeInsert method (in Step 2) into the Class Workshop for the CustomerOrderHandler class.
 5. Open the new BeforeInsert method.
 6. Proceed with the customizations described in “[Step 2. Override the BeforeInsert Method](#)” on page 195.

Customizing Generated HTML and Text Files Manually

[Chapter 3, “Customizing Generated HTML Templates,”](#) describes all generated page handler templates. These are all found in the `${FORTE_ROOT}/DocumentRoot/html_model` directory. You can edit the HTML files with the Page Handler Customization Wizard, but if you want to edit them manually, you can use either an HTML editor or a text editor to edit files with an .html extension.

Example: Adding a Lookup Reference Page

In the HTMLtutApp tutorial application, adding a new order to a customer record is somewhat clumsy and counter-intuitive. You navigate to the customer you wish to add an order to, but when you choose Insert, the page displays with all fields empty. Thus, there is no connection between the customer record you are viewing and the Insert page. This example offers a solution.

What This Example Does

This modifies the HTMLtutApp tutorial application, adding a lookup link from the Insert mode of the CustomerOrder page to a customer list. Selecting a customer from this list automatically passes the selected customer's number back to the Insert page.

The lookup link on the Insert page looks like this:

Figure 7-1 Lookup Link on Customer Order Insert Page

Insert CustomerOrder

Customer Number [View Customer](#) — Lookup link

Order Number

Requested Date

INSERT


Clicking on the lookup link displays a master list of customers:

Figure 7-2 Customer List Page

Customer List

	Customer Number	Name	Address	Phone
Use	1	Jane Doe	10 10th St.	332-1234
Use	2	J. Smith	11 11th Ave.	332-4321
Use	3	R. Simmons	111 A St.	443-4321

Use links


3 records

Clicking on the Use link associated with the desired customer redisplay the Customer Order Insert page with the selected customer's number showing in the appropriate field.

Figure 7-3 Customer Number Returned with the Insert Page

Insert CustomerOrder

Customer Number: 3 [View Customer](#)

Order Number:

Requested Date:

INSERT

Creating a Lookup Link

This section describes how to add a lookup link to the HTMLtutApp tutorial application. The *general* steps of this procedure are as follows:

1. In the HTML application model, create a CustomerList page based on the Customer class and link it to the Customer Order page; specify that the new page will start in Search mode.
2. Run the application and capture the URL of the Insert mode of the CustomerList page.
3. Customize the Insert template of the CustomerOrder page, adding a lookup link to the URL captured in Step 2, and an iPlanet UDS variable for the selected order's CustomerNumber value.
4. Add a link to the Data template of the CustomerList page to its Insert page, and pass the CustomerNumber of the selected customer with the link.
5. Remove the link between the CustomerOrder class and CustomerList class so that it only appears on the Insert mode of the CustomerOrder page.

Step 1. Add a Reference Page to the HTMLtutApp Model

This section describes how to modify the HTMLtutApp model as the first step in creating a lookup link.

► To set up the HTMLtutApp model for a lookup link

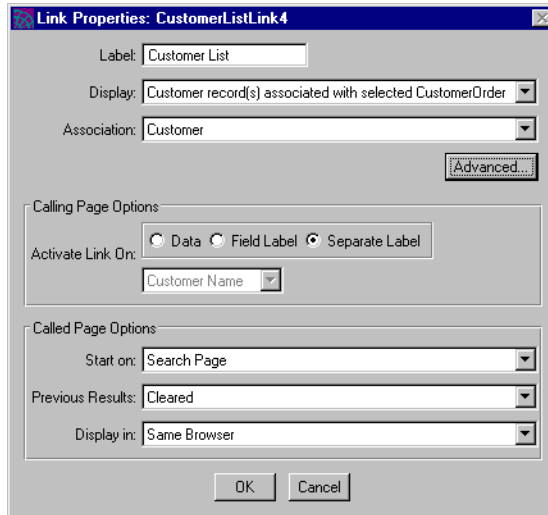
1. Start the HTML Application Model Workshop for the HTMLtutApp application.
2. Create a new list page based on the Customer class.
3. Enter CustomerList for the HTML Page Name, and make the title Customer List.

The Page Options page of the Page Wizard should look like this:

4. Create a link between the CustomerOrder page and this page.
5. Double-click on the link to display the Link Properties dialog.

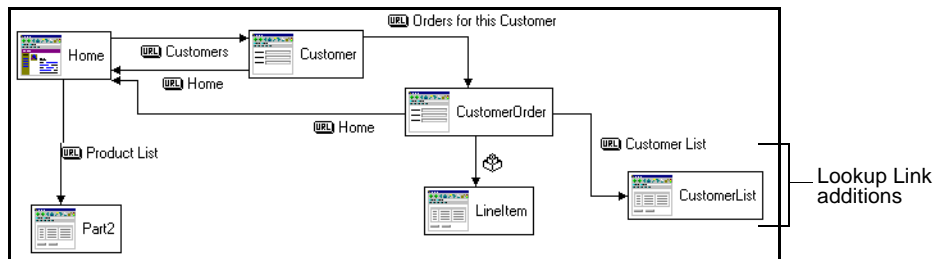
6. Name the link Customer List and set the Called Page Options to start on the Search Page.

The Link Properties dialog should look like this:



7. Click OK to apply your changes and dismiss the dialog.

Your model should look like this:



Step 2. Capture the Search Page URL

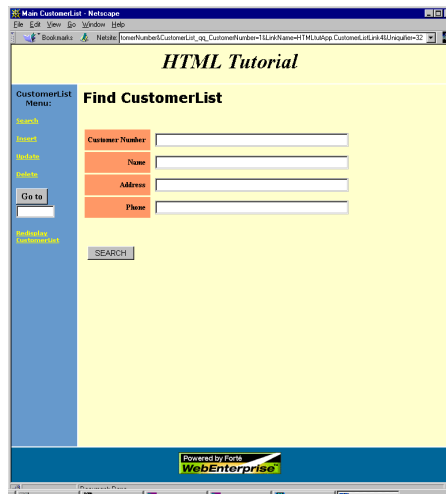
In this section, you run the application and capture the URL of the search mode of the CustomerList page.

- **To capture the URL of the Search page mode**

1. Run the HTMLtutApp application.

2. From the Home page, click the Customers link to get to the Tip-a-Canoe Customers page.
3. From the Customers page, click the Orders for this Customer link to get to the Customer Orders nested page.
4. Under the Customer Order portion of the page is the new Customer List link; click on it.

The search mode of the CustomerList page should appear, as shown here.



5. In the data section of the page, right-click to bring up the menu, and choose Open Frame in New Window.

This step is necessary, because the HTMLtutApp application uses a page design with frames. If you do not perform this step when copying the URL, later, when you run the application, you will get the frameless version.
6. Copy the URL (to the clipboard) in the new window and exit the application.

Step 3. Create a Link with the Captured URL

In this section, you use the URL string you copied in the previous section to create a link in the CustomerOrder Insert template. You will also add a variable for the CustomerNumber.

Before beginning this section, please review [“Methods for Editing Generated Files”](#) on page 165.

► **To customize the CustomerOrder page**

1. Using the HTML-editing option of your choice, open the `Insert_CustomerOrder.html` file.
2. Find the section defining the Customer Number field.

The original code looks like this:

```
<tr>
  [Begin definition of Customer Number field]
  <th class="labelmodify" style="width:20%">
    Customer Number
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="CustomerOrder_qq_CustomerNumber"
size=40>
  </td>
  [End definition]
</tr>
```

3. Add an iPlanet UDS variable for the CustomerNumber field:
`value="$$ (FORTE.CustomerNumber) "`
4. Also add an HREF line with the saved URL from the previous section.

The URL will look like this (this is all one unbroken string):

```
http://your_web_server/web.forte?ServiceName=HTMLtutAppService
&TemplateName=HTMLtutApp/Search_CustomerList.html&ReturnTemplate=H
HTMLtutApp/Display_CustomerList.html&Action=Clear&Action=Search
&Selection=CustomerList_qq_CustomerNumber
&CustomerList_qq_CustomerNumber=1
&LinkName=HTMLtutApp.CustomerListLink4&Uniquifier=$$ (FORTE.UniqueID) "
```

When you paste the URL into the HTML code, substitute the beginning of the string up to the first question mark:

```
http://your_web_server/web.forte?ServiceName...
```

with the iPlanet UDS ExecURL variable:

```
$$ (FORTE.ExecURL)?ServiceName...
```

5. And also substitute the final variable:

Uniquifier=32

with the iPlanet UDS UniqueID variable:

Uniquifier=\${FORTE.UniqueID}

6. Then add a label (View Customer) for the link at the end of the line.

Your final code should look like this (changes in **bold**):

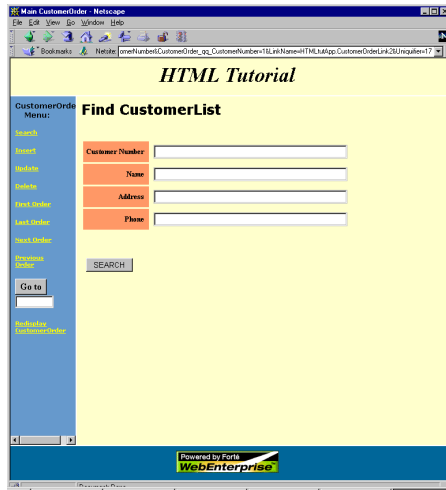
```
<tr>
[Begin definition of Customer Number field]
  <th class="labelmodify" style="width:20%">
Customer Number
</th>
<td class="dataform" style="width:80%">
[New iPlanet UDS variable added]
  <input type="text" name="CustomerOrder_qq_CustomerNumber" size=40
value="${FORTE.CustomerNumber}">
[Modified URL link (this is one line)]
<a href="${FORTE.ExecURL}?ServiceName=HTMLtutAppService&TemplateName=HTMLtutA
pp/Search_CustomerList.html&ReturnTemplate=HTMLtutApp/Display_CustomerList.htm
l&Action=Clear&Action=Search&Selection=CustomerList_qq_CustomerNumber&Customer
List_qq_CustomerNumber=1&LinkName=HTMLtutApp.CustomerListLink4&Uniquifier=${F
ORTE.UniqueID}">
[Add link label]
View Customer
</a>
[End definition]
</td>
```

7. Close the Insert_CustomerOrder.html file and save your changes.
8. Test your work by repeating **Step 1 to Step 4 on page 171**.
9. In the Customer Orders nested page, click on the Insert command in the command list (left panel).

The Insert mode page shown in **Figure 7-1 on page 167** shows the new View Customer link you just created.

10. Click on the View Customer link.

The CustomerList page opens in Search mode:



11. Exit the application.

Step 4. Pass the Selected Field Value

In this section, you modify the Data template of the Customer List page, adding a link to the CustomerOrder Insert page and passing the Customer Number of the selected customer as a parameter, so that it displays in the Insert page.

► To customize the Data template of the Customer List page

1. Using the HTML-editing option of your choice, open the Data_CustomerList.html file.
2. Find the first HREF statement.

The original HREF statement looks like this:

```
[First HREF instance]
<a class="rowidlist" HREF="$$$(FORTE.ExecURL)?ServiceName=HTMLtutAppService&Tem
plateName=HTMLtutApp/Display_$$$(USER.TopPage).html&Action=CustomerListHandler.
SetCurrentRow&Position=$$$(listentry.qqRowNumber)&Uniquifier=$$$(FORTE.UniqueID)
"
target="_fortedisplay"> $$$(listentry.qqRowNumber)</a>
```

Modify the first HREF statement as follows (changes are in **bold**):

[First HREF instance specifies the target of]
`<a class="rowidlist" HREF="$$$(FORTE.ExecURL)?ServiceName=HTMLtutAppService&Template=HTMLtutApp/Insert_CustomerOrder.html&ReturnTemplate=HTMLtutApp/Display_CustomerOrder.html&Uniquifier=$$$ (FORTE.UniqueID)&CustomerNumber=$$$ (listentry.CustomerList_qq_CustomerNumber)"`

[the new "Use" link]
`target="_fortedisplay">Use`

To summarize the changes:

- `&TemplateName=HTMLtutApp/Insert_CustomerOrder.html` identifies the template as the Insert mode of CustomerOrder
- `&ReturnTemplate=HTMLtutApp/Display_CustomerOrder.html` specifies the Display mode of the same page as the return template
- `&Uniquifier=$$$ (FORTE.UniqueID)` defeats the caching mechanism and retrieves a “fresh” page
- `&CustomerNumber=$$$ (listentry.CustomerList_qq_CustomerNumber)` passes the selected CustomerNumber value as a parameter
- Use—Adds a new label for each row

For more information on iPlanet UDS tags, see [Chapter 3, “Customizing Generated HTML Templates.”](#)

3. Exit the `Data_CustomerList.html` file and save your changes.

Testing Your Work Before the Final Step

Now it is time to test your work to verify that you made no mistakes and the customization works correctly. After testing, you will perform the final step, in [“Step 5. Remove the CustomerOrder-CustomerList Link,”](#) which follows this section.

► **To test the work you have done so far**

1. Run the HTMLtutApp application.
2. Navigate to the Customer Orders page and click on the `Insert` command in the command list (left panel).
3. In the Insert mode of the Customer Orders page, click on the View Customer link.

This takes you to the CustomerList page in Search mode.

4. Click on the Search button.

This takes you to the CustomerList page in Display mode, shown in [Figure 7-2 on page 167](#).

5. Click on the Use link of any customer record.

You are returned to the Insert mode of the Customer Orders page, as shown in [Figure 7-3 on page 168](#), with the selected customer's Customer Number showing.

6. Add an order number and date.

Enter the date in the format dd-mmm-yyyy hh:mm:ss (for example, 05-Mar-1999 00:00:00).

7. Click the Insert button.

You are returned to The Customer Order page.

8. Click on `Last Order` in the command list.

Your new order appears. (You may have to reload the page to display all the fields.)

Customer Orders	
Customer Name	Jane Doe
Customer Number	1
Order Number	10010
Requested Date	05-Mar-1999 00:00:00

record 3 of 3 [Home](#) [Customer List](#)

9. Exit the application.

Step 5. Remove the CustomerOrder-CustomerList Link

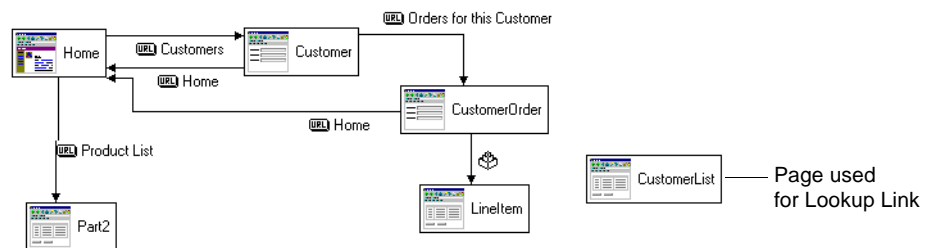
In this section, you remove the link you created in Step 1. You do this because you use the lookup link only when you are inserting a new customer order, as set up in Step 3. Therefore, you only want the link to appear on the Insert mode of the CustomerOrder page. (If the link remains, it is visible on all modes of the CustomerOrder page.)

► To finish the customization

1. Open the HTML Application Model Workshop for the HTMLtutApp application.
2. Delete the link between the CustomerOrder page and the CustomerList page.

Your model should look like [Figure 7-4](#).

Figure 7-4 HTMLtutApp Model Modified for a Lookup Link



Usage Recommendations

To use this link properly, always select the customer first, by using the Customer View link, then add the other customer order data on the Insert page. If you fill in the other fields before using the Customer View link, all your data will be lost when you return to the Insert page, except for the customer number.

Example: Passing Data with a Command Link

This customization is similar to the previous one ([“Example: Adding a Lookup Reference Page” on page 166](#)), in that you customize the application to pass a value from a selected record to an Insert page. In this example, there is no lookup list to select from; instead, the primary key value of the selected order is passed to the Insert page, so that the new record will automatically be inserted into the selected order.

What This Example Does

This example modifies the HTMLtutApp tutorial application to add data to a command link. As in the uncustomized HTMLtutApp tutorial, the user can choose a customer from the Customer page, then navigate to the master-detail page that displays orders for that customer. At this point, the user can browse through the orders until she comes to the one she wants to insert a line item to:

Figure 7-5 Selected Customer on Master-Detail Page

The screenshot shows a web application interface. At the top, there's a section titled "Customer Orders" with a light blue background. It contains four rows of data: "Customer Name" (R. Simmons), "Customer Number" (3), "Order Number" (1005), and "Requested Date" (01-Mar-1995 00:00:00). A bracket on the right side of this section is labeled "Selected customer order". Below this, there's a "Product Description" section with a table. The table has columns: Order Number, Item Number, PN, Description, Price, and Quantity. There are two rows of data: Row 1: Order Number 1005, Item Number 1, PN 90002, Description "Fiberglass Canoe", Price 59.99, Quantity 1. Row 2: Order Number 1005, Item Number 3, PN 90003, Description "5 Foot Oar", Price 9.99, Quantity 2. Below the table, there are several command links: "Search", "Insert", "Update", "Delete", and "Redisplay LineItem". A line points from the "Insert" link to the text "Insert command for Line Item" on the right. At the top left of the page, it says "record 1 of 2" and "Home".

The user clicks on the `Insert` command under the Product Description section (the nested `LineItem` page) to invoke the `LineItem Insert` page, which displays with the order number showing:

Figure 7-6 `LineItem Insert` Page Showing Passed Value

The screenshot shows a web application interface titled "Insert LineItem". It has a yellow background. There are four input fields, each with a label to its left: "Order Number" (with the value "1005" entered), "Item Number", "PN", and "Quantity". Below the input fields is a button labeled "INSERT".

Thus, the new line item will be added to the order selected by the user.

Creating the Customization

This section describes how to customize the HTMLtutApp tutorial application to add data to a command link. The *general* steps of this procedure are as follows:

1. Make sure the Insert and Update commands are defined for the page in question.
2. Add a variable to the Data mode of the page to hold the value.
3. Enclose the whole link definition in a “forte iterate” loop to single out one instance of the data.
4. Populate the appropriate field on the Insert page with the incoming data.

Step 1. Add Insert and Update Commands (If Required)

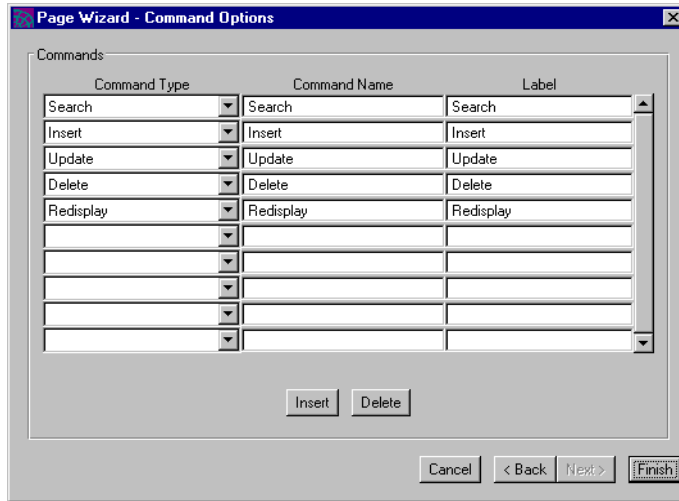
The Insert mode of the LineItem page must have `Insert` and `Update` commands defined for it. These are already defined in the HTMLtutApp tutorial application, as described in *Getting Started with WebEnterprise Designer*. However, if you have not finished the tutorial, or are modifying your own application, make sure the page in question has these commands.

► **To add the Insert and Update commands to a page (if necessary)**

1. In the HTML Application Model Workshop for the HTMLtutApp application, open the Page Wizard for the LineItem page by double-clicking on the page.
2. Click Next until the Commands page displays.

3. If there are no Insert and Update commands on the page, add them.

The Commands page should look like this (other commands can be missing, but it should at least have the Insert and Update commands):



4. Click Finish to apply your changes and close the Page Wizard.
5. Generate code for the model.

Step 2. Add a Variable to Hold the Value

In this section, you will add code to the `Data_LineItem.html` template that adds a variable to hold the order number data to the Insert command link.

Before beginning this section, please review [“Methods for Editing Generated Files” on page 165.](#)

➤ **To add a variable to the Insert command link of the LineItem page**

1. Using the HTML-editing option of your choice, open the `Data_LineItem.html` file.
2. Find the section that defines the link to the Insert mode of the page.

The original code looks like this:

```
<td>
[Start link definition]
<a class="alink" HREF="$$ (FORTE.ExecURL)?ServiceName=HTMLtutAppService&Templat
eName=HTMLtutApp/Insert_LineItem.html&ReturnTemplate=HTMLtutApp/Display_$$ (USE
R.TopPage).html&Uniquifier=$$ (FORTE.UniqueID)" target="_fortedisplay">
[End link definition]
  Insert</a>
</td>
```

3. Add a variable to hold the order number data.

Example

```
&ORDERNUMBER=$$ (listentry.LineItem_qq_OrderNumber)
```

where:

- ORDERNUMBER is a local variable
- `listentry.LineItem_qq_OrderNumber` is an iPlanet UDS variable that stores the value of ORDERNUMBER

The customized code should look like this (changes in **bold**):

```
<td>
[Start link definition]
<a class="alink" HREF="$$ (FORTE.ExecURL)?ServiceName=HTMLtutAppService&Templat
eName=HTMLtutApp/Insert_LineItem.html&ReturnTemplate=HTMLtutApp/Display_$$ (USE
R.TopPage).html&Uniquifier=$$ (FORTE.UniqueID)&ORDERNUMBER=$$ (istentry.LineItem
_qq_OrderNumber)" target="_fortedisplay">
[End link definition]
  Insert</a>
</td>
```

4. Continue on to the next section without closing the `Data_LineItem.html` file.

Step 3. Single Out One Instance of the Data

Because `LineItem` is a list page that could have multiple line items, you need some way to single out a single instance of the data. In this section, you will do this by using a “forte iterate” loop to the Insert link definition.

► **To add a forte iterate loop**

1. Enclose the code you entered in the previous section in a “forte iterate” loop.

The syntax of the forte iterate loop is:

```
[Begin forte iterate loop]
<!--<?forte iterate listentry entry.list_LineItem start="1" max="1">-->
    Your application logic here

[End forte iterate loop]
<!--<?/forte iterate listentry>-->
```

Your final modified code should look like this (changes in **bold**):

```
<td>

[Begin forte iterate loop]
<!--<?forte iterate listentry entry.list_LineItem start="1" max="1">-->

[Start link definition]
<a class="alink" href="$$ (FORTE.ExecURL)?ServiceName=HTMLtutAppService&Templat
eName=HTMLtutApp/Insert_LineItem.html&ReturnTemplate=HTMLtutApp/&ORDERNUMBER=$
$(listentry.LineItem_qq_OrderNumber)"Display_$$ (USER.TopPage).html&Uniquifier=
$$ (FORTE.UniqueID) target="_fortedisplay">

[End link definition]
    Insert</a>

[End forte iterate loop]
<!--<?/forte iterate listentry>-->
</td>
```

2. Close the `Data_LineItem.html` file and save your changes.

Step 4. Populate the Order Number with Incoming Data

In this section, you modify the Insert template of the LineItem page to display the value stored in the CUSTOMERORDER variable that you added to the link in [“Step 2. Add a Variable to Hold the Value”](#) on page 180.

► To customize the Insert mode of the LineItem page to display the data

1. Using the HTML-editing option of your choice, open the `Insert_LineItem.html` file.
2. Find the section that defines the OrderNumber field.

The original code looks like this:

```
[Begin table definition of Order Number field]
<tr>
  <th class="labelmodify" style="width:20%">
    Order Number
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="LineItem_qq_OrderNumber" size=40>
  </td>
</tr>
[End definition]
```

3. Add code to populate Customer Number with values:

```
value=${FORTE.ORDERNUMBER}
```

where ORDERNUMBER is the variable defined in [Step 3](#) of the previous section.

Your code should look like this (changes are in **bold**):

```
[Begin table definition of Order Number field]
<tr>
  <th class="labelmodify" style="width:20%">
    Order Number
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="LineItem_qq_OrderNumber" size=40>
  </td>
</tr>
```

```
[New variable]
  value=${FORTE.ORDERNUMBER}>
</td>

End definition
</tr>
```

4. Close the `Insert_LineItem.html` file and save your changes.

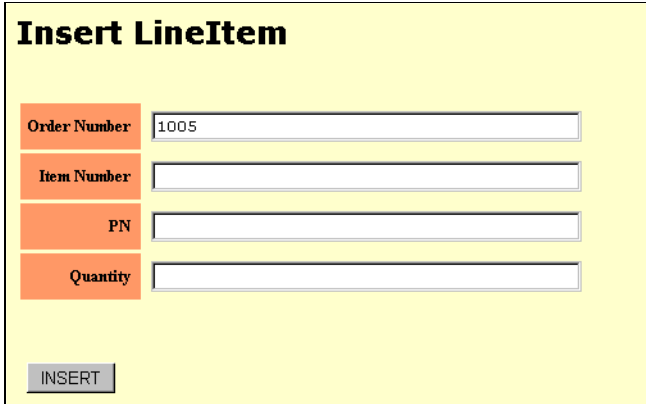
Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test the customization**

1. Run the HTMLtutApp application.
2. On the Home page, navigate to the Tip-a-Canoe Customers page.
3. Browse through the customer records and click the Orders for this Customer link for some customer.
4. On the Customer Orders nested page, click the Insert link below the Product Description.

The LineItem Insert page should display with the correct Order Number showing:



Order Number	<input type="text" value="1005"/>
Item Number	<input type="text"/>
PN	<input type="text"/>
Quantity	<input type="text"/>

5. Add an Item Number, a PN, and a Quantity.

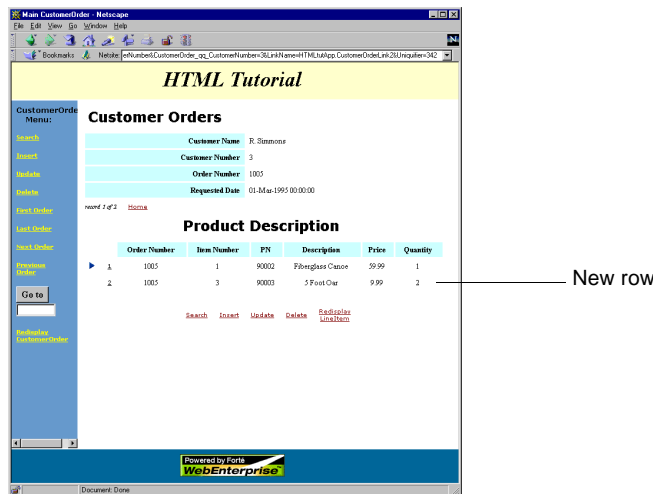
To test your work accurately, use values for Item Number and PN that exist in your database. For example (from the tutorial database):

Item Number: 1

PN: 90003

Quantity: 3

6. Click the Insert button to return to the Customer Orders page.
7. Reload the page to populate the rest of the fields. If you entered the values suggested in Step 5, your page should look like this:



Example: Automatically Populating Data on an Insert Page

This example describes a technique different from the previous example (“[Example: Passing Data with a Command Link](#)” on page 177) for passing a customer number value from a selected Customer to the Insert mode of a CustomerOrder page. This example creates an additional CustomerOrder page, then uses a variable created by WebEnterprise Designer to pass the value to the Insert and Update modes of this page.

What This Example Does

In the uncustomized HTMLtutApp tutorial, if a user wants to add an order to a particular customer, he invokes the Insert mode of the Customers page, which displays with all fields blank. If he does not remember the number of the particular customer he selected, he must return to the Customers page for the information.

In this customization, you add a new CustomerOrder page, called NewOrder, to the application and link it to the Customers page. When you run the application and navigate to the Tip-a-Canoe Customers page, you activate the link to the new page, which goes to the Insert mode of the page and passes the customer number value with the link. [Figure 7-7](#) shows an example display:

Figure 7-7 Insert Mode of NewOrder Page

The screenshot shows a web form titled "Insert NewOrder" with a yellow background. It contains three input fields with orange labels: "Order Number", "Customer Number", and "Requested Date". The "Customer Number" field contains the value "2". Below the fields is a grey "INSERT" button. A line points from the "Customer Number" field to the text "Selected customer's number" on the right.

Creating the Customization

This section describes how to customize the HTMLtutApp tutorial application to pass a CustomerNumber field value to the Insert mode of a CustomerOrder page. The *general* steps of this procedure are as follows:

1. Add a new business class page, the NewOrder page, to the model, based on the CustomerOrder business class; link it to the Customers page.
2. Customize the generated NewOrder Insert template to populate the CustomerNumber field with an incoming value.

Use a variable automatically created by WebEnterprise Designer to hold the value.

Step 1. Add a NewOrder Page to the HTMLtutApp Model

This section describes how to modify the HTMLtutApp model as the first step in automatically populating an Insert page.

► To modify the HTMLtutApp model

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Create a form page based on the CustomerOrder class.
3. On the Page Options page of the Page Wizard, enter:

Business Class: `CustomerOrder`

HTML Page Name: `NewOrder`

(It does not matter what you enter in the Page Title field because you will only access the Insert mode of this page, which does not display the page title.)

4. Click Next until you reach the Field Properties page.
5. Add spaces to the field labels as shown and click Finish:

Field	Label	Formatting Routine
OrderNumber	Order Number	
CustomerNumber	Customer Number	
RequestedDate	Requested Date	

Add spaces
in the field labels

6. Add a command link between the Customer page and the NewOrder page.

7. Open the Link Properties dialog and enter or set these values:

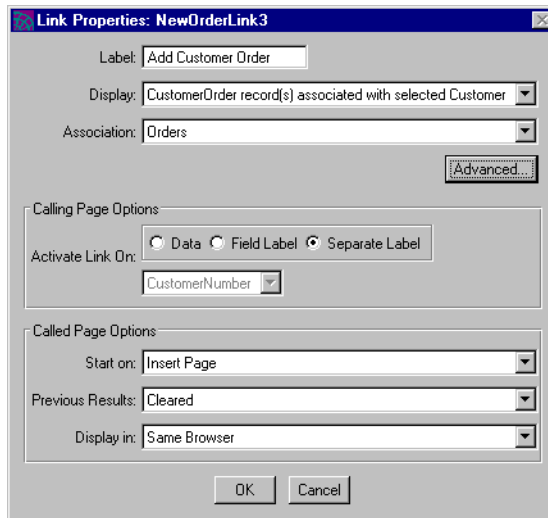
Label: Add Customer Order

Display: CustomerOrder record(s) associated with selected Customer

Activate Link On: Separate Label

Start on: Insert Page

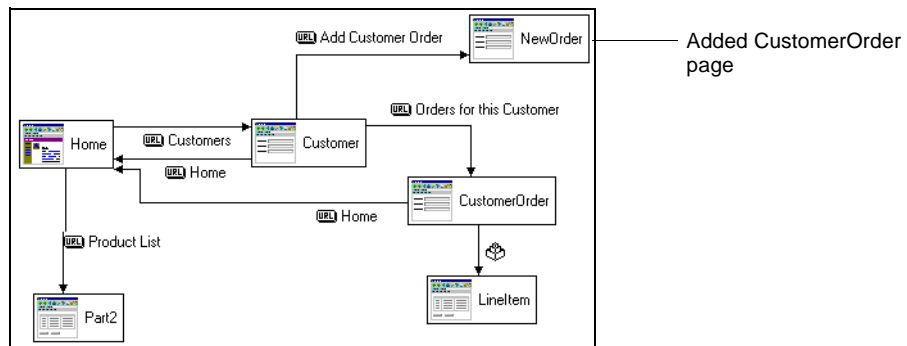
The Link Properties dialog should look like this:



8. Click OK to apply these values and close the dialog.

Your model should look like this:

Figure 7-8 HTMLtutApp Model with Additional CustomerOrder Page



9. Generate the application code.

At this point, you can test the application and insert orders by clicking the Add Customer Order link. However, none of the fields are populated with values yet, so you have to enter them manually. The next step shows you how to populate the Customer Number field with the value from the selected customer.

Step 2. Add a Variable to Hold the Value

As part of the generated HTML code, WebEnterprise Designer creates the variable you will use to hold the CustomerNumber value. This variable is `NewOrder_qq_CustomerNumber`, which is created in the Data template of the Customer page as part of the coding required to satisfy the link's display option (CustomerOrder record(s) association with Customer). The code in `Data_Customer.html` page looks like this:

```
<td>
  <!--<?forte if CustomerHandler.HasCurrentRow>--><a class="alink"
href="$$FORTE.ExecURL?serviceName=HTMLtutAppService&templateName=HTMLtutApp/M
ain_NewOrder.html&StartingPage=Insert&ReturnTemplate=HTMLtutApp/Display_NewO
rder.html&Action=Clear&Action=Search&Selection=NewOrder_qq_CustomerNumber&
NewOrder_qq_CustomerNumber=$$ (listentry.Customer_qq_CustomerNumber)&LinkName=
HTMLtutApp.NewOrderLink3&Uniquifier=$$ (FORTE.UniqueID) " target="_top">Add
Customer Order</a>
  <!--<?forte else>-->Add Customer Order<!--<?/forte if>-->
</td>
```

You will use the `NewOrder_qq_CustomerNumber` variable to populate the Customer Number field on the NewOrder Insert page with the Customer Number value of the customer selected on the Customer page.

Before beginning this section, please review [“Methods for Editing Generated Files” on page 165](#).

► **To populate the Customer Number value in the NewOrder Insert page**

1. Using the HTML-editing option of your choice, open the `Insert_NewOrder.html` file.
2. Find the table definition for the CustomerNumber field.

The original code looks like this:

```
[Begin table definition of Customer Number field]
<tr>
  <th class="labelmodify" style="width:20%">
    Customer Number
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="CustomerOrder_qq_ReqCustomerNumberze=40">
  </td>
</tr>
[End definition]
```

3. Add code to populate Customer Number with values:

```
value=${$(FORTE.NewOrder_qq_CustomerNumber)}
```

where `NewOrder_qq_CustomerNumber` is the variable defined in the `Data_Customer.html` template.

Your code should look like this (changes are in **bold**):

```
[Begin table definition of Customer Number field]
<tr>
  <th class="labelmodify" style="width:20%">
    Customer Number
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="LineItem_qq_OrderNumber" size=40
    [New variable]
    value=${$(FORTE.NewOrder_qq_CustomerNumber)}
  </td>
  End definition
</tr>
```

4. Close the `Insert_NewOrder.html` file and save your changes.
5. Regenerate the application's code.

Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test the customization**

1. Run the HTMLtutApp application.
2. On the Home page, navigate to the Tip-a-Canoe Customers page.
3. Browse through the customer records and choose one.
4. Click on the Add Customer Order link.

The Insert NewOrder page should display, as in [Figure 7-7 on page 186](#), with the Customer Number of the selected customer showing.

Example: Adding a Drop List for Entering and Formatting Dates

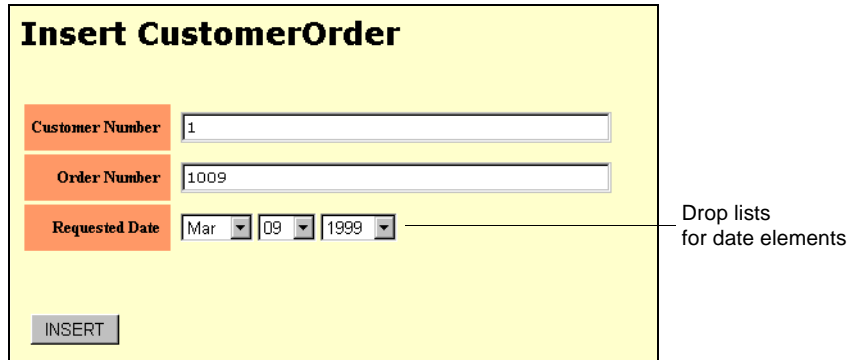
In the uncustomized HTMLtutApp tutorial application, one of the fields on the Customer Orders page requires a date value, but the application provides no help on the Insert page regarding date format. If a user enters the date in any other than a DD-*MMM*-YYYY HH:MM:SS format, she will get an error.

This customization solves this problem by creating drop lists with month, day, and year values for entry of new customer orders.

What This Example Does

In this customization, when the user invokes the Insert mode of the Customer Order page, it is displayed with drop lists in the Requested Date field for month, day, and year:

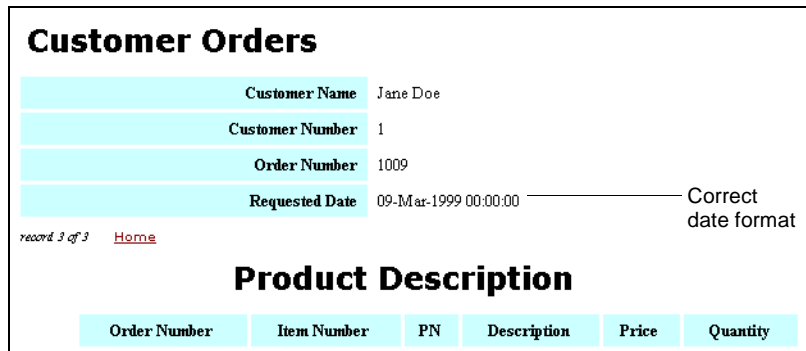
Figure 7-9 Requested Date Field Formatted with Drop Lists



The screenshot shows a form titled "Insert CustomerOrder" with a yellow background. It contains three input fields: "Customer Number" with the value "1", "Order Number" with the value "1009", and "Requested Date" with three dropdown menus showing "Mar", "09", and "1999". A line points from the text "Drop lists for date elements" to the date dropdowns. An "INSERT" button is located at the bottom left of the form.

The user fills in the other fields, then uses the drop lists to specify the requested date. When she clicks the Insert button, the Customer Order~LineItem page is displayed, showing the new order with the correct date format:

Figure 7-10 Inserted Customer Order Showing the Date



The screenshot shows a page titled "Customer Orders" with a light blue background. It displays the following information:

Customer Name	Jane Doe
Customer Number	1
Order Number	1009
Requested Date	09-Mar-1999 00:00:00

Below the table, it says "record 3 of 3" and "Home". A line points from the text "Correct date format" to the "Requested Date" value. Below the table is a section titled "Product Description" with a table header:

Order Number	Item Number	PN	Description	Price	Quantity
--------------	-------------	----	-------------	-------	----------

Creating Date-Formatting Drop Lists

This section describes how to customize the HTMLtutApp tutorial application to substitute drop lists for the date definition in the Insert template of the CustomerOrder page. The *general* steps of this procedure are as follows:

1. Customize the Insert template of the CustomerOrder page, replacing the table definition of the Requested Date field with HTML code that builds three drop lists for Month, Day, and Year.
2. Customize the BeforeInsert method to:
 - o build the DateTimedata object with the values from the drop list
 - o modify the RequestedDate attribute in the underlying business class (CustomerOrderClass)
 - o continue the normal WebEnterprise Designer processing

Step 1. Define Drop Lists for Date Elements

In this section, you will make changes to the Insert_CustomerOrder.html page to include a drop list for the three parts of the date, and provide appropriate data for each list.

Before beginning this section, please review [“Methods for Editing Generated Files” on page 165](#).

► To define drop lists for the Requested Date field

1. Using the HTML-editing option of your choice, open the Insert_CustomerOrder.html file.
2. Find the table definition for the RequestedDate field.

The original code looks like this:

```
[Begin table definition of Requested Date field]
<tr>
  <th class="labelmodify" style="width:20%">
    Requested Date
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="CustomerOrder_qq_RequestedDate" size=40>
  </td>
</tr>
[End definition]
```

3. Replace the definition line (beginning `<input type="text">`) with HTML code defining three drop lists.

Your code should look like this (changes are in **bold**):

```
[Begin table definition of Requested Date field]
<tr>
  <th class="labelmodify" style="width:20%">
    Requested Date
  </th>
  <td class="dataform" style="width:80%">
    [Begin Month drop list definition]
    <select name="Month" size="1">
      [Begin Month values]
      <OPTION value="Jan">Jan
      <OPTION value="Feb">Feb
      <OPTION value="Mar">Mar
      <OPTION value="Apr">Apr
      <OPTION value="May">May
      <OPTION value="Jun">Jun
      <OPTION value="Jul">Jul
      <OPTION value="Aug">Aug
      <OPTION value="Sep">Sep
      <OPTION value="Oct">Oct
      <OPTION value="Nov">Nov
      <OPTION value="Dec">Dec
    [End Month definition]
    </select>
    [Begin Day drop list definition]
    <select name="Day" size="1">
      [Begin Day values]
      <option>01<option>02<option>03<option>04<option>05<option>06
      <option>07<option>08<option>09<option>10<option>11<option>12
      <option>13<option>14<option>15<option>16<option>17<option>18
      <option>19<option>20<option>21<option>22<option>23<option>24
      <option>25<option>26<option>27<option>28<option>29<option>30
      <option>31
    [End Day definition]
    </select>
    [Begin Year drop list definition]
    <select name="Year" size="1">
      [Year values]
      <option>1999<option>2000<option>2001
```

```

[End Year definition]
    </select>
</td>

[End definition]
</tr>

```

4. Save and close the `Insert_CustomerOrder.html` file.

Step 2. Override the BeforeInsert Method

You want to prevent the user from inserting a blank value for the Requested Date field. Therefore, you will make changes to the page handler's `BeforeInsert` method.

In this section, you use a customization point in the Page Handler Customization Wizard to create a `BeforeInsert` method for the `CustomerOrderHandler` class (which will override the `ExpressClassHandler.BeforeInsert` method). You then customize the method as described.

► To override the BeforeInsert method for this customization

1. Open the `HTMLtutApp` model in the HTML Application Model Workshop.
2. Start the Page Handler Customization Wizard for the `CustomerOrder` page, as described in [“Customizing a Generated HTML Template” on page 72](#).
3. Open the Whole Object Operations category, then the Database Operation subcategory.
4. Double-click on the “Insert: Before sending an insert object request” customization point.

If this is the first customization you are making to this class, WebEnterprise Designer displays a confirmation dialog to expand the class hierarchy to include a customizable class for this class page. Click OK.

The Method Workshop opens, displaying an override of the `BeforeInsert` method.

5. Add the following code to the end of the method body:

```
TheMonth : Textdata = request.findNameValue('Month');
TheDay : Textdata = request.findNameValue('Day');
TheYear : Textdata = request.findNameValue('Year');
TheDate : Textdata = new(Value=TheDate.value);
TheDate.concat('-');
TheDate.concat(TheMonth.value);
TheDate.concat('-');
TheDate.concat(TheYear.value);
InsertDate : DateTimedata = new (value=TheDate.value);
Client.LogAttr(source, attr=CustomerOrderQuery.ATTR_REQUESTEDDATE);
CustomerOrderClass(source).REQUESTEDDATE.Setvalue(InsertDate);
```

6. Compile the method, close the Method Workshop and the Customization Wizard.

Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

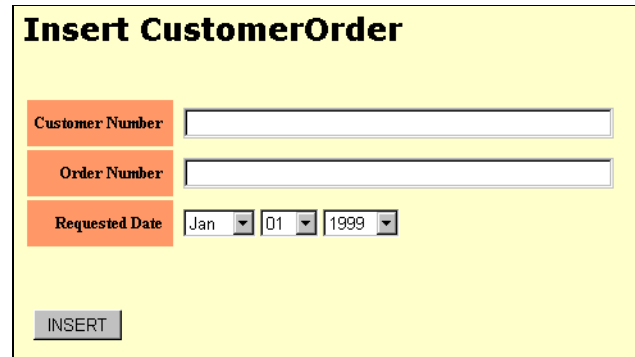
► To test the customization

1. Run the HTMLtutApp application.
2. On the Home page, navigate to the Tip-a-Canoe Customers page.
3. Browse through the customer records and click the Orders for this Customer link for some customer.

Remember the customer number this customer.

4. Click the `Insert` command in the left pane of the page.

The Customer Order Insert page is displayed and should show the new date drop lists.



Insert CustomerOrder

Customer Number

Order Number

Requested Date Jan 01 1999

INSERT

5. Enter the customer number from Step 4, and a fictitious order number.

Do not enter an order number from the tutorial database (1002-1006), because it will raise an error.

6. Use the drop lists to enter a date and click the Insert button.

The nested Customer Order~LineItem page is displayed.

7. Navigate to the new order and reload the page.

The new order should resemble [Figure 7-10 on page 192](#).

Example: Removing a JavaScript Validation from a Page Mode

WebEnterprise Designer allows JavaScript for field validations, and provides several sample validations for your use. Users can also create their own JavaScript validations. All JavaScript validations must exist in the `$FORTE_ROOT/userapp/http/cl0/scripts/validate` directory. The tutorial chapter in *Getting Started with WebEnterprise Designer* describes how to add a JavaScript validation as a possible enhancement to the HTMLtutApp application.

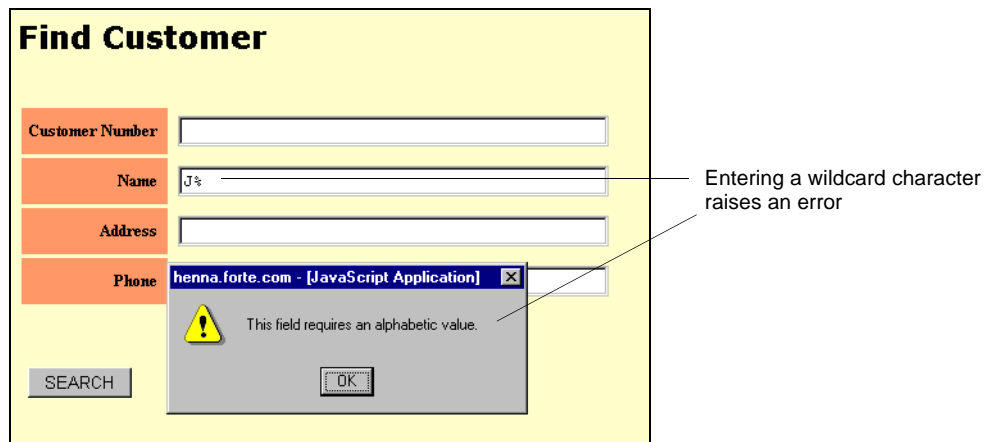
When you generate code for an application that has a field validation, WebEnterprise Designer performs the following actions:

- generates a `Scripts_bus_class_page.html` file that contains all JavaScript functions for the page, including:
 - invoking the selected JavaScript
 - adding a selected field to the Web page
- modifies the Insert, Search, and Update templates of `bus_class_page` page to call the JavaScript functions (from the `Scripts_bus_class_page.html` file)

Sometimes you do not want the validation applied to all the pages by default. This customization shows you how to remove the validation from a specific page mode template.

What This Example Does

This example modifies the HTMLtutApp application by applying the `IsAlphabetic` validation to the Customer Name field of the Customer page. This validation is a JavaScript script that enforces that only alphabetical characters can be entered into the field. This is a reasonable restriction when inserting or updating a name field, but presents an inconvenience on the search page, where iPlanet UDS normally allows the percent sign (%) as a wildcard character. If the `IsAlphabetic` validation is enforced on the Search page, this character will raise an error

Figure 7-11 IsAlphabetic Validation Error

This customization shows you how to avoid this problem by deleting the unwanted validation from the Search template of the Customer page.

Creating the Customization

This section describes how to add a validation to a field in the HTMLtutApp tutorial application, and then how to remove it from the Search template of that page. The *general* steps of this procedure are as follows:

1. Add the IsAlphabetic JavaScript validation provided by WebEnterprise Designer to the Name field of the Customer page.
2. Modify the Search template of the Customer page, commenting out the line that invokes the validation script and substituting a plain field definition.

Step 1. Apply the JavaScript to a Customer Page Field

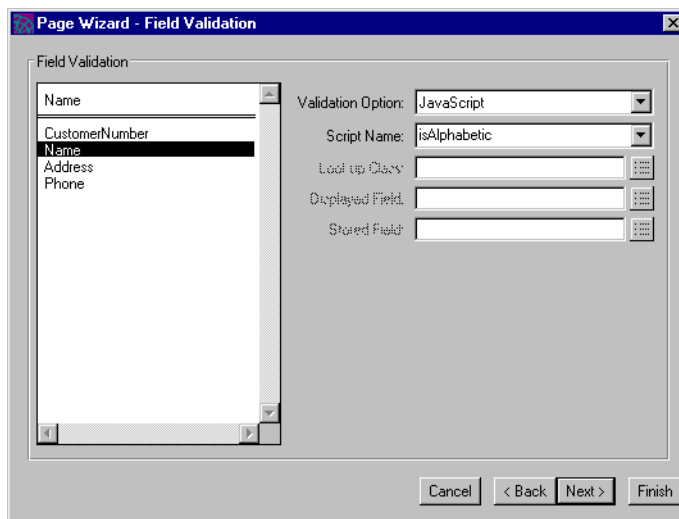
In this section, you apply the IsAlphabetic validation to the Name field of the Customer page.

► To set a validation on the Name field of the Customer page

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Double-click on the Customer page to open the Page Wizard.
3. Click Next until you reach the Field Validation page.

4. Select Name in the field name window.
5. Select JavaScript from the Validation Option's drop list.
6. Select IsAlphabetic from the Script Name's drop list.

The Field Validation page should look like this:



7. Click Finish to apply your changes and close the Page Wizard.
8. Generate the application's code.

At this point, you can run the application and test the validation by attempting to type a non-alphabetic character in the Name field on an insert, update, or search page. Hint: the validation's warning message pops up when you attempt to *leave* the field after typing something in, including tabbing to the next field, or clicking the cursor anywhere outside the field.

Step 2. Remove the JavaScript Validation from a Template

This section describes how to remove the JavaScript validation from the Search template of the Customer page. Before beginning this section, please review ["Methods for Editing Generated Files"](#) on page 165.

- **To remove the validation from the search template**
 1. Using the HTML-editing option of your choice, open the `Search_Customer.html` file.

2. Locate the table definition for the Name field.

The original code looks like this:

```
[Begin table definition of Name field]
<tr>
  <th class="labelmodify" style="width:20%">
    Name
  </th>
  <td class="dataform" style="width:80%">

[Call to JavaScript validation]
  <script language="JavaScript">
    Select_Customer_qq_Name( 'Customer_qq_Name' , 'Name' , 'qqNone' );

[End call]
  </script>
</td>

[End definition]
</tr>
```

3. Comment out the JavaScript call.

4. Add a new line to add the Name field to the Search page.

Your modified code should look like this (changes in **bold**):

```
[Begin table definition of Name field]
<tr>
  <th class="labelmodify" style="width:20%">
    Name
  </th>
  <td class="dataform" style="width:80%">

[Begin comment]
  <!--
  <script language="JavaScript">
    Select_Customer_qq_Name( 'Customer_qq_Name' , 'Name' , 'qqNone' );
  </script>

[End comment]
  -->

[Field definition added]
  <input type="text" name="Customer_qq_Name" size=40>
</td>
```

```
[End definition]
</tr>
```

5. Close the `Search_Customer.html` file and save your changes.
6. Regenerate the application's code.

NOTE The complete `Search_Customer.html` file is provided as an example in [“Example: Customizing a Field on a Search Page”](#) on page 106 of Chapter 3, [“Customizing Generated HTML Templates.”](#)

Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test the customization of the JavaScript validation**

1. Run the `HTMLtutApp` application.
2. Navigate to the Tip-a-Canoe Customers page.
3. To test that the validation still works on a page not customized, click `Insert` in the command list.
4. Enter some non-alphabetic characters and press the `Tab` key.

The validation's warning message should appear:

Figure 7-12 Validation Working on Insert Page

The screenshot shows a web form titled "Insert Customer" with a yellow background. It contains four input fields with orange labels: "Customer Number", "Name", "Address", and "Phone". The "Name" field contains the text "Hen3ry". A modal dialog box is open over the "Phone" field, displaying a yellow warning triangle icon and the message "This field requires an alphabetic value." with an "OK" button. An "INSERT" button is visible at the bottom left of the form.

5. To test that the validation was removed from the search page, return to the Customers page and click `Search` in the command list.
6. Enter a non-alphabetic character in the Name field, for example `Hen3ry`, and press the Tab key.

If you entered `Hen3ry`, you are automatically returned to the Customers page, showing that two records were returned (there are two customers with names beginning with "H" in the HTMLtutApp database). This demonstrates that your customization of the search page is working, and the validation was removed.

Example: Displaying the Record Just Inserted

When a user inserts a new record, by default WebEnterprise Designer places it at the end of the result set. When the display page is redisplayed, the record just entered is at the end of the list and frequently not visible. In many cases, it would be more convenient for the user to see the record he just entered immediately displayed on the display page. This customization causes this to happen.

What This Example Does

This example uses the HTMLtutApp tutorial application, customizing the `DoInsert` method to specify that the first visible row will be the newly-inserted record.

Thus, when you enter a new record:

Figure 7-13 Entering a New Customer Record

Insert Customer	
Customer Number	4
Name	Alfred E. Newman
Address	123 Main St.
Phone	456-1234
<input type="button" value="INSERT"/>	

After you click the Insert button, this record is immediately displayed on the Customer display page:

Figure 7-14 New Record Displayed

Tip-a-Canoe Customers	
Customer Number	4
Name	Alfred E. Newman
Address	123 Main St.
Phone	456-1234
record 4 of 4 Home Orders for this Customer	

Creating the Customization

This section describes how to use the Customization Wizard to override the `ExpressClassHandler.DoInsert` method in the page handler class of the Customer page to specify that the most recently-entered record displays immediately after insert.

Before beginning this customization, please review [“Customizing With the Page Handler Customization Wizard” on page 67](#). Alternatively, you can read the online Help topic found under the Help index entry [“Customize:Page Handler.”](#)

➤ **To code this customization with the Customization Wizard**

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Start the Page Handler Customization Wizard for the Customer page.
3. Open the Processing Commands category.
4. Double-click on the “Insert: Inserting a new object in the result set” customization point.

If this is the first customization you are making to this class, WebEnterprise Designer displays a confirmation dialog to expand the class hierarchy to include a customizable class for this class page. Click OK.

The Method Workshop opens, displaying an override of the DoInsert method.

5. Add the following code to the end of the method body:

```
pagedata.FirstVisibleRow = pagedata.data.items;
```

The whole method body should look like this (changes in **bold**):

```
super.DoInsert(request=request, response=response,
               parameters=parameters, rset=rset, pageData=pageData);
pagedata.FirstVisibleRow = pagedata.data.items;
```

6. Compile the method and close the Method Workshop.
7. Close the Page Handler Customization Wizard.
8. Regenerate code for the application.

Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

➤ **To test the DoInsert method customization**

1. Run the HTMLtutApp application.
2. Navigate to the Tip-a-Canoe Customers page.

3. Click the `Insert` command in the command list (left pane).

The Insert Customer page appears.

4. Enter valid data in all the fields, for example:

Customer Number: 4
Name: Alfred E. Newman
Address: 123 Main St.
Phone: 456-1234

5. Click Insert.

The Customer display page should appear, showing this record, as shown in [Figure 7-14 on page 204](#).

Example: Validating a Whole Form

By default, WebEnterprise Designer allows a user to leave fields blank when submitting an insert, update, or search. This section and the one that follows present customizations that prevent a user from leaving a particular field blank when submitting an insert. The example in this section uses a JavaScript script and other modifications to the HTML template.

NOTE This example validates multiple fields when the form is submitted (when the user presses the appropriate command button). To validate an individual field at data entry time (when the user tabs out of the field after entering data), see [“Example: Making a Field Mandatory” on page 213](#).

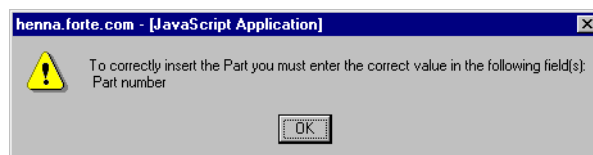
What This Example Does

This customization example modifies the HTMLtutApp tutorial application to force a user to enter data in the PN (part number) and Description fields when inserting a new record in the Tip-a-Canoe Products page.

Figure 7-15 Restricted Fields on Insert Page

The screenshot shows a form titled "Insert Part2" with a yellow background. It contains four input fields, each with an orange label box to its left: "PN", "Description", "Price", and "Picture". A callout line points from the text "Restricted fields (cannot be null)" to the four input fields. At the bottom left of the form is a grey button labeled "INSERT".

For example, if a user leaves one of these field blank, when she clicks the Insert button, an error message pops up, specifying the constraint:

Figure 7-16 Restricted Field Error Message

Creating a Field Constraint with JavaScript

This section describes how to add a constraint to a field by using modifications to an HTML template. The modifications include adding JavaScript code and modifying the existing HTML code. The *general* steps of this procedure are as follows (all modifications are to the Insert mode template of the Part2 page):

1. Add the JavaScript script to the HTML page.
2. Add a value attribute to the field descriptions.
3. Modify the Insert button

JavaScript Boilerplate

The JavaScript code used for this example is based on the following JavaScript boilerplate script. You can use this boilerplate as a starting point for your own validation script.

```

[Start script]
<SCRIPT LANGUAGE="JavaScript">

[Start Field Validation validation]
function fieldvalidation(form) {

[Contains allblanks function]
allblanks(form)
//Add functions as needed. Each function corresponds to a field
function1(form)
}

[Boilerplate for specifying a function affected by the validation.]
function allblanks(form) {
if((function1(form)) {
form.submit()
}

[Begin condition if field fails]
if((function1(form) == false){
compose(form)
}
}
function compose(form) {

var text = "You forgot to fill in correctly the following
field(s):"

[Begin condition if field fails]
if(function1(form) == false) {
text += "\n your Message"
}
alert(text)

[Creates error window]
}

[Text of error message]
function function1(form) {
if (form.FieldName.value == "") {

[Rest of text for function1]
return false
}
else {
return true
}

[Defines the function1 fn that checks that a specific field is not blank.]

```



```
    }  
  }  
</SCRIPT>
```

Step 1. Add the JavaScript Validation to the Template

In this section, you add a JavaScript validation script to the `Insert_Part2.html` template.

Before beginning this section, please review [“Methods for Editing Generated Files” on page 165](#).

► **To add a JavaScript validation script to restrict non-null field entry**

1. Using the HTML-editing option of your choice, open the `Insert_Part2.html` file.
2. At the top of the file, immediately after the `<head>` tag, enter the following JavaScript code (modifications to the boilerplate script are in **bold**):

```
<SCRIPT LANGUAGE="JavaScript">

function fieldvalidation(form) {
  allblanks(form)
  ispartnumber(form)
  isdescription(form)
}
function allblanks(form) {
  if((ispartnumber(form) && isdescription(form))) {
    form.submit()
  }
  if((ispartnumber(form) == false || isdescription(form) == false)) {
    compose(form)
  }
}
function compose(form) {
  var text = "To correctly insert the Part you must enter the correct
  value in the following field(s):"
  if(ispartnumber(form) == false) {
    text += "\n Part number"
  }
  if(isdescription(form) == false) {
    text += "\n Description"
  }
  alert(text)
}
function ispartnumber(form) {
  if (form.Part2_qq_PartNumber.value == "") {
    return false
  }
  else {
    return true
  }
}
function isdescription(form) {
  if (form.Part2_qq_Description.value == "") {
    return false
  }
  else {
    return true
  }
}
</SCRIPT>
```

3. Save your work, but do not close the file.

This script will be invoked every time the user submits an insert form. If all fields pass the validation, the form is submitted; otherwise, the error message is displayed.

Step 2. Add a Value Attribute to the Field Descriptions

In this section, you add a value attribute to each field that will be validated and initialize it to “. We will later access this attribute in our JavaScript script. In this example, you add the attribute to the PN and Description fields.

► To add a value attribute to the validated fields

1. Further along in the `Insert_Part2.html` template, find the section describing the fields.
2. Add the value attribute as follows (modification is in **bold**):

```
[Begin table definition of PN field]
<tr>
  <th class="labelmodify" style="width:20%">
    PN
  </th>
  <td class="dataform" style="width:80%">
    <input type="text" name="Part2_qq_PartNumber value="" size=40>
  </td>
</tr>
[End definition]
```

3. Make the same change for the Description field.
4. Save your work, but do not close the file.

Step 3. Modify the Insert Button

In this section, you make changes to the Insert button so that it calls the JavaScript script before the form gets submitted.

➤ **To modify the Insert button**

1. Towards the end of the `Insert_Part2.html` template, find the line section specifying values for the Insert button.
2. Add code that calls the validation script.

Your code should look like this (changes in **bold**):

```
[Begin table definition of PN field]
<tr>
  <td class="buttons">
    <INPUT TYPE="button" VALUE="INSERT"
      <b>onClick="fieldvalidation(this.form)"</b>
    </td>
[End definition]
</tr>
```

3. Save your work and close the `Insert_Part2.html` template file.

Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

➤ **To test the customization of the JavaScript validation**

1. Run the HTMLtutApp application.
2. Navigate to the Tip-a-Canoe product list page.
3. Click the `Insert` command in the command list (left pane) to display the Insert page.

4. Enter data in every field except the PN or Description field.

For example:

Insert Part2

PN	<input type="text"/>
Description	<input type="text" value="Sea Kayak"/>
Price	<input type="text" value="\$399"/>
Picture	<input type="text" value="/forte/examples/wedtut/images/kayak.gif"/>

5. Click the Insert button.

The error message window should pop up, as shown in [Figure 7-16 on page 207](#).

Example: Making a Field Mandatory

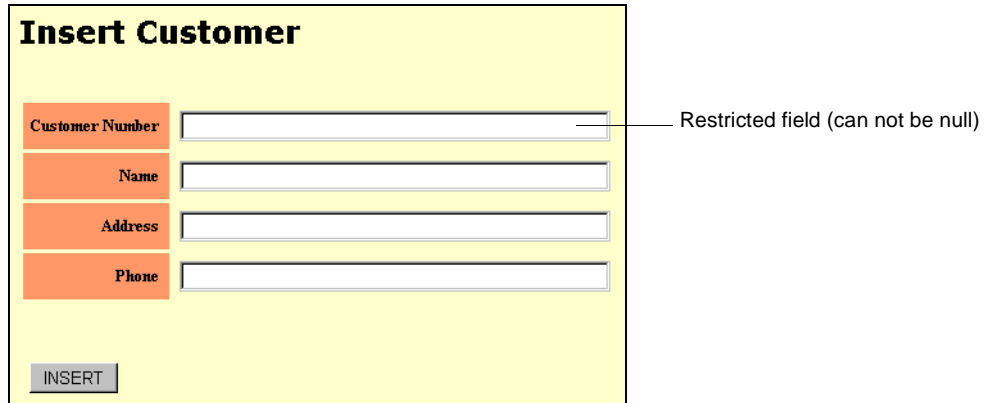
By default, WebEnterprise Designer allows a user to leave fields blank when submitting an insert, update, or search. This section describes a customization that prevents a user from leaving a particular field blank when submitting an insert. The implementation described in this example uses a TOOL customization.

NOTE You could also write a JavaScript function to accomplish the same thing. You would install it in the validation scripts directory (`{FORTE_ROOT}/userapp/http/clx/scripts/validate`), and then select it as a JavaScript validation option in the Validation Options page of the Page Handler.

What This Example Does

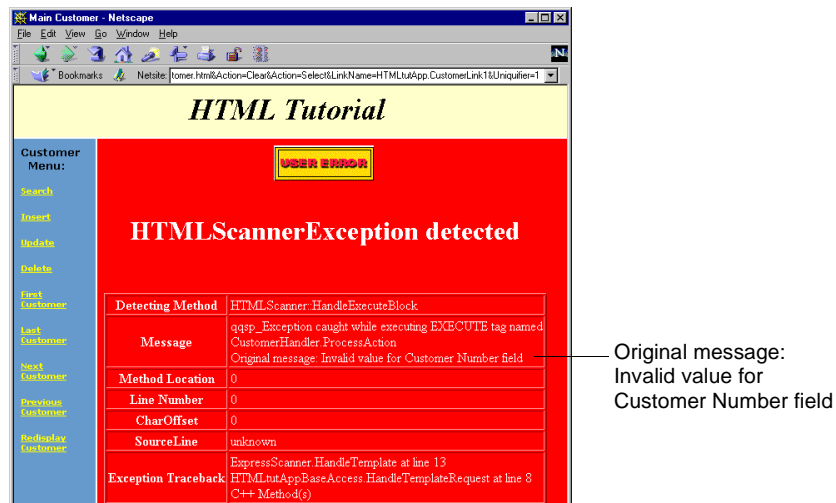
This customization examples modifies the HTMLtutApp tutorial application to ensure that a user enters data in the Customer Number field when inserting a new record in the Tip-a-Canoe Customers page.

Figure 7-17 Restricted Field on Insert Page



For example, if a user inserting a record leaves the Customer Number field blank, when she clicks the Insert button, an error page is displayed, specifying the constraint:

Figure 7-18 Restricted Field Error Page



Creating a Field Constraint with TOOL

This section describes how to add a constraint to a field by customizing TOOL code. In this particular example, you will customize code to prevent the user from inserting blank values into the Customer Number field of the Customer page.

You will use a customization point in the Page Handler Customization Wizard to create a BeforeInsert method for the CustomerHandler class (which will override the ExpressClassHandler.BeforeInsert method).

► To create a field constraint with TOOL

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Start the Page Handler Customization Wizard for the Customer page.
3. Open the Individual Field Operations category.
4. Double-click on the “Decoding or validating a field” customization point.

If this is the first customization you are making to this class, WebEnterprise Designer displays a confirmation dialog to expand the class hierarchy to include a customizable class for this class page. Click OK.

The Method Workshop opens, displaying an override of the DecodeValue method.

5. Add the following code to the beginning of the method body:

```
if ((GetFieldName ( assocID=assocID, fieldIndex=fieldIndex) =
'Customer_qq_CustomerNumber') and (value=NIL or value.value='')) then
  e : GenericException = new;
  e.SetWithParams (SP_ER_ERROR,

[Error message text.]
  'Invalid value for Customer Number field');
  task.ErrorMgr.AddError(e);
  raise e;
end if;
```

The GetFieldName method returns the name of the Customer Number field of the Customer page.

6. Compile the method, close the Method Workshop and the Customization Wizard.

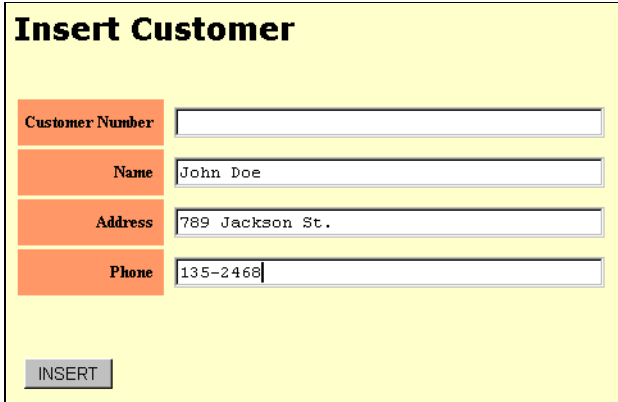
Testing Your Work

Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test the TOOL customization**

1. Run the HTMLtutApp application and navigate to the Customer page.
2. Click the `Insert` command in the command list (left pane).
3. Enter data in every field except the Customer Number field.

For example:



Insert Customer	
Customer Number	<input type="text"/>
Name	<input type="text" value="John Doe"/>
Address	<input type="text" value="789 Jackson St."/>
Phone	<input type="text" value="135-2468"/>
<input type="button" value="INSERT"/>	

4. Click the Insert button.

The error message page should display, as shown in [Figure 7-18 on page 214](#).

Drop List or Radio List Example: Entering Lookup Information Manually

Drop lists and radio lists require a stored value and a display value for each option in the list. When you specify this type of validation on a page field (in the Page Wizard), you can use the attributes of a business class (a “Lookup class”) to supply these values. These values are created at runtime. When you generate code for the application, WebEnterprise Designer generates a file that is populated with the stored and displayed values.

If you leave the Lookup class, display field, and stored field unspecified, WebEnterprise generates a Lookup file without values, and you must populate the file with data manually. This example describes how to do this.

What This Example Does

This customization modifies the HTMLtutApp tutorial application by adding a drop list to the Picture field of the Product List (Part2) page, so that the user can select a title from a drop list rather than having to type in a long file specification string.

Figure 7-19 Customized Drop List

The screenshot shows a form titled "Insert Part2" with several input fields and a drop list. The fields are:

- PN:** 90005
- Description:** Basswood Oars
- Price:** 29.99
- Picture:** A drop list with the following options:
 - Pair of Paddles (selected)
 - <Not Selected>
 - Medium Canoe
 - Large Canoe
 - Small Paddle
 - Large Paddle
 - Pair of Paddles

There is an "INSERT" button at the bottom left of the form.

NOTE The next customization, “[Drop List and Radio List Example: Removing <Not Selected> and <None>](#)” on page 223, shows you how to remove the <Not Selected> option from the drop list.

Creating the Customization

This section shows you how to customize a drop list that is not linked to the attributes of a business class. The *general* steps of this procedure are as follows

1. Add the drop list validation to the gifAddress field of the Part2 business class page.
2. Generate code so that WebEnterprise Designer generates a lookup file.
3. Modify the lookup list in the lookup file, adding displayed and stored values for each option.

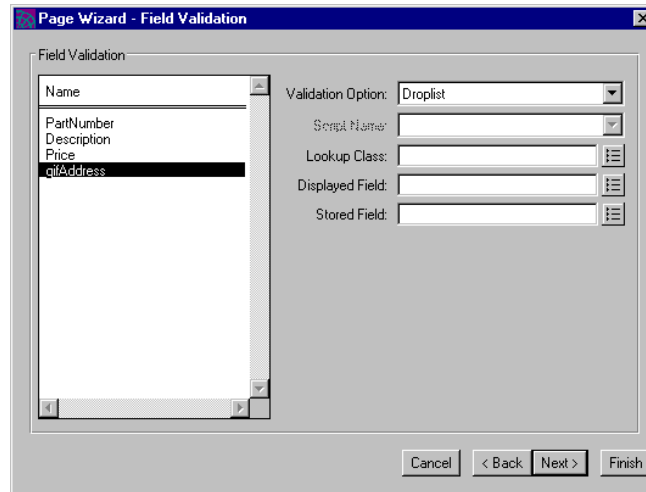
Step 1. Add a Drop List Validation to the Field

In this section, you set up for the customization by adding a drop list validation on the gifAddress (Picture) field of the Part2 page.

► **To add a drop list validation to the gifAddress field**

1. Open the HTMLtutApp application in the HTML Application Model Workshop.
2. Double-click on the Part2 page to display the Page Wizard.
3. Click the Next button repeatedly until you reach the Field Validation page.
4. Select the gifAddress field in the field name pane.
5. From the Validation Option drop list, select Droplist.

- Leave Lookup Class, Displayed Field, and Stored Field unspecified. The Page Wizard Field Validation page should look like this:



- Click Finish to apply your changes and close the Page Wizard.

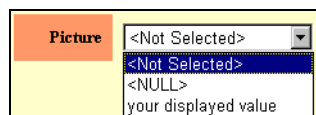
Step 2. Generate the Lookup File

Now, you must generate the lookup file.

► To generate a lookup file

- Regenerate the application's code.
- Test the validation by running the application.
- Navigate to the Tip-a-Canoe Products page.
- Click on the Insert, or Update command in the command list (in the tutorial, you removed this field from the Search page; otherwise, you could choose Search, too).

The appropriate page mode appears. The Picture field on the Insert or Update pages should contain three values:



These are the default values defined in the lookup file when there is no lookup class defined for the validation.

5. Exit the application.

Step 3. Customize the Lookup File with Your Values

In this section, you will modify the generated lookup file and add the values for your list. You can use the Page Handler Customization Wizard to access the file. For information, see [“Methods for Editing Generated Files” on page 165](#).

There are five graphic files provided for the HTMLtutApp tutorial. You will use the fully-defined file specification for the files as the stored values. The displayed values will be labels for the five pictures.

► To customize the lookup file

1. Using the Page Handler Customization Wizard, open the `Part2_qq_gifAddress.inc` file.

The file is found in the Lookup Data Files category of the Wizard. This category only exists when the application contains a drop list or radio list validation. Open the file by selecting it and clicking the Customize button.

The original code looks like this:

```
[Declare total number of values in the list.]
NumValues = 2;

[Initialize stored value array]
storedValues = new Array(NumValues);

    displayedValues = new Array(NumValues);
[Initialize displayed value array]

[Null value stored value]
storedValues[0] = 'qqNULLqqZa047!#$%^_yJk523qqNULLqq';

[Null value displayed value]
displayedValues[0] = '&lt;NULL&gt;';

[First non-null value stored value]
storedValues[1] = 'your stored value';

[First non-null value displayed value]
displayedValues[1] = 'your displayed value';
```

2. Modify the NumValues value to the number of values you will have.

For this example, specify five values:

```
NumValues = 5;
```

3. Specify whether the lookup list will have a <NULL> value or not.

For this example, we do not want a <NULL> value, so comment out the null value line(s):

```
// storedValues[0] = 'qqNULLqqZa047!#$%^_yJk523qqNULLqq';
// displayedValues[0] = '&lt;NULL&gt;';
```

4. Replicate the storedValues, and displayedValues lines, so that the total number of paired lines matches the NumValues number (in this case, five pairs total).
5. Modify the elements' indices so that they start at 0 and end in NumValues - 1.
6. Modify the values assigned to storedValues and displayedValues, entering appropriate values to the array.

In this example, the storedValues strings are the file specifications for the five graphic product files. All files are stored in the `/forte/examples/wedtut/images` directory.

NOTE Graphics files for WebEnterprise Designer applications are assumed to be stored under the Web server root directory, so their specifications are relative to this directory.

The displayedValues strings will be labels for the files, as follows:

File name	displayedValues string
canoea.gif	Medium Canoe
canoef.gif	Large Canoe
paddle1.gif	Small Paddle
paddle2.gif	Large Paddle
2paddles.gif	Pair of Paddles

Your modified code should look like this (changes are in **bold**):

```
storedValues[0] = '/forte/examples/wedtut/images/canoea.gif';  
displayedValues[0] = 'Medium Canoe';  
storedValues[1] = '/forte/examples/wedtut/images/canoef.gif';  
displayedValues[1] = 'Large Canoe';  
storedValues[2] = '/forte/examples/wedtut/images/paddle1.gif';  
displayedValues[2] = 'Small Paddle';  
storedValues[3] = '/forte/examples/wedtut/images/paddle2.gif';  
displayedValues[3] = 'Large Paddle';  
storedValues[4] = '/forte/examples/wedtut/images/2paddles.gif';  
displayedValues[4] = 'Pair of Paddles';
```

7. Close the `Part2_qq_gifAddress.inc` file and save your changes.

NOTE Save a copy of the `Part2_qq_gifAddress.inc` file in another directory. If you make changes to the model and regenerate code, this file is replaced with the uncustomized version. After regeneration, replace the new file with your saved version.

Testing Your Work

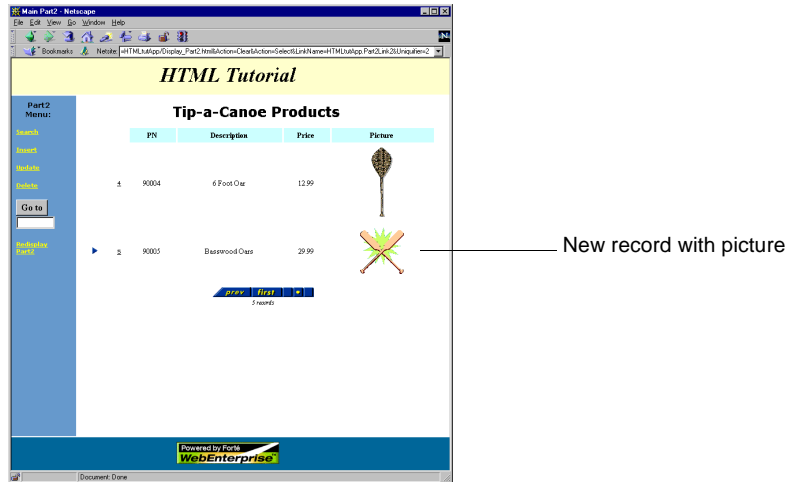
Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test your manually customized lookup file**

1. Start the HTMLtutApp tutorial application.
2. On the home page, click the Product List link to access the Products page.
3. Click Insert to display the Insert mode of the Products page.
4. Enter the following values on the Insert page:
PN: 90005
Description: Basswood Oars
Price 29.99
5. Select the Pair of Paddles option from the Picture drop list.
6. Click the Insert button to redisplay the Tip-a-Canoe Products page.

7. Navigate to the last record.

Your page should look like this:



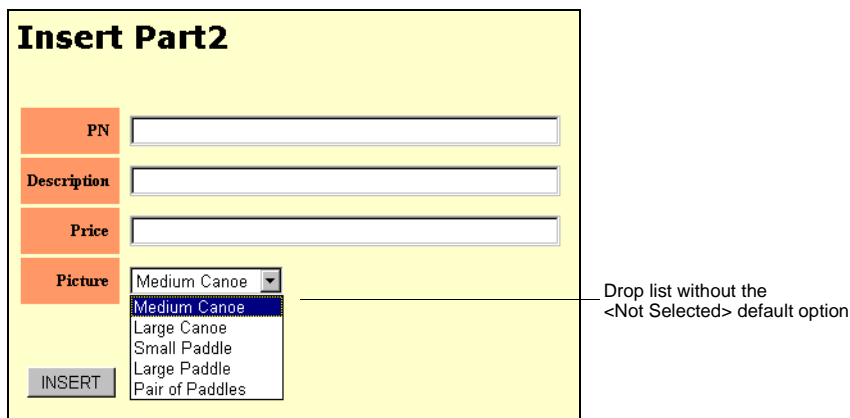
Drop List and Radio List Example: Removing <Not Selected> and <None>

By default, the WebEnterprise Designer drop list and radio list include <Not Selected> and <None> as options. In fact, <Not Selected> is the default option for the drop list, and <None> is the default for the radio list. This example describes how to remove these options, in case you do not want them in your drop list or radio list.

What This Example Does

This example builds on the customization described in the previous section (“[Drop List or Radio List Example: Entering Lookup Information Manually](#)” on page 217), removing the <Not Selected> default from the drop list on the Insert and Update modes of the Product List page. The resulting drop list looks like this:

Figure 7-20 Customized Drop List



Creating the Customization

You can remove the default options in one of two ways:

- modify the default option specification in the generated page mode files
- modify the default option parameter in the `Scripts_bus_class_page.html` file

You use the same procedures to remove the <None> default from a radio list. The only difference is whether in the HTML model you specify a radio list or a drop list field validation.

This section describes both techniques. Although the drop list validation used for this example was customized manually (in [“Drop List or Radio List Example: Entering Lookup Information Manually”](#) on page 217), the techniques described here work the same for a drop list or radio list that uses values from a Lookup business class.

Technique 1: Customizing the Page Mode Template

In this section, you customize the Insert and Update templates of the Product List page, changing the default selection string into either a nonsensical string or a legal string (that is, one of the stored values in the list). If you use a nonsensical value, the default option is not displayed; if you use a legal value, that one becomes the default value.

Before beginning this section, please review [“Methods for Editing Generated Files”](#) on page 165.

► **To remove the <Not Selected> option from a drop list**

1. Using the HTML-editing option of your choice, open the `Insert_Part2.html` file.
2. Locate the table definition for the Picture field.

The original code looks like this:

```
[Begin table definition of Picture field]
<tr>
  <th class="labelmodify" style="width:20%">
    Picture
  </th>
  <td class="dataform" style="width:80%">

[Call to JavaScript validation]
  <script language="JavaScript">
    Select_Part2_qq_gifAddress('Part2_qq_gifAddress', 'Picture', 'qqNone');

[End call]
  </script>
</td>

[End definition]
</tr>
```

The “qqNone” is passed to the script as a parameter for the default value to be selected in the drop list or radio list.

3. You could either:

- remove the `Select_Part2_qq_gifAddress('Part2_qq_gifAddress','Picture','goaway'` parameter completely
- add any string value, for example:

```
Select_Part2_qq_gifAddress('Part2_qq_gifAddress','Picture',  
, 'goaway');
```

- add a valid selection that you want to highlight, for example, replace “qqNone” with one of the stored values used in [“Step 3. Customize the Lookup File with Your Values” on page 220](#):

```
Select_Part2_qq_gifAddress('Part2_qq_gifAddress','Picture',  
, '/forte/examples/wedtut/images/paddle2.gif');
```

4. Close and save the `Insert_Part2.html` file.
5. Repeat [Step 1](#) through [Step 4](#) for the `Update_Part2.html` file.
6. Close the Page Handler Customization Wizard.
7. Go to [“Testing Your Work” on page 228](#) for a description of what your results should look like.

Technique 2: Customizing the Scripts File

This section describes how to implement the same customization of the previous section, but modifying the `Scripts_Part2.html` file instead of the page mode templates.

► **To remove the <Not Selected> option from a drop list**

1. Using the HTML-editing option of your choice, open the `Scripts_Part2.html` file.
2. Locate the line containing the “qqNone” string.

The original code looks like this:

```
if (Selected_Value == 'qqNone')
document.writeln('<option selected value="">&lt;Not Selected&gt;');
```

3. Comment this section out:

```
//if (Selected_Value == 'qqNone')
//document.writeln('<option selected value="">&lt;Not Selected&gt;');
```

4. Close and save the `Scripts_Part2.html` file.
5. Close the Page Handler Customization Wizard.

Testing Your Work

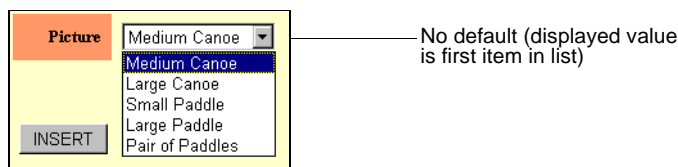
Now it is time to test your work to verify that you made no mistakes and the customization works correctly.

► **To test your drop list customization**

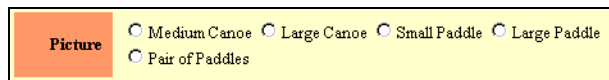
1. Run the customized HTMLtutApp application.
2. From the Home page, click on the Product List link to access the Tip-a-Canoe Product List page. On the Product page, click on either the `Insert` or `Update` command in the command list.

If you substituted a nonsensical string in your customization, the results should look like this:

Drop list: No <Not Selected> option, and no default



Radio list: No <None> option, and no default

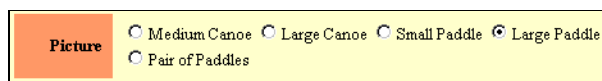


If you substituted a valid stored value string in your customization, the results should look like this:

Drop list: No <Not Selected> option, and the default is the displayed value for the `paddle2.gif` file



Radio list: No <None> option, and the default is the displayed value for the `paddle2.gif` file



Customizing Application Security

WebEnterprise Designer provides a number of application security models, ranging from open access to all pages to restricted page-based access to authenticated users.

This chapter provides information on all types of security customization, including:

- WebEnterprise session management
- how to implement a logon procedure to restrict access to authenticated users
- how to grant application page access based on the authenticated user
- how to share WebEnterprise session management across several HTML applications
- other common WebEnterprise security customizations, such as session timeout

Security and HTML Applications

WebEnterprise Designer's implementation of security is based on the session management feature of WebEnterprise. For complete information on this feature, see *A Guide to WebEnterprise*.

Normally, the HTTP requests issued by Web browsers are stateless, meaning each request is atomic and does not require any knowledge of past (or future) requests. Many simple read-only Web applications are built in this manner,

More sophisticated applications, such as those created with WebEnterprise Designer, require knowledge of the browser client's state, including such information as intermediate data in a multiple-request operation, the username and access rights of the browser client, and so forth.

Storing state information In standard iPlanet UDS window-based applications, you can store state information on both the client (using local objects) and the server (using transaction or session dialog duration service objects). However, in a Web application, you cannot use either of these mechanisms. Instead, you use WebEnterprise session management to enable the application to associate all of the browser's application requests into a *session*, which contains this information. See *A Guide to WebEnterprise* for more details on session management.

Embedding session ids in cookies vs. URLs Sessions are tracked in one of two ways: using Web "cookies" containing a unique session id, or embedding the session id directly in all the application URLs. WebEnterprise will attempt to use cookies but, if the client's browser has disabled cookies, will instead modify all URLs with session ids. This "cookie detection" happens automatically when a client first requests a page from an HTML application; no special action is required.

Managing shopping cart applications One type of sophisticated application, generally known as a "shopping cart" application, permits users to enter data into the system. Such applications do not restrict access to the individual HTML pages, but must track a sequence of HTML requests together as a transaction. When the transaction is completed (for example, by submitting a purchase form), the request includes identifying data (such as name, address, and credit card information) to ensure that the submission is valid.

The session management used in default WebEnterprise Designer applications supports the creation of these shopping cart-style applications. All application pages are accessible to all clients. No user logon process is required. Information required to place an order (for example, name, address, credit card number, and so forth) can be entered on an order form and submitted.

Managing secure Web applications Another type of sophisticated application permits access to specified HTML pages only to validated users. Typically, these applications present a logon page, which is presented to the user before he can enter the application. The user supplies an "identity" (usually a username and password), which, when validated, grants access to the application's HTML pages. The identity becomes a property of each Web request and can be used by the application to restrict access to certain pages or activities.

This chapter describes how to customize an HTML application to build this third type of "secure" application, including:

- creating a logon page
- integrating an application with your own user authentication system

- granting different levels of application access to different users
- sharing a WebEnterprise security environment across several HTML applications

Authenticating Users

When a WebEnterprise Designer application does not have a logon page defined for it, all generated HTML templates are defined with the `SESSION_AUTOCREATE` session property. A WebEnterprise session is then automatically created whenever a Web client requests an HTML template. This “Web session” is persistent, binding subsequent Web requests from that browser client together, until it eventually expires (based on the value of the `SessionTimeout` attribute). In this scenario, all application pages are accessible without any logon or authentication procedure.

For information on the `SessionTimeout` attribute, see online help or *A Guide to WebEnterprise*. You can override the default (for WebEnterprise Designer, 60 minutes) in WebEnterprise Designer applications with the application-wide “Modify session timeout” customization point of the Page Handler Customization Wizard.

Creating the Logon Page

You create your application’s logon page with the HTML Application Model Workshop’s Page Wizard.

► To create a Logon page



1. In the HTML Application Model Workshop for your application, click the New HTML Page button.

The Page Wizard appears.

2. Select Logon Page in the page type list and click Next.

The Logon Page Options page is displayed.

3. Modify the fields on the Logon Page Options page as desired.

For information on all the fields on this dialog, press the Help key.

Notice that the default logon page contains two user-supplied fields. If your site's security framework requires additional (or fewer) logon fields, you will need to customize the generated Logon.html file accordingly. See [Chapter 3, "Customizing Generated HTML Templates,"](#) for details.

4. Click Finish to create the logon page.

Code Generated for the Logon Page

When you generate code for an application that contains a logon page, WebEnterprise Designer generates a LogonHandler TOOL class in the *html_modelHandlers* project, and three HTML templates for the logon page:

- `Logon_logon_page.html`

This template contains the HTML form that is presented to the un-authenticated user when they attempt to access any page in the application. You can customize your logon page by editing this file.

- `Validate_logon_page.html`

This template is requested when the user submits the Logon form. It contains the `WebEnterprise` tag that causes the user authentication data to be validated.

This page is never actually displayed and need not be edited.

- `LogonFailed_logon_page.html`

If the user's supplied authentication data fails validation, then this HTML page is returned to the user. You can supply an application or company-specific error page by editing this file.

If the user is successfully validated, the application's start page is returned to his or her browser. The user then has access to all of the HTML pages in the application. (See below for how to add further granularity to your application's security model.)

How the Logon Page is Activated

For background on this subject, review [“ExpressHTTPAccess” on page 44](#) and [“ExpressLogonHandler” on page 54](#).

Creating a logon page has the following effect on the HTML application: Every HTML page in the application (except the logon page) is marked with the `WebEnterprise` `SESSION_REQUIRED` property. This means that the page will not be returned unless the client already has a valid `WebEnterprise` session. If a request is received for one such page and there is no valid session (if the request has neither a valid session cookie or session id in the URL), then `WebEnterprise` redirects the request to the `Logon_logon_page.html` page, which is returned to the browser. The Web address of the logon page becomes the `WebEnterprise` `SessionCreationURL` of the application.

For additional information about the `SessionCreationURL` attribute, see online help.

► To activate the logon page

1. The browser user requests the HTML application's start page (`Start.html`).
2. If logon security is implemented in the application, the logon page is returned.
3. The user enters the appropriate user identification information and submits the form (clicks the Logon button).

4. The application validates the information and, if authenticated, creates a WebEnterprise session.
5. The application's starting page is returned to the browser.

If the user information fails validation, no session is created and the LogonFailed page is returned.

Integrating the Application with an Authentication System

When the user fills out and submits the logon page, the `ExpressLogonHandler.HandleCondition` method is invoked. This method calls the application's `html_modelAccess.LogonSession` method to authenticate the data entered on the logon page. If the logon is successful, then:

- WebEnterprise creates a WebEnterprise session and attaches it to the Web response
- if `LogonSession` returns a "credentials" object, then it is attached to the WebEnterprise session using the name "Logon-Credentials."

During processing of subsequent requests, these credentials can be retrieved by means of the `HTTPSession.GetSessionData` method.

By default, the `LogonSession` method returns a value of `TRUE`, indicating that the user is permitted to create a session and access the application's pages. Adding security to your HTML application requires you to customize the `LogonSession` method.

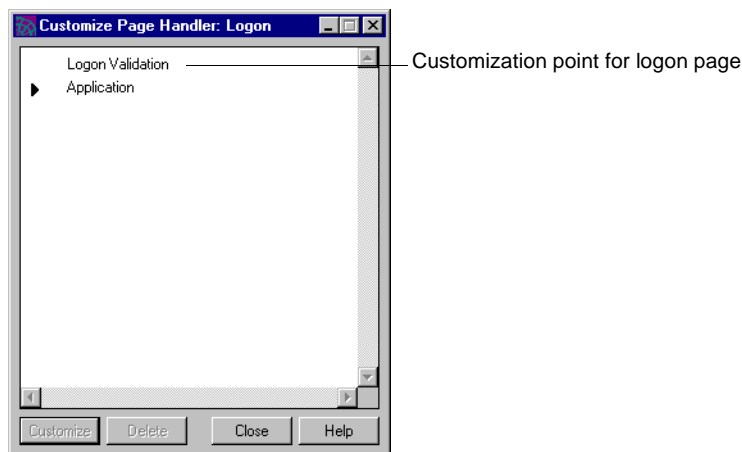
Your `html_modelAccessService.LogonSession` method captures the parameters entered on the logon page and passes them to your own user authentication subsystem (such as an LDAP service, the Windows NT user authentication system, or an X.500 server) for validation. `LogonSession` should return the boolean `TRUE` if the user's logon is validated, and `FALSE` if not. You can optionally return a "credentials" object containing any user-specific data (username, group, and so forth) that your application requires.

Use the Page Handler Customization Wizard to customize the logon page.

► **To customize an application's logon page using the Customization Wizard**

1. Open your application in the HTML Application Model Workshop.
2. Select the logon page.
3. Choose the Component > Customize... command to open the Wizard.

The Page Handler Customization Wizard appears.



4. Select the "Logon Validation" customization point and click Customize.
The Method Workshop opens, displaying the LogonSession method. This method overrides the HTTPAccess.LogonSession method.
5. Add your TOOL code to authenticate the user information and, optionally, place persistent user information into the credentials object.

Example: Adding LogonSession Code

For this example, assume that a SecuritySO service object exists containing an Authenticate method that can validate a username-password pair with an external authentication subsystem. This method also returns a "Rights" object that contains credential information. The Rights object might, for example, denote various levels of permitted access that restricts some page access to certain users.

The LogonSession method might look like this:

```

LogonSession(request: HTTPRequest, output credentials: Object): boolean
//
// LogonSession
// The LogonSession method is used to validate a new session.
// LogonSession should return TRUE if the logon information is
// valid, signalling that the session may be created. FALSE
// should be returned if the logon information is invalid and
// the session creation request should be rejected.
//
// request
// The request parameter holds the HTTPRequest object of
// the HTTP request that is attempting to create a new
// session.
// Default logon forms will have 'Username' and 'Password'
// parameters.
//
// credentials
// The credentials parameter is used to return authentication
// information. It will be attached to the session using the
// descriptor 'Logon-Credentials' if the session is
// successfully created. Subsequent requests may use the
// GetSessionData method of the HTTPSession object using
// the 'Logon-Credentials' descriptor to retrieve the
// credentials object.
//
// return value
// TRUE - logon validated; session may be created.
// FALSE - logon rejected.
//
//
begin
credentials = NIL;

//
// First retrieve the data entered on the Logon page
//
userName : TextData = request.FindNameValue('Username');
passWord : TextData = request.FindNameValue('Password');

// Now call our authentication subsystem to validate the user
// (actual implementation is site-specific)
//
if SecuritySO.Authenticate(userName, passWord, credentials) = FALSE then
return FALSE;
end if;

//
// The credentials object will be stored with the HTTPSession.
// It can be retrieved via
// GetSessionData(IntegerData(value='Logon-Credentials'))
//
return TRUE;
end method;

```

Restricting Access to Application Pages

This section shows what you can do with authentication credentials for a WebEnterprise Designer client after you have captured them.

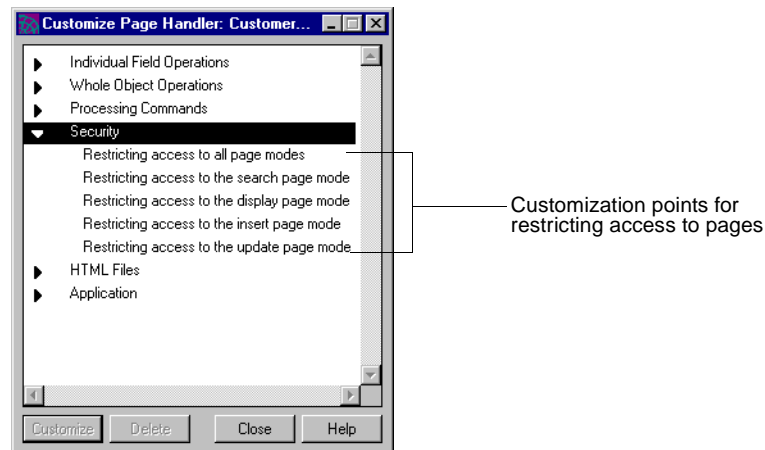
A common application requirement is granting different levels of application access to different users or groups of users. This section explores a technique for solving this problem. Building on the example in the previous section, the example in this section will restrict “update” and “insert” access to users who are managers.

Use the Page Handler Customization Wizard to customize the appropriate page of the application.

► To restrict access to a page

1. Open your application in the HTML Application Model Workshop.
2. Select the page you wish to restrict access to.
3. Choose the Component > Customize... command to open the Wizard.

The Page Handler Customization Wizard appears.



4. Select the “Restricting access to the insert page mode” customization point and click Customize.

The Method Workshop opens, displaying the RestrictInsertAccess method in the page handler class of the selected page. This method overrides the ExpressHandler.RestrictInsertAccess method. The customizable section of this method is near the bottom and looks like:

```

if (not restrict) then
//
// Add custom code here to compute restrictions:
//
// restrict = . . .
end if;

```

Assume that the “credentials” object saved in the above section is a TextData object with values of either “Manager” or “Worker”. To permit access only to managers, replaces the above block with the following code:

```

if (not restrict) then
  credential : TextData =
    TextData(response.CurrentSession.GetSessionData(
      TextData(value='Logon-Credentials')));
  if (credential.Compare(source='manager', ignorecase=TRUE)
<> 0) then
    restrict = TRUE;
  end if;
end if;

```

5. Add the identical customization for the “Restricting access to the update page mode” customization point.

Only managers can now modify the application page.

You can customize each application page in this manner to restrict access to individual page modes (namely, search, display, insert, and update) or to all page modes. Users who request a page that is denied to them receive an appropriate error page.

Sharing a Security Environment Across HTML Applications

WebEnterprise Designer supports designs that span multiple HTML applications. One HTML model can include another, thus providing links between related applications. The WebEnterprise Help system describes how to include one HTML application in another (refer to the topic “New Included HTML Application Model”).

WebEnterprise supports several different security designs for included applications. Consider an HTML Model “MainModel” that includes a second HTML Model, “SubsidiaryModel.” Assume that MainModel implements a logon page that restricts access to the application. For queries within MainModel, full WebEnterprise security will be in effect. But what happens when a user clicks on the link from a MainModel application page to the SubsidiaryModel page? It depends.

Avoiding Security Leaks

The default WebEnterprise Designer security model grants open access to all pages to all users. Therefore, unless you employ one of the mechanisms described below, users jumping to a SubsidiaryModel page are not bound by MainModel’s WebEnterprise security environment. When the user jumps back to the MainModel application, the requested pages are again governed by the existing MainModel session.

Alternative 1: Give both applications a logon page One alternative is to define logon pages in both SubsidiaryModel and MainModel. In this case, each HTML application then implements its own independent session management. When users first accesses a MainModel page, they are required to logon. When they subsequently jump to a SubsidiaryModel page, they are again required to logon, this time starting a SubsidiaryModel session. When they return to a MainModel page, they resume running within their existing MainModel session.

NOTE In this security model, because the two applications each implement their own session management environment, they can use different authentication subsystems and enforce security in completely different ways.

Alternative 2: Let all applications share the session A more sensible security model lets the `SubsidiaryModel` (and any other included applications) share the `WebEnterprise` session initiated by the main application, `MainModel`. `MainModel` defines a logon page, performs authentication, and implements session management for itself and all applications sharing its session management environment.

Sharing session management between a main HTML model and one or more secondary models requires customization of each HTML model, as described next.

Customizing Subsidiary HTML Models to Share Security

Follow these procedures with each subsidiary model.

► To customize subsidiary models to share the security environment

1. In each subsidiary application, create a logon page.

This page will never be returned to the user, so there is no need to customize the generated `LogonSession` method.

2. Customize each subsidiary application's logon page using the Page Handler Customization Wizard.

Use the "Is subsidiary application" customization point under the Application group. This action creates an `html_modelAccess.IsPrimarySession` method, which does not require any change.

Customizing the Main HTML Model to Share Security

Follow these procedures to customize the main HTML application model.

► To customize the main model to share the security environment

1. Implement an authenticated logon page in the `MainModel`, using the procedures described in "[Authenticating Users](#)" on page 231.

2. Customize the main application's logon page using the Page Handler Customization Wizard.

Use the "Define subsidiary applications" customization point under the Application group. This creates an *html_model*.Access.ConnectManagedSessions method.

3. The lower part of the method contains a block of code that you must now modify to add each subsidiary application to the ManagedSessions array.

For example, if the main application will manage two subsidiary applications, Sub1 and Sub2, then add their Access service objects to ManagedSessions, as shown:

```
ManagedSessions = new;
//
// User code should be added here to populate the managedSessions array
// with references to the ExpressHTTPAccess service objects of the
// subsidiary applications. Replace subsidiaryAccessService with
// subsidiary application's AccessService service object. Add additional
// lines as necessary to specify other subsidiary applications.
//
ManagedSessions.AppendRow( Sub1AccessService );
ManagedSessions.AppendRow( Sub2AccessService );

super.ConnectManagedSessions;
```

Restrictions

Ensure that all HTML applications that share a common session manager are also all accessed through the same Web server. The Fortecgi Location and, if specified, Plug-in URL properties for all linked HTML applications must reference the same Web server URL.

The ConnectManagedSessions method overridden in "[Customizing the Main HTML Model to Share Security](#)," above, must execute after each of the subsidiary Access services has completed initialization. The ConnectManagedSession template contains a method call to delay for 20 seconds before referencing the subsidiary Access services:

```
//
// This is so that the managed applications' ExpressHTTPAccess service
// have a chance to complete their startup.
// On some systems you may need to increase the delay to allow sufficient
```

```
// time for the managed applications' startup to complete.  
//  
task.delay(20000); // 20-second delay
```

In some networked configurations, this delay might be insufficient. If the referenced objects are not completely instantiated, then a distributed access exception will be raised. In that case, you should increase the length of the delay appropriately.

Summary

The above implementation will cause a single session manager to be created, shared by the three HTML applications. A single logon page is presented to the user, governing access to all application pages.

Session Timeout

In this section, you will learn how “session timeout” works in a WebEnterprise Designer application, and how you can customize this feature.

How Session Timeout Works

WebEnterprise creates a session when a user at a browser first accesses a WebEnterprise Designer application. Objects required to track the session, including current database result sets, are then associated with the session and reside within iPlanet UDS services.

Web applications generally do not have a “Logout” procedure; the user simply moves on to a different Web site, or closes their browser, or goes home. In each case, the Web application receives no notification that the user is done with the application. Web applications, then, must decide for themselves when a user’s session can be deleted.

WebEnterprise defines a `SessionTimeout` for each application (or, set of applications when the security environment is shared). Each client request updates a timestamp in its `HTTPSession` object. WebEnterprise periodically scans all managed sessions, identifies those that have been inactive for longer than the `SessionTimeout` value, and deletes the session and any data associated with it.

For information on the `SessionTimeout` attribute, see online help for either `HTTPSession.SessionTimeout` or `SessionMgr.SessionTimeout`. For additional information, see *A Guide to WebEnterprise*.

Consider the case of a user sitting at a browser window using a WebEnterprise Designer application. She has successfully gone through the Logon process and is navigating through the application. She has entered a Search page and selected a data set, the first 10 rows of which are displayed on her screen. She goes to lunch, returning an hour later. She sits at her workstation and clicks the application's Next button. What happens?

Resuming a timed-out application While she was enjoying her Curried Prawns, WebEnterprise realized that her session “timed out” and deleted her `HTTPSession` object. It now receives a request that contains the previously-valid session id. But, not finding it in its cache of sessions, WebEnterprise does what it always does when a request without a valid session is received: it returns the logon page. Our user, perceptively realizing that she must start over, re-enters her username and password, is authenticated, enters search criteria, and resumes her work.

Finding the Ideal Setting for Session Timeout

The “ideal” setting for `SessionTimeout` varies from application to application. If it is set too low, users may find their session invalid when they answer a short phone call. Requiring the user to logon again after a short distraction will not endear the user to the application.

On the other hand, each active session does entail a cost, namely, the memory required to store the `HTTPSession` object and any data associated with the session. For WebEnterprise Designer applications, the data sets associated with business queries may be substantial. The WebEnterprise partition may find itself running out of memory if it must manage a large number of active sessions, each with a large amount of session data.

You can adjust the memory profile of WebEnterprise Designer applications in a variety of ways:

- boost the partition's available iPlanet UDS memory by setting the `FORTE_GC_SPECIAL` environment variable
- reduce the amount of session data by restricting the number of rows fetched into the intermediate result set

By default, WebEnterprise Designer fetches and caches 100 rows of data.

- when designing application pages, select very long fields in pages that are used infrequently
- customize the application to reduce session timeout
See the next section for information on this suggestion.

Example: Customizing Session Timeout

The default session timeout for WebEnterprise Designer applications is one hour. In practice, however, due to the frequency with which session expiration is checked, an inactive session could remain in memory for slightly less than two hours before it is deleted (if the user issued a request shortly after a check, it would not be a candidate for the next check, since, by that time, 60 minutes had not yet passed; therefore, it would not be deleted for yet another 60 minutes).

► To change the session timeout

1. Start the Page Handler Customization Wizard for the logon page, as described in [“Integrating the Application with an Authentication System” on page 234](#).
2. Open the Application category and select the “Modify session timeout” customization point and click Customize.

The Method Workshop opens, displaying a customizable `SetSessionManagement` method. This method overrides the `ExpressHTTPAccess.SetSessionManagement` method.

3. Locate the line of code that defines the `SessionTimeout` value:

```
timeOut.SetValue('00:00:00:01:00:00');
```

`timeOut` is an `IntervalData` object. Change the value string to a different, valid `IntervalData` value. When the application is next executed, all sessions will observe the new session timeout interval.

Other Security Customizations

A WebEnterprise Designer application’s security framework is primarily defined within a single method: `ExpressHTTPAccess.SetupSessionManagement`. If your application requires security customizations not described in this chapter, then it may be necessary to override and modify this method. For details on WebEnterprise security and session management, see *A Guide to WebEnterprise*.

Partitioning and Deploying a WebEnterprise Designer Application

This chapter provides information about the elements of a distributed WebEnterprise Designer application and how to partition and deploy a WebEnterprise Designer application.

Topics in this chapter include:

- WebEnterprise Designer application projects
- WebEnterprise Designer application service objects
- how to make a default partitioning configuration
- how to test the application in a distributed environment
- how to deploy a WebEnterprise Designer application
- how to start a WebEnterprise Designer application

For complete information on iPlanet UDS partitioning and deployment, see *A Guide to the iPlanet UDS Workshops* and the *iPlanet UDS System Management Guide*.

About Partitioning a WebEnterprise Designer Application

After generating your code for your HTML application and making any desired enhancements, use the Partition Workshop to:

- test the application in a distributed environment
- partition the final application for deployment and make the appropriate application distributions

The following sections provide background information about the projects and the service objects in the HTML applications, and the default partitioning scheme iPlanet UDS provides for WebEnterprise Designer applications.

About HTML Application Projects and Service Objects

As described in *Getting Started with WebEnterprise Designer*, when you generate the code for your HTML application model, two or more iPlanet UDS projects are created:

Project Name	Description
<i>html_model</i> Handlers	The main project for the application.
<i>business_model</i> Services	A supplier project for the “Handlers” project. There is one “Services” project for each business model that is a supplier to the HTML application model.

Partitioning the application The “Handlers” project generated from the HTML application model is the *main project* for the application. The “Services” project generated from the business model functions as a *supplier project* to the main project. Therefore, when it is time to test your application in a distributed environment or partition the application for deployment, you must partition the “Handlers” project.

WebEnterprise Designer applications frequently consist of multiple business models and multiple HTML models. A Services project is generated for each business model, and a Handlers project is generated for each HTML model. Whenever an HTML model references a business model, the generated Handlers project will include the generated Services project as a supplier. If an HTML model includes another HTML model, then the generated Handlers project for the first model will include the generated Handlers project for the second model as a supplier.

When you partition your HTML application using the Partition Workshop, iPlanet UDS creates a default configuration for the application based on the application’s service objects. Both the “Handlers” and “Services” projects contain service objects that affect this default configuration.

Briefly, the service objects in your HTML applications are:

Service Object	Partition	Visibility	Dialog Duration	Description
<i>html_model</i> AccessService	iPlanet UDS Web Access Server	Environment	Session (can be changed)	The application contains one AccessService service object per model, which provides the primary link between a Web browser and the iPlanet UDS application. It responds to HTTP requests from the Web client by returning the appropriate Web page.
<i>html_model</i> ScannerService	iPlanet UDS Web Access Server	User	Session	The application contains one ScannerService service object per model, which responds to HTTP requests from AccessService by opening the HTML template file in the request and processing the iPlanet UDS tags to generate a Web page. The Scanner will always be in the same partition as the AccessService with which it is associated.
<i>service_name</i> Service	Applicatio n Server	Environment	Message	The application contains one Service service object for each Express service defined in the Business Model Workshop. You can change the replication of this service object to provide load balancing and/or failover.
<i>service_name</i> DBService	Applicatio n Server	User	Session	The application contains one DBService service object for each Service service object. The DBService service object will always be in the same partition as the Service service object with which it is associated.

A pair of service objects is generated for each model in the application. The service objects of included HTML models are by default partitioned with the corresponding service objects of the main model, though they can be put in other partitions. The following two sections provide further information about these service objects.

business_modelServices Service Objects

See *A Guide to Express* and *Customizing Express Applications* for complete information about the *business_modelServices* service objects *service_nameService* and *service_nameDBService*.

One *business_modelService* service object and one *business_modelDBService* service object are generated for each business model. These are generated into the *business_modelServices* project. The HTMLtutApp application's one business model supplier is the HTMLtutorial business model, whose generated service project is therefore the HTMLtutorialServices project. Two service objects are generated into this project, namely:

- HTMLtutorialService
- HTMLtutorialDBService

Replicating these services These service objects work together to manage database interaction. You can replicate these services, which is desirable for any of the following requirements:

- You want to distribute processing in the deployment environment.
- You want to provide different replication options (load balancing or failover) for different service objects.
- You want your application to interact with more than one database.

Each service is bound to a particular database session, so you must create a separate service for each database you wish to access. You can then assign the corresponding service objects to the nodes that have the appropriate resources.

Creating new services You create new services in the Business Model Workshop, using the Component > New Service command (described in *A Guide to Express*). Each time you create a new service, Express adds a Service service object and a DBService service object to your application.

If you have multiple business models in your application, you must consider your application usage patterns to determine how best to partition the business model services. Technote 10467, which you can access from the iPlanet UDS Support and Services Web page, provides a discussion of this subject.

html_modelHandlers Service Objects

See [Chapter 1, “WebEnterprise Designer Application Architecture”](#) for information on the superclasses on which the *html_modelHandlers* services are based, namely HTMLScanner and HTTPAccess. These are found in the HTTP library. For complete information on all WebEnterprise Designer and WebEnterprise projects and libraries, see iPlanet UDS online help.

One *html_modelAccessService* service object and one *html_modelScannerService* service object are generated for an HTML application. These are generated into the *html_modelHandlers* project. These service objects work together to field HTTP requests from the client browser by way of the Web server. The *html_modelAccessService* and *html_modelScannerService* are bound together, and will always be in the same partition. The ScannerService is user-visible and therefore can be referenced only by the AccessService in the same partition. The AccessService retains a reference to a session object for each user, which includes the user’s result set. The ScannerService refers to the result set when processing a page.

When you partition your HTML application, you must assign the partition that contains the AccessService service object to an iPlanet UDS *server* node. You cannot assign the AccessService service object to a client node.

Failover and load balancing The AccessService service object can be replicated for load balancing or failover. (Failover happens whether you replicate or not if Autostart is enabled; you might want to replicate for failover if Autostart is disabled, however.) If the replicates are partitioned to different nodes, they can all use the same unique port number (the WebEnterprise Listening Port number) assigned to the application when it was created.

When there are multiple AccessService service object replicates partitioned to the same node, each replicate must be assigned a unique port number. This is achieved by specifying a range of port numbers with the FORTE_WW_PORTS environment variable. If FORTE_WW_PORTS is set, the HTTPAccess service object randomly selects a port number from the range and uses it as the listening port number. When a user first accesses a WebEnterprise Designer application from a browser, the request is routed to one of the replicates. A cookie is sent back to the browser with the necessary information to get back to the same replicate. Subsequent requests will include that cookie, so that the user will be routed to the replicate that includes her session object.

The syntax for this environment variable is:

FORTE_WW_PORTS *startrange-endrange*

The startrange and endrange numbers should be set to integer values between 1025 and 65535, and must designate ports that will not be in use on the node when the deployed application is run. For example:

```
setenv FORTE_WW_PORTS 6100-6300
```

The ScannerService service object is bound to the AccessService service object. Each replicate of the AccessService service object has its own copy of the ScannerService service object.

NOTE We recommend that the partitions that contain the AccessService and the ScannerService service objects be compiled, to provide improved performance for the partition.

CAUTION If you compile the partition that contain an AccessService and a ScannerService service object, you must be sure to install the compiled HTTP library in the deployment environments where your iPlanet UDS HTML application will be running. The partition that contains these service objects 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.

If you have multiple HTML models, there will be a pair of service objects for each model. If you are running the application on a machine with native threads, you may choose to include the AccessService and ScannerService service objects from multiple models in the same partition.

Relationship Between the Service Objects

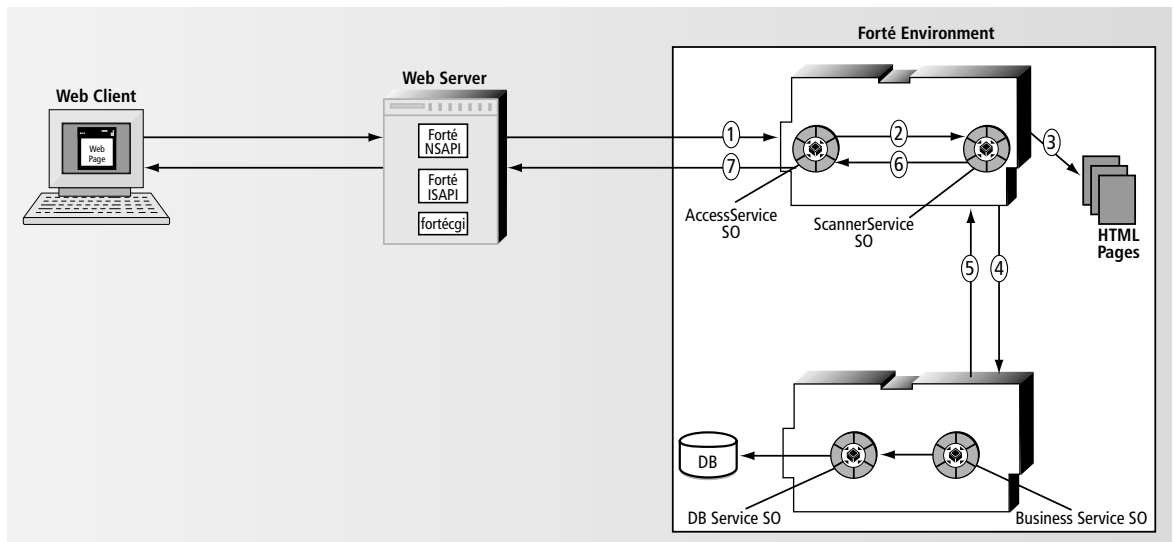
The interactions between the four service objects in the HTML application are shown in [Figure 9-1](#), below:

1. The AccessService service object receives HTTP requests for pages from fortectgi or an iPlanet UDS Web server plug-in on the Web server.
2. The AccessService service object invokes a method on the ScannerService service object, requesting the appropriate page and passing any parameter values relevant to the page.

3. The ScannerService opens the requested HTML page template and processes the iPlanet UDS tags to create the page, and executes methods, which may request data from the database.
4. Database requests are sent to the business service application server where the DBService provides the database session used by the Service service object to access the database.
5. Result sets are returned to the ScannerService, which processes them and populates the page template with the result set. The entire result set is cached in the service in case the user needs to browse through the result set.
6. The ScannerService sends the requested page information back to the AccessService service object.
7. The AccessService service object delivers the HTML information to the Web server, which passes it to the client Web browser and displays the page.

Figure 9-1 illustrates:

Figure 9-1 Relationship between AccessService, ScannerService, Service, and DBService



On the application server partition, the DBService service object is a user-visible DBSession service object, private to the partition. The Service service object is an environment-visible, shared object in the application. The user-visible service object can be accessed only *through* the environment-visible service object—other partitions cannot access it directly. The DBService service object is protected from inappropriate access, because no other partitions can access it.

The same relationship that exists between the two business model service objects also exists between the HTML model service objects. The ScannerService service object is user-visible and private to the partition, and the AccessService service object is environment-visible and shared.

Creating a Default Partitioning Configuration

When iPlanet UDS partitions an application, it assigns all compatible service objects to the same partition. iPlanet UDS assigns the Express business model service objects to one partition and the WebEnterprise service objects to another.

Exceptions to default configuration The default partitioning will be altered if any of the following conditions are true:

- any of the service objects have replication turned on

Replication is discussed in *A Guide to WebEnterprise*.

- external resource managers for the service objects differ in type or location

This could occur if pages are built from multiple business models that have different resource managers. (See the *iPlanet UDS System Management Guide* or *Getting Started with WebEnterprise Designer* for information on setting a resource manager for a business model.) In this case, the Partition Workshop assigns each server partition to the node where the required external resource is installed.

See *A Guide to the iPlanet UDS Workshops* for information on the Partition Workshop and partitioning in general.

► **To partition an HTML application**

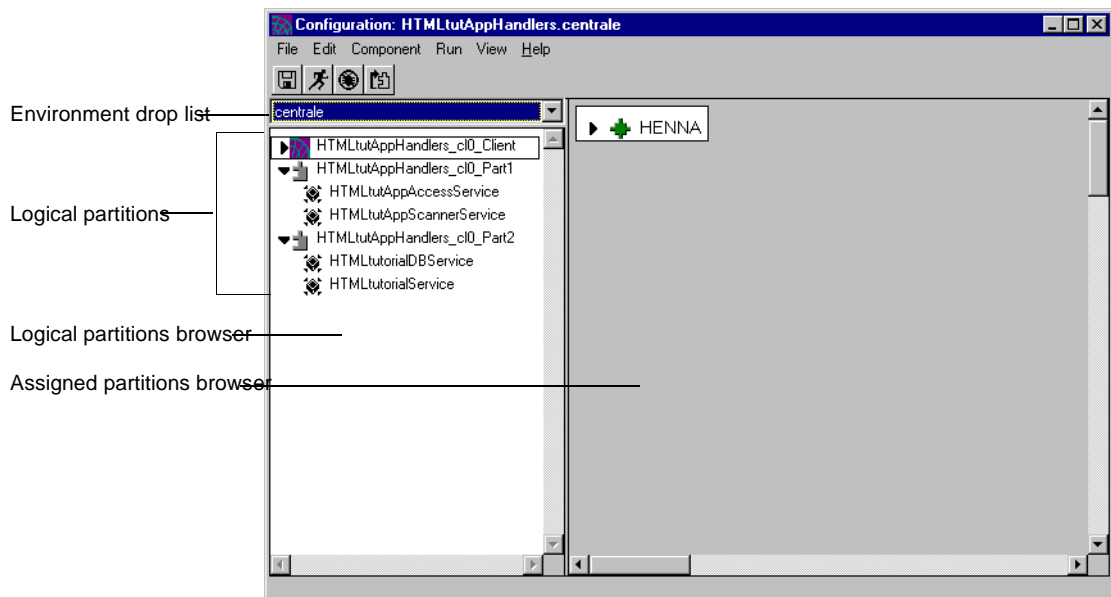
1. In the Repository Workshop, select the *html_modelHandlers* project and click the Partition button.



Alternatively, you can choose the Plan > Run > Partition command. Or, you can open the *html_modelHandlers* project and click the Partition button or choose Run > Partition from there.

The Partition Workshop is displayed with the default configuration for the HTML application. **Figure 9-2** shows the default configuration for the WebEnterprise Designer tutorial example.

Figure 9-2 Default Configuration for HTMLtutApp



The client partition (*html_modelHandlers_C10_Client*) is used to keep the server partitions running in the development environment and launch, if selected, the browser window. In a deployed environment, it has no function. Because of this, the process of deploying and running a WebEnterprise Designer application is different from other iPlanet UDS applications. The WebEnterprise Designer process is described in **“Deploying the Application”** on page 256.

Unassigned service objects If you have any DBService service objects in the application that cannot be supported in the current environment (for example, because the appropriate database resource is not defined in the environment), both the DBService service object and its corresponding Service service object will be *unassigned*. To run your application in this environment, you must either update the DBService service object definition so it can be assigned to a partition, or you must define the appropriate resource manager in your environment. (See *A Guide to Express* for information about updating the DBService service object definition.) You can then assign the partition to the appropriate node.

Common variants to the default configuration There are several changes you might wish to make to the default configuration:

- replicate the AccessService service object for load balancing and/or failover and mark the assigned partition to be compiled
- replicate business service object partitions for load balancing and/or failover, and mark the assigned partitions to be compiled

Modifying the Configuration

The same methods and considerations that apply to other WebEnterprise applications apply to WebEnterprise Designer applications. Please refer to *Getting Started with WebEnterprise Designer* for information on modifying your HTML application configuration. Remember to set the FORTE_WW_PORTS variable if you replicate the AccessService SO partition on the same node. Refer to [“html_modelHandlers Service Objects” on page 249](#) for more information.

Testing the Application in a Distributed Environment

Once you have put the HTML application through a test run from the HTML Application Model Workshop, you should test it in a distributed environment from the Partition Workshop.

► **To test run an HTML application in distributed mode from the Partition Workshop**

1. In the Partition Workshop (displaying the *html_modelHandlers* configuration), click the Run button.



Alternatively, you can choose the Run > Run command.

2. When you are done testing the application, exit it and close the Partition Workshop.

If you are unable to exit the application, you can use the Run > Cancel Run command at any time to cancel execution.

This command cancels the client partition for the application. The remote partitions will still continue to run; use the Run > Stop Remote Partitions command to stop remote partitions.

► **To test run your application in distributed mode from other workshops**

NOTE You can only do this after you have partitioned your application once.

1. In the Repository Workshop, select the HTMLtutAppHandlers project and choose Plan > Run > Run Distributed.

Alternatively, open the *html_modelHandlers* project and choose Run > Run Distributed.

2. When you are done testing the application, exit it and close the Partition Workshop.

If you are unable to exit the application, you can use Plan > Run > Cancel Run (Repository Workshop) or Run > Cancel Run (Project Workshop) at any time to cancel execution.

This command cancels the client partition for the application. The remote partitions will still continue to run; use the Run > Stop Remote Partitions command to stop remote partitions.

See *A Guide to the iPlanet UDS Workshops* for further information about testing applications in a distributed environment.

Deploying the Application

To deploy a WebEnterprise Designer 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.

The application distribution is a representation of the application outside the repository—you use the application distribution to install the application in an environment.

► **To deploy your HTML application**

1. In the Repository Workshop, double-click the main project (the *html_modelHandlers* project) for your application.

This opens the Project Workshop for the *html_modelHandlers* project.

2. In the Project Workshop, choose the File > Configure as > Server command.

This opens the Partition Workshop for the *html_modelHandlers* project.

NOTE If, after configuring for deployment, you want to modify this application and run it from the development environment, you must reconfigure the *html_modelHandlers* project as a client. (And then you must redistribute your application.)

3. In the Partition Workshop, select the desired environment from the environment drop list, if it is not already selected.

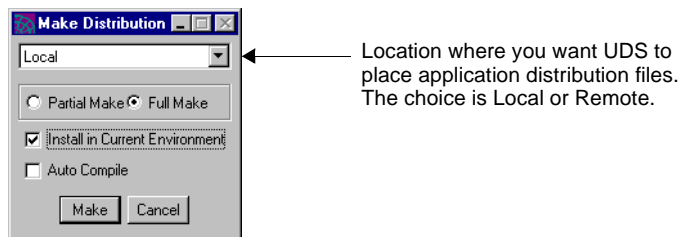
The Partition Workshop opens the configuration for the project or creates a default configuration if one does not already exist.

4. Modify the configuration as desired.

See *A Guide to WebEnterprise* for information on modifying the default configuration.

5. When your configuration is the way you want it, use the File > Make Distribution command to make the application distribution.

iPlanet UDS displays the Make Distribution dialog.

Figure 9-3 Make Distribution Dialog

6. Specify where you want the application distribution files placed by selecting an item from the drop list.
7. Specify whether to perform a Partial Make or Full Make distribution.

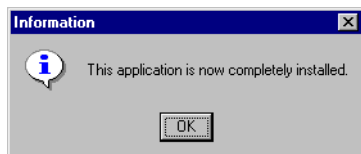
The first time you make a distribution, you should choose the Full Make option. Subsequently, you can choose Partial Make. A Partial Make creates the distribution only for those components that have changed since the last make.

NOTE 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 must compile the partitions by hand (see *A Guide to the iPlanet UDS Workshops* for information about compiling partitions).

You should also make sure that you have marked the appropriate service objects as compiled.

8. Click the Make button.

iPlanet UDS creates the distribution files and installs them on your machine. When finished, iPlanet UDS displays the following message to inform you that the distribution is complete.



9. If you are deploying in more than one environment, select the next environment, make modifications to the configuration if necessary, and use the File > Make Distribution command to create the distribution.

Making the Application's Template Files Accessible to the ScannerService SO

Your HTML template files that iPlanet UDS generated in subdirectories of the document root directory must be accessible to the ScannerService service object. If you have replicated this service object, these files must be accessible to all replications.

NOTE This does not apply to the document root/.base directory or its files. These are for development purposes only. For information on the .base directory, see ["Generated Maintenance Files and Directory" on page 95](#).

To make the application's generated template files accessible to the ScannerService service object, you must copy them to the node where the Scanner's partition exists, in the directory specified by the document root property for that application.

Copying the Application's Template Files to the Scanner Partition

You can copy the document root directory containing the subdirectory of the application's templates from your development environment to the environment of your Scanner Service partition using either of the following techniques:

- Copy the application's template file directory from one environment to the other using file copy, FTP, or any other methods of your operating system.

You can perform this copy either before or after deployment.

- After you make the distribution, but *before* you install it, copy the application's template file directory into the Scanner Service partition folder under the `$FORTE_ROOT/appdist/partition_id` directory.

At deployment time, the files will be copied into the partition's machine.

The *partition_id* directory name is constructed from the partition name in the Partition Workshop. For example, the partition containing the ScannerService service object in [Figure 9-2 on page 253](#) is named `HTMLtutAppHandlers_C10_Part1`. The *partition_id* directory will be named `HTMLtu1`.

For complete information on this technique, see the *iPlanet UDS System Management Guide*.

Once the template files are on the target node, either move them to the location specified by the application's document root, or set the document root to the directory the files are in. The next section provides information on setting a WebEnterprise Designer application's document root.

Setting Document Root on the ScannerService Service Object Partition

The Scanner Service must be aware of the top-most directory of the copied files. This top-most directory is called the document root. You can set the value of an HTML application's document root using any of the following methods, given in order of precedence:

1. Use the `-docroot` command-line argument when starting the partition containing the ScannerService service object.
2. Set the value of the `FORTE_WW_DOCUMENT_ROOT` environment variable.

This must be defined in portable (not local) file syntax (such as `"%{FORTE_ROOT}/html/docs"`).

3. Set the value of the DocumentRoot attribute of the ScannerService service object in the HTML model properties.
4. Use the default value, which is `"%{FORTE_ROOT}/html/docs."`

NOTE These methods and priorities regarding the value of document root are slightly different than those discussed in *A Guide to WebEnterprise*, and apply specifically to WebEnterprise Designer.

Running the Application

After you have deployed your application, you can run it.

► **To run a deployed HTML application**

1. Start the Environment Console.

If you did not enable the Install in Current Environment option when you created your distribution (see [Figure 9-3 on page 257](#)), first load (using File > Load Distribution) and then install (using Component > Install) the application.

2. Double-click the node on which you made the distribution to open it.

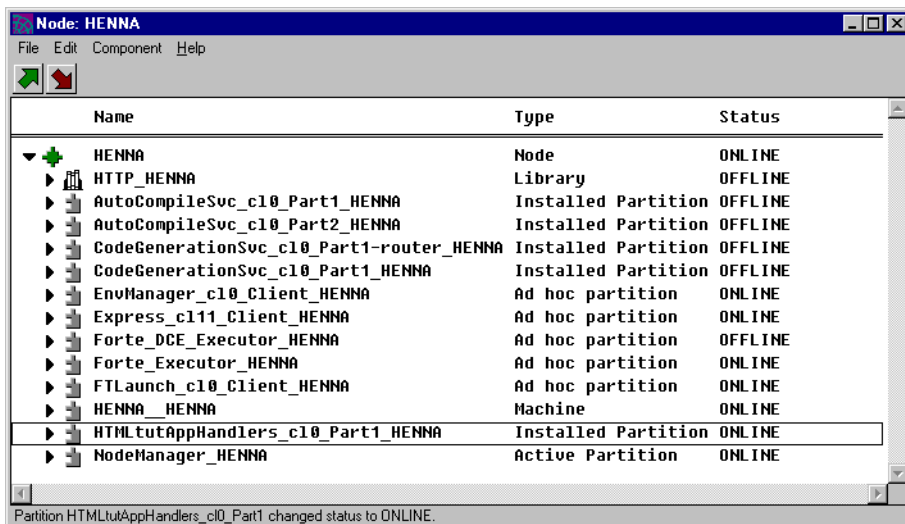
Alternatively, select the node and choose Component > Open.

3. From the node window, click the triangular control to display the node's contents.
4. Select the application and click the Start Up button.



Alternatively, select a server partition and choose Component > Start Up.

When the partitions are marked “online,” your application is running. When the AccessService service object starts up, it will register itself with the Web server. Your Web server needs to be running when the AccessService partition starts.

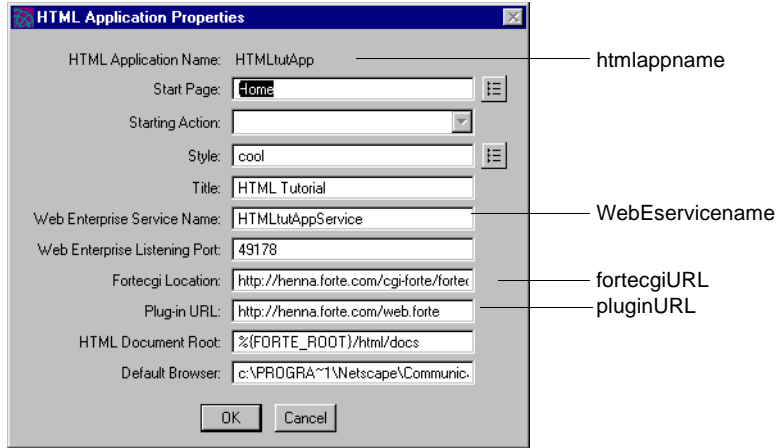


5. Start your browser, if it is not already running.
6. Type in a URL with the following syntax:

```
[pluginURL/fortecgiURL]?servicename=WebEservicename
&templatenamename=htmlappname/Start.html
```

Note that this is all one string with no breaks. You obtain the values for the variables in this string from the HTML Application Properties dialog, as shown in [Figure 9-4](#). It is also found in the log for the Access partition.

Figure 9-4 Obtaining URL Elements from the Properties Dialog



For example, a string using the values in **Figure 9-4** would be:

```
http://henna.forte.com/web.forte?servicename=HTMLtutAppService
&templatename=HTMLtutApp/Start.html
```

See the *iPlanet UDS Programming Guide* for complete information about partitioning applications and making distributions. See the *iPlanet UDS System Management Guide* for information on using the Environment Console.

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 the *WebEnterprise Installation Guide* for information. The library must be installed on any iPlanet UDS node that runs a partition that includes an AccessService service object.

NOTE If you install the compiled HTTP libraries on UNIX, follow these instructions: after you have compiled a partition that accesses the compiled HTTP library, *do not* move the compiled HTTP library (for example, by moving \$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.

If you install the compiled HTTP libraries on NT, you need to add the \$FORTE_ROOT/userapp/http/c10 directory to the "PATH" environment variable.

The AccessService Log File

When the partition that includes the AccessService is started, its log file will include messages indicating that it has been successfully connected to the Web server, and give the starting URL for the application. the partition log; for more information, see the *iPlanet UDS System Management Guide*.

The following message indicates that this service started successfully:

```
Enable access to HTMLtutAppService at port 6203 from
http://henna.forte.com/cgi-forte/fortecgi.exe (plugInURL =
http://henna.forte.com/web.forte)
```

The following message indicates the starting page for the application:

```
*** URL of starting page = http://henna.forte.com/web.forte?ServiceName=
HTMLtutAppService&TemplateName=HTMLtutApp/Start.html
```

If your application includes multiple HTML models, there will be several such messages. Your users will only need to know the starting URL for the main model. If that model references other models, its links will have the proper URL.

Memory Considerations

Because of the nature of Web applications, the result set of data for a user is maintained in memory in the partition that includes the `AccessService`. The `AccessService` holds a session object for each user that includes that result set. The session object will be cleared and garbage collected when the session times out (which, by default, is irrespective of whether the user exits the browser or otherwise quits the application). The default session timeout is set with the `ExpressHTTPAccess.SetupSessionManagement` method. For a WebEnterprise Designer application, the timeout default is one hour.

You can change the application's timeout setting with the Page Handler Customization Wizard customization point "Modify session timeout" under the Application category.

When you set the maximum memory for partitions that include an `AccessService`, you must take the timeout setting into account, as well as the number of concurrent users you will have and the average size of the result sets they will work with. See the *iPlanet UDS System Management Guide* for information on setting the memory parameters for a partition.

Index

A

- AccessServer partition
 - and WebEnterprise Designer architecture 27
 - location on a server node 249
 - when to compile 249
- AccessService SO
 - and error processing 152
 - and port numbers 249
 - description of functions 27, 250
 - load balancing considerations 249
 - replication status 249
 - startup (object interaction diagram) 44
- Action URL parameter 58
- adata style element 142
- AddConstraints method 53
- AddError method 82
- AddForeignData method 55
- Adding data to command link example 177
- alabel style element 142
- alink style element 142
- Always Generate Custom Classes option
 - disabling to delete classes 74
 - effect on class hierarchy 66
 - enabling 67
 - of Custom Generation Options command 66
- amenu style element 142
- Association IDs
 - and business class page fields 51
 - and the GetFieldAttrID method 52
 - and the GetFieldIndexID method 52
 - description 51

Attributes

- IDs and result sets 84
 - programmatically changing value of 84
- ## Authenticating users
- integrating with authentication system 234
 - using a logon page 231
 - using ExpressHTTPAccess.LogonSession 46
 - using logon information 54

B

- Bad selector errors 134
- .base directory
 - function 103
 - meaning of file extensions in 104
- BeforeInsert method
 - and processing custom fields 90
 - customization 195, 214
- BeforeSearch method 90
- BeforeUpdate method 90
- bus_class_pageHandler* class, *See* Page handler classes
- Business class pages
 - generated HTML templates, fancy 98
 - generated HTML templates, generic 94
 - generated HTML templates, simple 97
 - modeling as a lookup page 169
 - restricting access to 237
 - transferring data between classes and 52
- Business classes, transferring data between pages and 52

Business models, generated service objects 246–252

Business rules, customizing 87

Business services, generated service objects 247–252

business_modelServices project 246

BusinessClass class

- attribute IDs 84
- InstanceStatus attribute 83
- record status 83

BusinessClass objects

- and attribute IDs 84
- and InstanceStatus attributes 83
- and their UpdateQuery attribute 85
- checking whether updated 85
- undoing changes 85

businessClass_qq_attribute.inc file 219

BusinessClient class, Revert method 85

buttonlogon style element 143

buttons style element 143

C

captionform style element 142

captionlist style element 142

captionlogon style element 142

captionmenu style element 142

class identifiers

- and page styles 139
- scope 140

Class interaction diagram of WebEnterprise Designer classes 39

Code generation directives

- and customization 124
- and page design templates 122
- description 122
- page object 134

Component menu

- New Service command 248
- Open command 260
- Start Up command 260

Configure as Server command 256

ConnectManagedSessions method 241

Cookies, in WebEnterprise applications 230

cool.css file, location 138

CSS1 style sheets, *See* Styles

CurrentRow attribute 55

Customizable classes

- and ExpressHTTPAccess 46
- automatically creating 66
- deferring deletion 75
- deleting page handler classes 72

Customization examples

- adding a lookup link 166
- adding data to command links 177
- creating a drop or radio list manually 216
- customizing a page design 124
- customizing logon validation 235
- customizing styles 144
- displaying the record just inserted 203
- entering and formatting dates 191
- list of examples in examples chapter 164
- managing subsidiary applications 241
- mandatory field 212
- modifying session timeout 244
- populating data on an Insert page 185
- removing a validation from a page mode 198
- restricting access to pages 238
- setting or removing a radio or drop list
 - default 222
- whole field validation 206

Customizations, preserving

- file types in .base directory 104
- function of .base directory 103
- HTML changes only 103
- lookup files 221
- model changes 103
- model changes, conflicting 104
- summary 102

Customizing application security

- customizing login validation example 235
- customizing main models 240
- customizing session timeout example 244
- customizing subsidiary models 240
- integrating with an authentication system 234
- restricting access to pages 237
- sharing security environment across
 - applications 239
- using logon pages 234

- Customizing error pages
 - and `HTMLScanner.GetErrorTemplate` method 158
 - and `HTTPAccess.GetErrorTemplate` method 161
 - application-specific error template example 159
 - application-specific template with custom data example 160
 - creating your own 156
 - default error processing steps 152
 - error class-error template mapping 152
 - error directory 152
 - errors in customization 162
 - modifying default error pages 156
 - page design templates for 118
 - result set variables 154
 - with the Customization Wizard 157
- Customizing generated HTML templates
 - customizing a field on a search page 106
 - customizing font size on a data page 109
 - identifying hidden fields 102
 - identifying `WebEnterprise` tags 102
 - identifying `WebEnterprise` variables 102
 - linking to the Insert page URL 172
 - passing a selected value as a parameter 174
 - preserving customizations, *See* Customizations, preserving
 - removing validation from a single template 198
 - setting or removing a radio or drop list default 222
 - types of customization 100
 - using a variable from the calling page's Data template 189
 - when customizing styles 147
 - where and where not to customize 101
 - whole field validation 206
 - without the Customization Wizard 166
- Customizing generated lookup files
 - manually customizing drop lists or radio lists 216
 - preserving customizations 221
 - without the Customization Wizard 166
- Customizing generated scripts files
 - setting or removing a radio or drop list default 222
 - without the Customization Wizard 166
- Customizing page designs
 - bad selector errors 134
 - code generation errors 132
 - customizing menus 127
 - description of code generation directives 122
 - elements of design files 117
 - example 124
 - identifying a design in the Page Handler dialog 125
 - role of data template 128
 - role of display template 131
 - when to customize 117
- Customizing page handler classes
 - adding data to command links example 177
 - displaying the record just inserted 203
 - entering and formatting dates 191
 - lookup link example 166
 - making a field mandatory 212
 - populating data on an Insert page example 185
 - removing validation from a Search page 198
 - techniques with business rules 87
 - techniques with `ClassHandler` classes 86
 - where to 80
 - without the Customization Wizard 165
- Customizing styles
 - adding new elements 147
 - basic procedures 145
 - browser caching problems 150
 - creating a new style sheet 146
 - identifying a style in the Page Handler dialog 148
 - making a style browser-independent 150
 - only modifying existing elements 146
 - scope considerations 144
- Customizing `WebEnterprise Designer` classes
 - creating a single customizable page handler class 66
 - creating customizable classes, overview 65
 - creating customizable page handler classes for all pages 66
 - customization techniques 86
 - error reporting 82
 - general considerations 64
 - global customization 90
 - how to override methods 80
 - local vs. global 82
 - result sets, manipulating 88
 - techniques with business rules 87

D

Data templates

See also Page modes

customization examples 174, 181, 189

modifying the design of 128

Data types, declared type vs. runtime type 40

dataform style element 143

datalist style element 143

Dates, formatting

creating drop lists for 193

example 192

TOOL code for 195

DBService SO

See also *service_name* DBService

description of function 251

replicating 248

DecodeValue method 52, 53, 89

Decoding data, from a WebEnterprise request 53

Decoding or validating a field (customization) 214

Define HTML template for exceptions

(customization) 157

Define subsidiary applications (customization) 47, 240

Deleting customizable page handler classes 72

Deploying WebEnterprise Designer applications

as client or server 256

installing HTML template files on Scanner

partition 258

procedures 256–259

telling the Scanner SO where HTML pages are 259

Diagrams

AccessService startup 44

class interactions, WebEnterprise Designer classes 39

ExpressHandlers class hierarchy 34

HTML application runtime anatomy 27

HTTP class hierarchy 33

life of a template request 48

partitions and objects at runtime 30

scanner startup 47

display style element 142

Display templates, modifying design template of 131

Displaying the record just inserted example 203

DisplayPage method 50, 53, 89

–docroot command-line flag 259

Document root

accessibility of directory to the ScannerService SO 258

default specification 259

defined by

FORTE_WW_DOCUMENT_ROOT 259

DoInsert method, customization 203

DoSearch method 90

Drop list customizing examples

manually modifying lookup file 216

removing or setting the default 222

E

EnableAccess method 46

Entering and formatting dates example 191

entry result set 60

entry.CurrentRow_ busClassPage variable 62

entry.CurrentRowIndex_ busClassPage variable 62

entry.FirstVisibleRow_ busClassPage variable 62

entry.list_ busClassPage variable 62

entry.RequestStatus_ busClassPage variable 62

entry.Rows_ busClassPage variable 62

entry.VisibleRows_ busClassPage variable 62

Environment variables

FORTE_WW_DOCUMENT_ROOT 259

FORTE_WW_EDITOR 72

FORTE_WW_HTMLEDITOR 72, 165

FORTE_WW_PORTS 249

error directory 152

Error handling

See also Customizing error pages

customization of 156

default process 151

error directory 152

Error pages, page design templates for 118

ErrorMgr class, AddError method 82

Express Services classes, and the Web application server 31

- ExpressClassHandler class
 - AddConstraints method 53
 - BeforeInsert method 90
 - BeforeInsert method customization 195, 214
 - BeforeSearch method 90
 - BeforeUpdate method 90
 - DecodeValue method 52, 53, 89
 - decoding data from a WebEnterprise request 53
 - DisplayPage method 50, 53, 89
 - DoInsert method customization 203
 - DoSearch method 90
 - FillResultSet method 53
 - FindHandler method 50
 - formatting data into a result set 52
 - FormatValue method 52, 88
 - GetFieldAttrID method 52
 - GetFieldIndexID method 52
 - GetPageData method 87
 - HandleCondition method 27, 50
 - HandleTag method 27, 50
 - implementing the TagHandlerIFace interface 48
 - NewQuery method 86
 - ProcessAction method 50
 - referenced objects 51
 - role in class structure 48
 - transferring data between classes and pages 52
 - UpdateAttr method 84
 - UpdateClass method 53
 - ExpressHandler class, FindHandler method 54
 - ExpressHandlers project
 - class hierarchy 34
 - subclassing for global customization 90
 - ExpressHTTPAccess class
 - customizing 46
 - EnableAccess method 46
 - functional description 44
 - HasLogonHandler attribute 46
 - LogonSession method 46
 - SetSessionManagement method 244
 - Setup method 44
 - SetupAccess method 45, 46
 - SetupSessionManagement method 46, 264
 - ExpressLogonHandler class
 - functional description 54
 - HandleCondition method 54
 - ExpressLogonHandler class, HandleCondition method 234
 - ExpressLookupInfo class
 - functional description 56
 - GetDisplayedNullValue method 56
 - ExpressPageData class
 - AddForeignData method 55
 - and foreign result sets 55
 - CurrentRow attribute 55
 - FirstVisibleRow attribute 55
 - functional description 55
 - ExpressScanner class
 - functional description 47
 - Handlers attribute 54
 - ExpressValueGenerator class 55
 - ExpressWindows project
 - class hierarchy 34
- ## F
- Field indexes
 - and business class page fields 51
 - and the GetFieldAttrID method 52
 - and the GetFieldIndexID method 52
 - converting between 52
 - Fields, hidden
 - examples 107
 - identifying in HTML templates 102
 - FillResultSet method 53
 - FindHandler method 50, 54
 - FirstVisibleRow attribute 55
 - Folder pages, and ExpressPageData 55
 - footer style element 142
 - Form pages, page design templates for 119
 - FormatValue method 52, 88
 - FORTE result set 60
 - FORTE tags (FORTE EXECUTE, FORTE IF, FORTE ITERATE, etc.), *See* WebEnterprise tags
 - FORTE.ExecURL variable
 - description 60
 - example 107

- FORTE.UniqueID variable
 - description 61
 - example 111
- FORTE_WW_DOCUMENT_ROOT environment variable
 - and deployment 259
 - constraints 259
 - setting 259
- FORTE_WW_EDITOR environment variable 72
- FORTE_WW_HTMLEDITOR environment variable 72, 165
- FORTE_WW_PORTS environment variable 249

G

- Generated .base directory files 104
- Generated classes
 - customizing 66
 - from business model, *See Customizing Express Applications*
 - from HTML application model 35
 - modifying 79
- Generated HTML pages, *See Generated HTML templates*
- Generated HTML templates
 - business class pages, fancy 98
 - business class pages, generic 94
 - business class pages, simple 97
 - customizing a field on a search page 106
 - customizing font size on a data page 109
 - identifying hidden fields 102
 - identifying WebEnterprise tags 102
 - identifying WebEnterprise variables 102
 - installing on Scanner partition 258
 - link pages, generic 97
 - logon pages, generic 97, 232
 - Main template 95
 - preserving customizations, *See Customizations, preserving*
 - Scripts template 95
 - Start page 96
 - summary 37
 - where and where not to customize 101
- Generated lookup files 96

- Generated maintenance files 95
- Generated TOOL code, for logon pages 232
- generic.html file
 - corresponding exceptions 152
 - result set variables 155
- GetDisplayedNullValue method 56
- GetErrorTemplate method
 - html_model*Access class 161
 - html_model*Scanner class 158
 - HTMLScanner class 158
 - HTTPAccess class 161
 - warning 162
- GetFieldAttrID method 52
- GetFieldIndexID method 52
- GetPageData method 87
- GetTextData method 82
- Global customization
 - and supplier plans 92
 - procedures 90
 - when to do 82

H

- HandleCondition method
 - and logon pages 234
 - and runtime control flow 50
 - and the ScannerService SO 27
 - and user authentication 54
 - and WebEnterprise tags 48
 - defined in TagHandlerIFace 48
- Handler classes, *See Page handler classes*
- Handlers attribute 54
- HandleTag method
 - and runtime control flow 50
 - and the ScannerService SO 27
 - and WebEnterprise tags 48
 - defined in TagHandlerIFace 48
- HasLogonHandler attribute 46
- header style element 142
- HTML 4.0, and WebEnterprise Designer 137
- HTML application model
 - classes generated from 35

- HTML applications
 - load balancing 28
 - HTML document root, *See* Document root
 - HTML templates
 - and page design elements 117
 - using links 56
 - html_model* HTML templates, *See* Generated HTML templates
 - html_model*Access class
 - and customizing ExpressHTTPAccess 46
 - ConnectManagedSessions method 241
 - GetErrorTemplate method 161
 - IsPrimarySession method 240
 - LogonSession method 234
 - SetSessionManagement method 244
 - html_model*AccessService SO
 - See also* AccessService SO
 - and error processing 152
 - and the deployment process 249
 - and the ExpressHTTPAccess.Setup method 45
 - description 247
 - html_model*Handlers project
 - customizable subclasses 37
 - description 35
 - HTML application's main project 246
 - service objects of 249
 - html_model*Scanner class
 - and the *html_model*ScannerService SO 47
 - description 28
 - GetErrorTemplate method 158
 - html_model*ScannerService SO
 - See also* ScannerService SO
 - and error processing 152
 - and runtime control flow 50
 - and the deployment process 249
 - description 247
 - object interaction diagram of startup 47
 - HTMLScanner class, GetErrorTemplate method 158
 - htmlscanner.html file
 - corresponding error class 152
 - result set variables 154
 - HTMLScannerException class
 - attribute-result set variable list 154
 - corresponding error file 152
 - HTTP library
 - and compiled partitions 250
 - class hierarchy 33
 - installing compiled 263
 - HTTPAccess class
 - GetErrorTemplate method 161
 - SessionCreationURL attribute 54, 233
 - httpaccess.html file
 - corresponding error class 152
 - result set variables 155
 - HTTPAccessException class
 - attribute-result set variable list 155
 - corresponding error file 152
- ## I
- id identifier, and page styles 141
 - Insert command
 - adding a JavaScript call to 211
 - adding data to 178, 186
 - and ExpressValueGenerator class 55
 - customizing BeforeInsert method 195, 214
 - Insert templates
 - See also* Page modes
 - customization examples 172, 183, 190, 193, 206, 209, 224
 - InstanceStatus attribute 83
 - Is subsidiary application (customization) 47, 240
 - IsPrimarySession method 240
- ## J
- JavaScript examples
 - adding call to Insert button 211
 - boilerplate script 207
 - location of validation scripts 198
 - making a field mandatory 209
 - removing a default from a drop list 224, 226
 - removing a script call from a template 200
 - JavaScript fields, page design templates for 121

- ## L
- labelform style element [143](#)
 - labellist style element [143](#)
 - labelmodify style element [143](#)
 - Link pages
 - HTML templates generated for [97](#)
 - page design templates for [120](#)
 - Link tags, identifying page styles with [139](#)
 - LinkName URL parameter [58](#)
 - List pages, page design templates for [119](#)
 - listentry result set [60](#)
 - listentry.*busClassPage_qq_fieldName* variable [62](#)
 - example [109, 175, 181](#)
 - listentry.qqRowNumber variable [62](#)
 - Load balancing
 - and port numbers [249](#)
 - with AccessService SOs [28, 249](#)
 - Local customization [82](#)
 - Logon pages
 - and SESSION_REQUIRED [233](#)
 - code generated for [232](#)
 - creating [231](#)
 - customizing [234](#)
 - how activated [233](#)
 - HTML templates generated for [97](#)
 - validation fields [232](#)
 - Logon Validation (customization) [235](#)
 - LogonSession method [46, 234](#)
 - Lookup files
 - businessClass_qq_attribute.inc* file [219](#)
 - generated [96](#)
 - manually customizing drop lists or radio lists [216](#)
 - preserving customizations in [221](#)
 - Lookup link example [166](#)
- ## M
- Main projects
 - and partitioning [246](#)
 - html_modelHandlers* project [246](#)
 - Main template [95](#)
 - Make Distribution command [256](#)
 - manifest.txt file [95](#)
 - Memory considerations, and session timeout [264](#)
 - Menu design templates, modifying [127](#)
 - menu style element [142](#)
 - Menus, creating in a page design [127](#)
 - Methods, how to override [80](#)
 - Modify session timeout (customization) [244, 264](#)
 - modify style element [142](#)
 - MsgCatalog class, GetTextData method [82](#)
- ## N
- Nested pages
 - adding data to command links of [178](#)
 - and ExpressPageData [55](#)
 - New HTML Page command [231](#)
 - NewClass class, compared to NewClassForInsert class [86](#)
 - NewQuery method [86](#)
- ## O
- Object interaction diagrams
 - a template request [48](#)
 - AccessService startup [44](#)
 - scanner startup [47](#)
- ## P
- Page design templates
 - description of template file names [118](#)
 - fancy [98](#)
 - for different page elements [119](#)
 - for error pages [118](#)
 - for form and list pages [119](#)
 - for JavaScript scripts [121](#)
 - for link pages [120](#)
 - for lookup fields [121](#)
 - for page modes [119](#)

- Page design templates (*continued*)
 - for start pages 118
 - simple 97
 - variations on fancy 99
- Page design templates, selectors
 - _RC_ 118
 - _RR_Name_ 119
 - _UniqueName_ 121
 - Data 119
 - HasJavaScripts 121
 - IsAabModel 118
 - IsDataWindow 119
 - IsFormWindow 119
 - IsLinkWindow 120
 - IsListWindow 119
 - LookupFields 121
- Page designs
 - and code generation 122
 - and menus 127
 - directory of templates 116
 - identifiers 125
 - role in Web page production 116
 - scope 116
 - what they determine 115
 - when to customize 117
- Page handler classes
 - and HandleTag and HandleCondition
 - methods 48
 - BeforeInsert method customization 195, 214
 - connections between 54
 - customization techniques 86
 - description 28
 - DoInsert method customization 203
 - finding by name 54
 - referencing other page handlers 54
 - RestrictInsertAccess method customization 238
- Page Handler Customization Wizard
 - about 67
 - accessing online help 68, 77
 - application-wide customizations 76
 - automatically generating a customizable page handler class 71
 - customizing error pages 157
 - deleting customizations 72
 - indicator that customization exists 73
 - list of all customization points 77
 - Lookup Data Files example 219
 - using 69
- Page handler customizations
 - Application category 76
 - creating an initialized object in a page 86
 - customization techniques 86
 - customizing manually 79
 - Decoding or validating a field 214
 - Define HTML template for exceptions 157
 - Define subsidiary applications 47, 240
 - Fields category 88
 - getting the initial query 86
 - getting the result set 87
 - Insert (Database Operations) 195, 214
 - Is subsidiary application 47, 240
 - list of all customization points 77
 - local vs. global 82
 - Logon Validation 235
 - Modify session timeout 244, 264
 - Restricting access to page modes 238
 - summary of categories 76
- Page modes
 - adding a variable to Insert command link in Data template 181
 - adding customized link to Data template 174
 - adding drop lists in the Insert template 193
 - adding lookup link to Insert template 166
 - adding mandatory field validation to Insert template 206
 - creating drop lists for date formatting on Insert template 191
 - customizing 101
 - page design templates for 119
 - passing data to Insert template 177
 - populating a field in the Insert template 190, 209
 - populating data on an Insert template 185
 - removing default from a drop list 224
 - removing validation from a search template 200
 - restricting access to 238
 - validating a whole form from a search template 206
- Partition command 253
- Partitioning WebEnterprise Designer applications
 - as client or server 253
 - canceling if unable to exit 255
 - default configuration 252
 - main project (*html_modelHandlers*) 246

Partitioning WED applications (*continued*)

- partition_id directory name 259
- procedures 245–254
- service objects 246
- starting server partitions 260
- stopping remote partitions 255
- supplier project (*business_modelServices*) 246
- testing distributed 254
- unassigned service objects 254

PDF files, viewing and searching 22

Populating data on an Insert page example 185

Port numbers

- and AccessService SOs 249
- and FORTE_WW_PORTS 249

ProcessAction method 50

R

Radio list customizing examples

- manually modifying lookup file 216
- removing or setting the default 222

Referenced objects, and ExpressClassHandler class 51

Registration, customizing 46

report.txt file

- and simple HTML changes 103
- conflicting model changes 105
- description 95
- simple model changes 104

#reqstatus style element 143

Restricting access to page modes (customization) 238

RestrictInsertAccess method 238

Result set variables

- creating your own 156
- description 154

Result sets

- and attribute IDs 84
- and BusinessClass.InstanceStatus attributes 84
- and session management features 53
- entry 60
- foreign 55
- formatting data into 52

FORTE 60

- inserted record, changing default position 203
- listentry 60
- manipulating 88
- USER 60

ReturnTemplate URL parameter 59

Revert method 85

rowidlist style element 143

Run menu

- Cancel Run command 255
- Partition command 253
- Run command 255
- Stop Remote Partitions command 255

S

ScannerServer partition

- and WebEnterprise Designer architecture 27
- installing HTML template files on 258
- when to compile 249

ScannerService SO

- accessibility to the document root directory 258
- and error processing 152
- and the HandleCondition method 27
- and the HandleTag method 27
- description of functions 27, 250
- replication status 249

Scripts template 95

Search templates, customization examples 200, 206

Security environment, sharing

- about 239
- restrictions 241

Security, *See* Customizing application security

Selection URL parameter 59

Service objects

- creating 248
- description of interaction 250
- generated from business model 246–252
- generated from HTML model 247–252
- html_modelAccessService* 247
- html_modelScannerService* 247
- service_nameDBService* 247
- service_nameService* 247

- Service objects (*continued*)
 - unassigned 254
 - which can be replicated 249
- Service SO
 - See also service_nameService*
 - description of function 251
 - replicating 248
- service_nameDBService* 247
- service_nameService* 247
- ServiceName URL parameter 57
- Session management
 - and logon pages 233
 - and secure applications 230
 - and shopping cart applications 230
 - authenticating users 231
 - customizing 46
 - for included applications 46
 - how session ids are tracked 230
 - how session timeout works 242
 - ideal timeout setting 243
 - session property without logon page 231
 - sharing security environment across applications 239
 - storing state information in Web applications 230
 - tracking sessions 230
- Session properties
 - when SESSION_AUTOCREATE is used 231
 - when SESSION_REQUIRED is used 233
- Session timeout
 - and AccessService service object 264
 - customizing 244, 264
 - default 264
 - finding the ideal setting 243
 - how it works 242
 - resuming the session 243
- SESSION_AUTOCREATE property 231
- SESSION_REQUIRED property 233
- SessionCreationURL attribute 54, 233
- SessionTimeOut attribute 242
- SetSessionManagement method 244
- Setup method 44
- SetupAccess method 45, 46
- SetupSessionManagement method
 - and session timeout 264
 - role in customization 46
- Start page 96
- Start pages, page design templates for 118
- steel.css file, location 138
- Style elements
 - #reqstatus 143
 - adata 142
 - alabel 142
 - alink 142
 - amenu 142
 - buttonlogon 143
 - buttons 143
 - captionform 142
 - captionlist 142
 - captionlogon 142
 - captionmenu 142
 - dataform 143
 - datalist 143
 - display 142
 - footer 142
 - header 142
 - labelform 143
 - labellist 143
 - labelmodify 143
 - menu 142
 - modify 142
 - rowidlist 143
- Styles
 - and deprecated features 137
 - class identifier 139
 - conformation to HTML 4.0 137
 - customizing, *See* Customizing styles
 - definition of elements 142
 - description 138
 - id identifier 141
 - identifying in link tags 139
 - location of style sheet files 138
 - specifying in HTML template 139
- Subsidiary applications, managing (example) 241
- Supplier plans, *See* Supplier projects
- Supplier projects
 - and partitioning 246
 - business_modelServices* 246

T

- TagHandlerIFace interface 28
- TagHandlerIFace interface, implemented by ExpressClassHandler 48
- Template request, life of
 - detailed description 42
 - general overview 27
 - object interaction diagram of 48
- TemplateName URL parameter 57
- Testing distributed 254
- Three-tier architecture 26

U

- UniqueID variable 112
- Uniquifier URL parameter 59, 112, 175
- UpdateAttr method 84
- UpdateClass method 53
- UpdateQuery attribute 85
- URL parameters
 - Action 58
 - LinkName 58
 - ReturnTemplate 59
 - Selection 59
 - ServiceName 57
 - TemplateName 57
 - Uniquifier 59
 - usage 57
- USER result set 60
- USER.TopPage variable
 - description 61
 - example 111

V

- Validation
 - adding to a field 199
 - and ExpressLookupInfo 56
 - creating a drop or radio list manually 216
 - making a field mandatory 212

- removing a default from a drop list 224
- removing from a page mode 198
- removing from a template 200
- setting or removing a radio or drop list default 222
- whole field validation 206

- Variables, environment, *See* Environment variables
- Variables, WebEnterprise Designer, *See* WebEnterprise Designer variables

W

- Web application server
 - and Express Services classes 31
 - architecture 28
 - example 29
- Web server
 - and Forte Web server plug-in 27
 - and fortectgi program 27
- Web server root directory
 - and graphics files 220
 - and styles 138
- WebEnterprise Designer
 - conformation to HTML 4.0 137
 - customization guidelines 29
 - customizing security, *See* Customizing application security
 - error directory 152
 - error handling 151
 - logical architecture 26
 - projects 31
 - result set variables 60, 154
 - runtime architecture 27
 - runtime scenarios 43
 - session management, *See* Session management
 - styles, *See* Styles
 - URL parameters 57
 - variables 60
- WebEnterprise Designer applications
 - deploying 256–259
 - deploying as client or server 256
 - distributed testing 254
 - generated service objects 246–252
 - main project (*html_modelHandlers*) 246

- WebEnterprise Designer applications (*continued*)
 - partitioning 245–254
 - partitioning, default configuration 252
 - supplier project (*business_modelServices*) 246
- WebEnterprise Designer class structure
 - and server registration 46
 - and session management 46
 - authenticating logon information 46
 - diagram 39
 - role of ExpressClassHandler 48–54
 - role of ExpressHTTPAccess 44
 - role of ExpressLogonHandler class 54
 - role of ExpressLookupInfo class 56
 - role of ExpressPageData 55
 - role of ExpressScanner class 47
 - role of ExpressValueGenerator class 55
- WebEnterprise Designer classes
 - customizable 66
 - working with business classes 83
- WebEnterprise Designer projects
 - ExpressHandlers project 34
 - html_model*Handlers project 35
 - HTTP Library 33
- WebEnterprise Designer URL parameters
 - Action 58
 - LinkName 58
 - ReturnTemplate 59
 - Selection 59
 - Uniquifier 59
- WebEnterprise Designer variables
 - entry.CurrentRow_*busClassPage* 62
 - entry.CurrentRowIndex_*busClassPage* 62
 - entry.FirstVisibleRow_*busClassPage* 62
 - entry.list_*busClassPage* 62
 - entry.RequestStatus_*busClassPage* 62
 - entry.Rows_*busClassPage* 62
 - entry.VisibleRows_*busClassPage* 62
 - examples 172, 173, 175, 181, 189
 - FORTE.ExecURL 60
 - FORTE.*parameter_name* 154
 - FORTE.UniqueID 61
 - listentry.*busClassPage_qq_fieldName* 62, 175, 181
 - listentry.qqRowNumber 62
 - lookup link example 175
 - usage 60
 - USER.TopPage 61
- WebEnterprise error handling 151
- WebEnterprise Listening Port number
 - and FORTE_WW_PORTS 249
 - and load balancing 249
- WebEnterprise tags
 - about 28
 - example of FORTE IF 107, 109
 - format convention used in book 50
 - identifying in HTML templates 102
- WebEnterprise URL parameters
 - ServiceName 57
 - TemplateName 57
 - usage 57
- WebEnterprise variables
 - examples 109
 - identifying in HTML templates 102

