
WebObjects Direct to Web Guide

[Internet & Web](#) > [WebObjects](#)



2007-07-11



Apple Inc.
© 2007 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Logic, Mac, Mac OS, Pages, QuickTime, WebObjects, and Xcode are trademarks of Apple Inc., registered in the United States and other countries.

Enterprise Objects is a trademark of Apple Inc.

Java and all Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Times is a registered trademark of Heidelberger Druckmaschinen AG, available from Linotype Library GmbH.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

Introduction Introduction to WebObjects Direct to Web Guide 7

Organization of This Document 7

Chapter 1 An Introduction to Direct To Web 9

Creating a Direct to Web Project 10

- The Different Looks for WebObjects Applications 11

The Structure of a Direct to Web Project 13

Using Your Direct to Web Application 14

- Launching a Direct to Web Application 14
- The Login Page 15
- Dynamically Generated Pages 16

Customizing Your Application With the WebAssistant 24

Running the WebAssistant With appletviewer 25

WebAssistant Overview 25

Restricting Access to Entities 27

Customizing Pages 27

Setting Which Properties are Displayed 28

Changing How Properties Are Displayed 29

- Textual Attributes and Formatting 30
- Representation of Relationships 30

WebAssistant Expert Mode 32

- Choosing a Page to Customize 33
- Named Configurations 33

Generating Components 34

User Templates 37

- Generating a Template 37

Chapter 2 Direct to Web Architecture 39

Direct to Web Components 39

Direct to Web Templates 40

Direct to Web Reusable Components 41

Property-Level Components 42

Direct to Web Component Organization 42

The Direct to Web Context 43

- Maintaining State 43
- Providing Binding Values 44

The Direct to Web Factory 45

Rendering a Direct to Web Page: An Example 46

- Creating the Query Page 46

- Rendering the Direct to Web Template 47
- Setting the Property Key 48
- Resolving Keys That Depend on the Property 49
- The Rule System 49
 - Deciding Which Candidate Should Fire 50
 - Rules and the Web Assistant 51
 - Rule Firing Cache 51

Chapter 3 Customizing a Direct to Web Application 55

- Adding a Logo to Your Direct to Web Pages 55
- Using Direct to Web in Other WebObjects Applications 56
 - Embedding Direct to Web Components 56
 - Linking to a Direct to Web Page 57
- Modifying the Direct to Web Factory 60
- Creating a Custom Property-Level Component 61
 - Specifying the Custom Component 61
 - Using the Custom Component With Direct to Web 62
- Modifying a Direct to Web Template 63
 - Freezing Your Modified Direct to Web Template 64
- Adding a New Direct to Web Task to Your Application 64
 - Creating the Direct to Web Template 64
 - Adding Rules to Define the Default Behavior 66
 - Adding a Hyperlink to the New Task Page 69
- Adding Authentication to the Main Component 70
 - Hooking Up the Main Component 70
 - Add Logic 71
 - Add Better Logic 73

Appendix A EditStatePopup Listings 75

- EditStatePopup.html 75
- EditStatePopup.wod 75
- EditStatePopup.java 75

Figures, Tables, and Listings

Chapter 1 **An Introduction to Direct To Web 9**

- Figure 1-1 Header controls 17
- Listing 1-1 QueryMovieRole.java generated by the Web Assistant 36

Chapter 2 **Direct to Web Architecture 39**

- Figure 2-1 Edit page component organization 42
- Figure 2-2 Master-detail page component organization 43
- Figure 2-3 Direct to Web Architecture 45
- Table 2-1 Direct to Web tasks 39
- Table 2-2 Direct to Web Templates 40
- Table 2-3 Reusable components 41
- Table 2-4 Direct to Web templates and reusable components 41
- Table 2-5 Number property-level components (java.lang.Number, java.math.BigDecimal) 42
- Table 2-6 Initial Direct to Web context dictionary 46
- Table 2-7 Direct to Web context dictionary after setting propertyKey 48
- Table 2-8 Example of Cached Rule 52
- Table 2-9 Example of Cached Rule First Time it Fires 52
- Table 2-10 Cached Rule for Real Estate Agent Login 53
- Listing 2-1 Bindings file for a Direct to Web template 44
- Listing 2-2 D2WQueryAllEntitiesPage.showRegularQueryAction 46
- Listing 2-3 Rules used to resolve the pageName key 46
- Listing 2-4 BASQueryPage.html excerpts 47
- Listing 2-5 BASQueryPage.wod excerpts 48

Chapter 3 **Customizing a Direct to Web Application 55**

- Figure 3-1 Default Main component 70
- Figure 3-2 Add the login action 70
- Figure 3-3 Dragging from the action method to the Login button 71
- Listing 3-1 Sample code that sets up a next-page delegate 59
- Listing 3-2 PageWrapper.html example 60
- Listing 3-3 PageWrapper.wod example 60
- Listing 3-4 Customizing the D2W class 61
- Listing 3-5 Setting Rules With Rule Editor 62
- Listing 3-6 Implementation of the saveChanges method in NEUEditListPage.java 66
- Listing 3-7 Implementation of the editList method in NEUListPage2.java 69

Introduction to WebObjects Direct to Web Guide

Important: The following tools are deprecated and no longer supported in WebObjects 5.4 and later: EOModeler, RuleEditor, WebObjects Builder, WOALauncher, and Java Client. WebObjects templates are not available for creating new projects in Xcode on Mac OS X v10.5 and later.

Note: This document was previously titled *Developing WebObjects Applications With Direct to Web*.

The Direct to Web framework is a configurable system for creating WebObjects applications that access a database. All Direct to Web needs to create the application is a model for the database, which you can build using EOModeler.

This application is not a set of static web pages. Instead, Direct to Web uses information from the model at runtime to dynamically generate the pages. Consequently, you can modify your application's configuration at runtime using the Direct to Web Assistant (WebAssistant for short). You can hide entities, hide their properties, reorder the properties, and change the way they are displayed without recompiling or relaunching the application.

Organization of This Document

This book provides an introduction to Direct to Web and then describes the concepts you'll need to know when you customize a Direct to Web application.

To help you find what you are looking for, this book is divided into three remaining chapters:

[“An Introduction to Direct To Web”](#) (page 9) shows how to build a Direct to Web project and perform simple customization tasks on it.

[“Direct to Web Architecture”](#) (page 39) describes the components and classes involved in generating Direct to Web pages.

[“Customizing a Direct to Web Application”](#) (page 55) discusses several ways to customize the behavior of a Direct to Web application and presents three full examples:

- a property-level component that edits dates using pop-up lists
- a new task page called an edit-list page that edits several objects at a time.
- adding authentication to a Direct to Web application.

INTRODUCTION

Introduction to WebObjects Direct to Web Guide

The first topic is suitable for a WebObjects beginner. The last two topics are intended for a programmer who is already familiar with WebObjects, specifically entity-relationship modeling and the WebObjects request-response cycle. For more information about entity-relationship modeling, see *WebObjects Enterprise Objects Programming Guide*. For more information about the WebObjects request-response cycle, see *WebObjects Web Applications Programming Guide*.

An Introduction to Direct To Web

Direct to Web is a Java-based technology that provides a quick and easy method to create a web application that accesses a database. It lets you experiment and prototype, while also allowing you the flexibility to access the full power of WebObjects.

There are several stages you can go through, depending on your needs:

- First, you create a WebObjects project and specify a *model* to use. Direct to Web uses the model, which defines the mapping between your database and enterprise object classes, to generate an application that provides an interface to your database. This application consists of a set of pages that allow you to do queries on the entities in your database, display results, and add and delete records.

A complete model file that correctly defines all the relationships is the key to creating a WebObjects application with Direct to Web.

- To change the way the pages are presented, you can use the WebAssistant, which is a Java applet that runs in your web browser. For each page in your application, the WebAssistant allows you to specify which page is shown, which properties are shown on the page, how these properties are displayed, and the order in which they are listed. You can experiment with different configurations until you are satisfied, without writing any code.
- If you want to customize beyond what the WebAssistant provides, you can “freeze” any or all of the pages in your application as WebObjects components. This gives you the full power of WebObjects: you can modify a component’s layout using WebObjects Builder, and you can customize its behavior by writing Java code using Project Builder.
- You can also modify the templates that Direct to Web uses to generate its pages. These templates are WebObjects components that you can edit with WebObjects Builder. This way you can modify the “look” or style of the pages that Direct to Web generates.

You can also use Direct to Web in other types of WebObjects applications. Your application can take two approaches:

- embedding Direct to Web components in your pages; these include query forms, lists, or edit/inspect forms
- linking to dynamically generated Direct to Web pages

This chapter describes the elements that make up a Direct to Web application and shows you the steps you follow when creating and modifying an application. If you find that your application needs some specific behavior that Direct to Web does not provide, see [“Customizing a Direct to Web Application”](#) (page 55). For more information on using Project Builder, see Project Builder Help. For more information about using Project Builder and WebObjects Builder to develop WebObjects applications, see *Developing Web Applications*. For more information about using WebObjects with database applications, see *Enterprise Objects* as well as *Using EOModeler*.

Creating a Direct to Web Project

To create a Direct to Web application, begin by using Project Builder to create a WebObjects application project. Follow these steps:

1. Launch Project Builder.
2. Choose File > New Project.
3. Select WebObjects/Direct to Web Application.
4. Set the name and location of your project.
Enter `D2WTutorial` in the Project Name field.
Click Set and select a location for your project.
5. Click Next in the “Enable J2EE Integration” window.
6. Click Next in the “Enable Web Services” window.
7. In the “Choose EOAdaptors” window, make sure `JavaJDBCAdaptor` is selected.
Click Next.
8. Add additional frameworks to the project.
No additional frameworks are required for this tutorial, so click Next.
9. Choose a model file.
You’ll use one of the models defined in the sample projects.
Click Add.
Navigate to the `/Developer/Examples/JavaWebObjects/Frameworks/JavaRealEstate` directory.
Select `RealEstate.eomodel` and click Choose.
Click Next.
10. Select a look for the user-interface.
This pane offers a selection of user-interface styles (“looks”) for your Direct to Web application; see [“The Different Looks for WebObjects Applications”](#) (page 11) for more information.
Select Basic Look from the Direct To Web Looks list and click Next.
11. Build and launch the project.
This pane asks if you would like to build and launch your application immediately. If you choose not to have the wizard build and launch your application, see [“Using Your Direct to Web Application”](#) (page 14), which tells you how to launch your WebObjects application and describes what you see when you launch it. For information about building a project, see *Developing Web Applications*.
For the purposes of this tutorial, make sure “Build and launch project now” is selected and click Finish.

The Different Looks for WebObjects Applications

In this release, Direct to Web offers three different user-interface styles, or looks, for WebObjects applications: Basic, Neutral, and WebObjects. Currently the only simple way to change the look of an application is to re-create a project using Project Builder. Therefore it is advisable to know which look you want in advance.

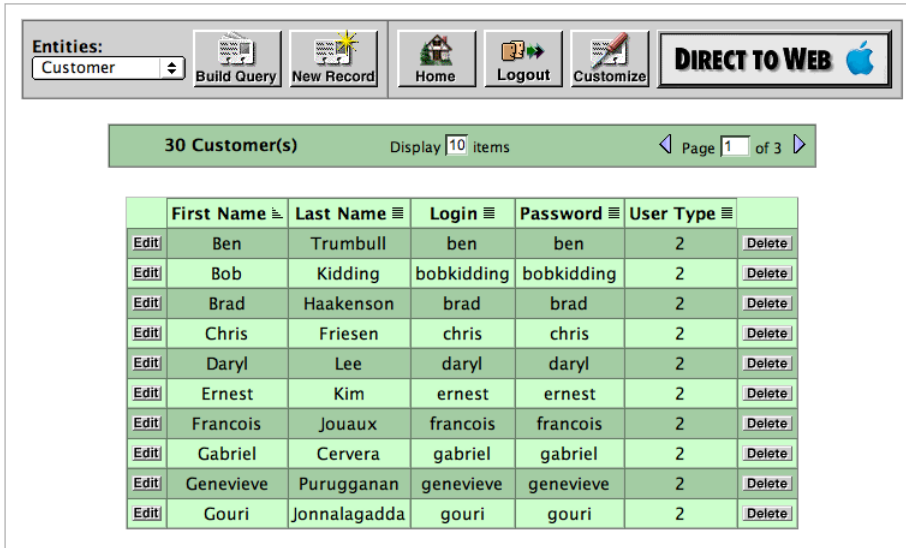
All the looks provide the same functions for the user. They only differ in the style and placement of their user-interface elements. The Neutral look and the WebObjects look are very similar but the Neutral look does not display the Apple logo, which makes it easier if you want to use your own logo.

The login page for the Basic look has a panel-like submit form for the entry of user name and password:

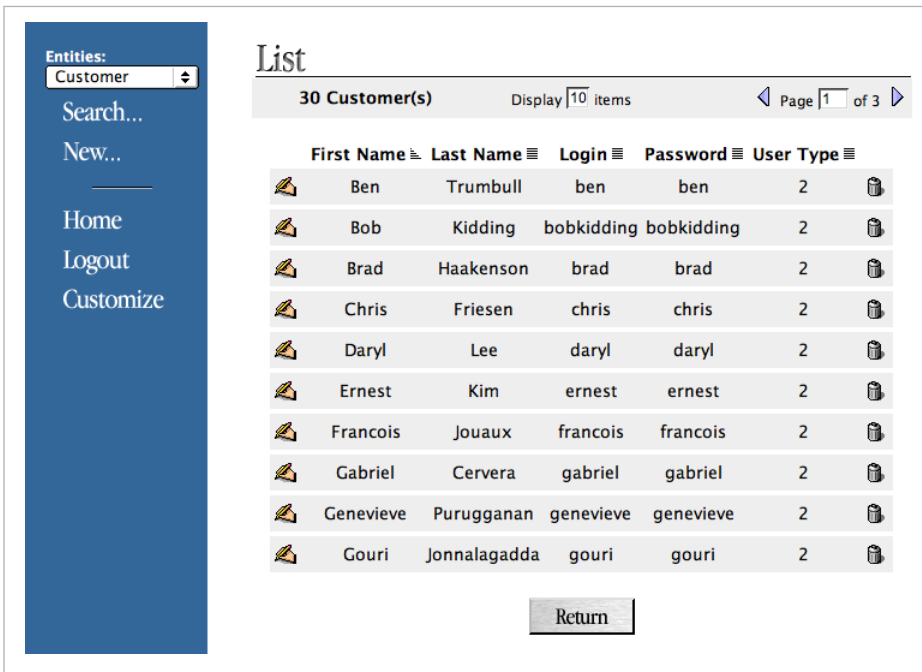
The login page for the Neutral look is much simpler:

This is the login page for the WebObjects look:

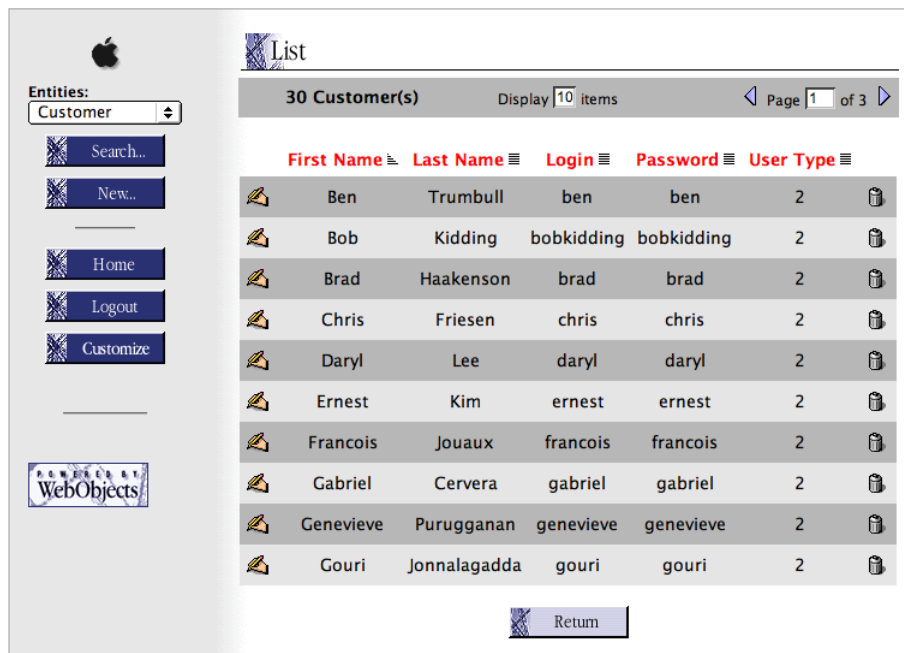
In the dynamically-generated pages (query, list, inspect, and so on), the Basic look differs from the Neutral and WebObjects looks even more strikingly. In the Basic look the control header runs across the top of the page whereas in the Neutral and WebObjects looks it appears on the left side of the page. In addition, the Basic look is more tabular while the Neutral and WebObjects looks tends to present records in visual “blocks.” For example, the following is an example of a list page in the Basic look:



The list page in the Neutral look appears like this:



The following illustrates what a list page looks like in the WebObjects look:



The Structure of a Direct to Web Project

A Direct to Web project has a structure similar to other WebObjects application projects. A newly created project contains three components, each enclosed in a subgroup, which you can access if you disclose the contents of the Web Components group in the Groups & Files list in Project Builder's main window.

- `MenuHeader.wo` is a reusable component that contains the header with the control buttons on the left side of each page (or the top of the page in the Basic look.) You can add text or other elements to this component if you choose.
- `Main.wo` is the main component, representing the login page of the application.
- `PageWrapper.wo` is a reusable component that wraps the content of the pages of the application (except for `Main.wo`). It contains a header, the menu header component (`MenuHeader.wo`), and footer text and elements common to these pages. If you want to customize the headers and footers for all pages of your application, you can add text or other elements to this component.


You can add code to the `.java` files corresponding to each component to extend their functionality. See the WebObjects API Reference for more information on the Direct to Web API.

Each of the subgroups that contains a component also holds the component's `.api` file. This file specifies the exported keys, both optional and required, for each component.

As you run your application, Direct to Web creates additional pages, using information in your model file and the settings specified in the WebAssistant. These pages do not show up as components in your project. Rather, Direct to Web creates them dynamically using a set of reusable components in the Direct to Web framework. However, you can generate components or user templates. When you do that, you can modify the resulting components just as you would with any other WebObjects components. See [“Generating Components”](#) (page 34) and [“User Templates”](#) (page 37) for more information.

The Resources group contains the model file you specified when you created the project (in this example, `RealEstate.eomodel`). It also contains `user.d2wmodel`, which stores the preferences you have specified using the WebAssistant. Advanced users can edit this file; see “The Rule System” (page 49) for more information about the rule file.

Using Your Direct to Web Application

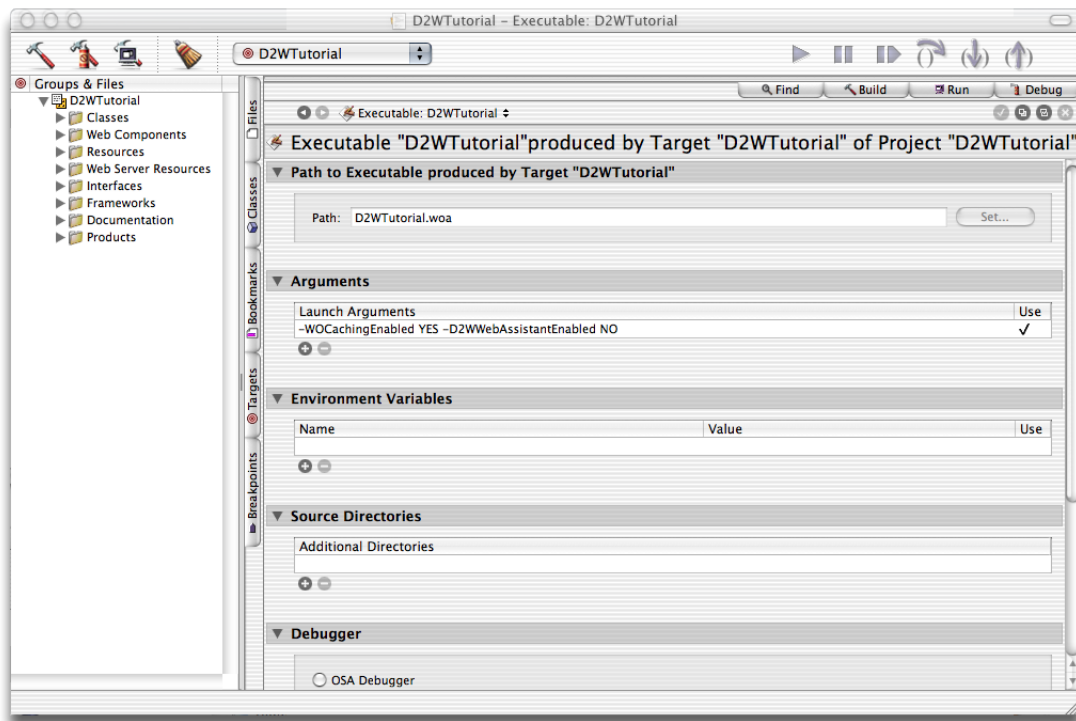
Once you have created a Direct to Web application using Project Builder and the WebObjects application wizard, and have compiled the resulting project files, you can launch the application by clicking Build and Run in Project Builder’s toolbar . The application pages are displayed in a web browser, where you can test the application’s presentation of data and, with the WebAssistant enabled, modify the layout of that data.

Launching a Direct to Web Application

To launch your application from Project Builder, click  in the toolbar in Project Builder’s main window.

Before you launch the application you might want to set some command line options. For example, when running a Direct to Web Application for deployment, you should turn on caching and disable the WebAssistant (to prevent anyone from connecting to the application using WebAssistant). To do this, set the `-WOCachingEnabled` and `-D2WebAssistantEnabled` options, respectively:

1. Under the Project menu choose “Edit Active Executable”
2. Under the Launch Arguments listing in the content pane, click the “plus” icon.
3. Enter `-WOCachingEnabled YES -D2WebAssistantEnabled NO` in the new entry that was created.



For other command-line options for WebObjects applications, such as `-W0Port`, see *Deploying WebObjects Applications*.

You can test the Direct to Web application using a web browser on a machine remote from the machine on which the application is running (that is, the server). When you launch the application, look in the console output, which is displayed in the Launch panel, for the line containing application's URL.

```
Welcometo D2WTutorial!
Opening application's URLin
browser:http://localhost:1234/cgi-bin/WebObjects/D2WTutorial.woa
```

Enter the URL in your browser, after substituting the host name of the server machine for "localhost". In fact, you can exclude every thing in the URL after the application port number. For example, if the server host name is "foobar" you would enter the following URL in the browser to load the WebObjects application:

```
http://foobar:1234/
```

The Login Page

When you launch your application, your web browser displays the Direct to Web login screen:



The login page is the default implementation of your Main component, `Main.wc`. It contains text fields to enter a name and password, as well as a submit button (Login) and an Enable Assistant checkbox. To go to the application's default first page, select Enable Assistant and click Login button. You don't need to enter a name and password, because the default application provides no password-checking logic. If you don't select Enable Assistant before clicking Login, you won't have access to the WebAssistant.

You can modify the login page (`Main.wc`) to provide any behavior or appearance you like. For example, you can add your own password-checking logic, as shown in the section [“Adding Authentication to the Main Component”](#) (page 70).

Dynamically Generated Pages

Besides the login page, there are nine types of dynamically-generated pages (or reusable components) in a Direct to Web application:

- **A query-all page** that displays all entities that are currently exposed and lets users construct queries on the attributes (but not the relationships) of those entities; see [“Query Pages”](#) (page 17). The properties of this page cannot be customized.
- **A query page** that allows the user to construct a query for a particular entity; see [“Query Pages”](#) (page 17).
- **A list page** that displays one or more records of a particular entity in tabular form. List pages and select components are implemented with the same components; see [“List Pages and Select Components”](#) (page 19). The result of a query is always a list page.
- **An inspect page** that displays a single record of a given entity. Inspect pages and edit pages are implemented with the same components; see [“Inspect and Edit Pages”](#) (page 20).
- **An edit page** that displays a single record of a given entity and also allows you to make changes to the record and save it to the database. Edit and inspect pages are implemented with the same components; see [“Inspect and Edit Pages”](#) (page 20).
- **A select component** that lets users select a record from a list, thereby adding it to a relationship or populating an edit component with it. List pages and select components are implemented with the same components; see [“List Pages and Select Components”](#) (page 19).
- **A confirm page** that prompts users to confirm that they want to delete records. The properties of this page cannot be customized.
- **An edit-relationship page** is a multiple component page for removing and adding objects to a relationship. See [“Edit-Relationship Pages”](#) (page 22).
- **An error page** for displaying information related to exceptions and other errors. The properties of this page cannot be customized.

All pages in your application contain the standard Direct to Web header (defined in `MenuHeader.wo`) at the top of the page. This header provides a number of controls, shown in [Figure 1-1](#) (page 17).

Figure 1-1 Header controls



For best results when navigating through a Direct to Web application, don't use your web browser's backtrack buttons. Instead:

- To return to the previous page from an edit or inspect page, click Cancel.
- To return to a query page from a list page, click Return.


Query Pages

Direct To Web has two kinds of pages for constructing queries on the properties of entities: a query-all page and a query page. When you log into a Direct To Web application, the query-all page is displayed first by default.

Find...	
Administrator	where <input type="text" value="lastName"/> is <input type="text" value="="/> <input type="text" value=""/> more..
Agent	where <input type="text" value="lastName"/> is <input type="text" value="="/> <input type="text" value=""/> more..
AgentPhoto	where <input type="text" value="photo"/> is <input type="text" value="="/> <input type="text" value=""/> more..
AgentRating	where <input type="text" value="comment"/> is <input type="text" value="="/> <input type="text" value=""/> more..
ContactInfo	where <input type="text" value="notes"/> is <input type="text" value="="/> <input type="text" value=""/> more..
ContactType	where <input type="text" value="type"/> is <input type="text" value="="/> <input type="text" value=""/> more..
Customer	where <input type="text" value="lastName"/> is <input type="text" value="="/> <input type="text" value=""/> more..
Feature	where <input type="text" value="feature"/> is <input type="text" value="="/> <input type="text" value=""/> more..
Listing	where <input type="text" value="virtualTourURL"/> is <input type="text" value="="/> <input type="text" value=""/> more..
ListingAddress	where <input type="text" value="aptNum"/> is <input type="text" value="="/> <input type="text" value=""/> more..
ListingPhoto	where <input type="text" value="photo"/> is <input type="text" value="="/> <input type="text" value=""/> more..
ListingType	where <input type="text" value="listingType"/> is <input type="text" value="="/> <input type="text" value=""/> more..
Rating	where <input type="text" value="ratingDescription"/> is <input type="text" value="="/> <input type="text" value=""/> more..
User	where <input type="text" value="lastName"/> is <input type="text" value="="/> <input type="text" value=""/> more..
UserDefaults	where <input type="text" value="userDefaults"/> is <input type="text" value="="/> <input type="text" value=""/> more..

The query-all page enables you to construct a query on an attribute of a particular entity (queries on relationships are not allowed). To use this page, select a property from an entity's pop-up list, specify the comparison operator, type the string to search on and click the magnifying-glass button.

The query page, on the other hand, is tied to a particular entity but allows you to construct queries on relationships as well as attributes. The following illustrates a query page:

Customer Query	
Agent	where Last Name is <input type="text"/>
Contact Info	where Notes is <input type="text"/>
First Name	starts with <input type="text"/>
Last Name	starts with <input type="text"/>
Login	starts with <input type="text"/>
Password	starts with <input type="text"/>
Suggested Listings	where Virtual Tour URL is <input type="text"/>
User Type	<input type="text"/> to <input type="text"/> (Number)
Agent	where Last Name is <input type="text"/>
Contact Info	where Notes is <input type="text"/>
	

The first column in the table lists the current entity's properties. The second column contains pop-up lists and text fields that let you enter values to construct queries on single and multiple properties. When you specify values for multiple properties, the query becomes the logical AND of the queries on the individual properties.

A property is either an *attribute* (a value stored directly in this entity's table) or a *relationship* (an association between this entity and another entity). For example, in the figure above, First Name is an attribute and Contact Info is a relationship. You can use the WebAssistant to hide properties that you don't want users to see.

Note: Direct to Web only displays properties that are class properties. In addition, primary keys and attributes marked as the source of a relationship are hidden by default.

Properties are represented in various ways. For example, in the figure, you enter a single string value for First Name, while you enter a range of values for User Type. You can change the representation of most properties using the WebAssistant. In particular, you may want to change how relationships are shown, since by default, you query them by specifying an ID, which is something the user is unlikely to know. See "[Changing How Properties Are Displayed](#)" (page 29) for more information on the different ways of representing properties in your application's pages.

You can choose a string operator (starts with, contains, ends with, is, like, =, <>, <, <=, >, >=) and specify a string with optional special characters in query fields for string searches. For example, you could select "starts with" in the Customer entity's Last Name pop-up list and enter "sh" in the text field to search for all customers whose last name begins with those characters. You can also use the "like" operator and enter a string with the asterisk character to indicate "all occurrences." For instance, you could enter "*rob*" to return all customers whose last name contains the substring "rob". Alternatively you could select "contains" in the pop-up list and enter "rob" to return the same customers.

In the Customer query, to get a list of all customers whose last name contains "id" and first name begins with "r", you would:

1. Select "starts with" in the First Name pop-up list and enter "r" in the text field.
2. Select "contains" in the Last Name pop-up list and enter "id" in the text field.
3. Click Query DB.

The results are displayed in a list page; see "[List Pages and Select Components](#)" (page 19).

To clear the query page, click Build Query.

List Pages and Select Components




A list page displays a table showing multiple records of an entity. List pages are used to display the results of a query, or to show the records satisfying a to-many relationship in another list or inspect page.

30 Customer(s)							
Display 5 items				Page 1 of 6			
	First Name	Last Name	Login	Password	User Type	Agent	Suggested Listings
Edit	Ben	Trumbull	ben	ben	2	Inspect	Inspect Delete
Edit	Bob	Kidding	bobkidding	bobkidding	2	Inspect	Inspect Delete
Edit	Brad	Haakenson	brad	brad	2	Inspect	Inspect Delete
Edit	Chris	Friesen	chris	chris	2	Inspect	Inspect Delete
Edit	Daryl	Lee	daryl	daryl	2	Inspect	Inspect Delete

Each row in the table represents a record. By default, a batch of ten records are shown in a page. To change the batch size, type a number in the “Display _ items” field and press Return or Enter. To display additional records in either direction, click the triangle buttons or enter the page number you want to go to.

Each column in the list represents one of the entity’s properties. By default, all properties are shown in alphabetical order. You can hide columns and change their order by using the WebAssistant; see [“Customizing Your Application With the WebAssistant”](#) (page 24).

The symbols to the right of attribute names represent their sort order:

-  : ascending order
-  : descending order
-  : unsorted

To change the sort order for any attribute, click the title to cycle between ascending, descending, and unsorted. By default, the records are sorted in ascending order by the attribute in the first column. You can specify up to three columns to sort on; the last one specified becomes the primary sort key.

For properties that represent relationships, an Inspect button appears in the cell by default (DisplayToManyFault).

Note: By default, the list page does not display relationships (including the Inspect buttons). You can configure the list page to display relationships using the WebAssistant; see [“Customizing Your Application With the WebAssistant”](#) (page 24).

When you click the Inspect button one of two things happen, depending on the type of relationship:


- If it is a to-one relationship, an inspect page appears, showing the destination record.

In the above example, the Customer entity's Agent relationship is a to-one relationship to the Agent entity. If you click the Inspect button, an inspect page appears for the Agent entity corresponding to the selected customer; see "Inspect and Edit Pages" (page 20).

- If it is a to-many relationship, another list page appears, showing all the destination records in the relationship.

In the above example, the Customer entity's Suggested Listings relationship is a to-many relationship to the Listing entity. If you click the Inspect button, a list page appears, showing all the suggested listings in the selected customer.

3 Listing(s)							
	Asking Price	Bathrooms	Bedrooms	Lot Sq Ft	Size Sq Ft	Year Built	
Edit	\$300,000.00	2	3	6000	1400	1962	Delete
Edit	\$350,000.00	2	3	5000	1400	1989	Delete
Edit	\$700,000.00	4	4	9000	3000	1992	Delete



You can use the WebAssistant to display the related records directly in the table instead of with an Inspect button; see "Customizing Your Application With the WebAssistant" (page 24).

The select component looks a lot like the list page, but instead of the Edit button there is a Select or Add button. The select component occurs in multiple-component pages. In the edit-relationship page you click Select or Add to add a record to a to-many relationship or select a record for a to-one relationship. In the master-detail page you click Select to select a record to edit. A select component looks like this:

5 Listing(s)						
	Asking Price	Bathrooms	Bedrooms	Lot Sq Ft	Size Sq Ft	Year Built
Add	\$300,000.00	2	3	6000	1400	1962
Add	\$350,000.00	2	3	5000	1400	1989
Add	\$355,000.00	2	3	6800	1300	1983
Add	\$375,000.00	2	3	6200	1500	1975
Add	\$550,000.00	3	3	5000	1500	1972

Inspect and Edit Pages

Inspect pages and edit pages display the data for a single record of an entity. An edit page allows you to make changes to the record and save the changes, while an inspect page is read-only.

An inspect page looks like this

Agent	
Contact Info	▸ 2 ContactInfos
Customers	▸ 9 Customers
First Name	Todd
Last Name	Fernandez
Listings	▸ 4 Listings
Ratings	▸ 0 AgentRatings
User Type	1
<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> Delete</div> <div style="text-align: center;"> Return</div> <div style="text-align: center;"> Edit</div> </div>	

Note the buttons at the bottom of the page:

- Delete allows you to delete the record from the database.
- Return takes you back to the page from which you accessed this inspect page.
- Edit brings up the equivalent edit page for this record, so that you can make changes. (However, if your application specifies a particular entity as read-only, you won't be able to edit it.)

Also note the Customers property in the example above. You click the triangle to display the Customers of this Agent in a list, browser, or table, as in the following example:

Agent	
Contact Info	▸ 2 ContactInfos
Customers	<div style="border: 1px solid black; padding: 2px;"> <div style="text-align: right; padding-right: 5px;">▼</div> <div style="padding: 2px;">2, ray, ray, Kiddy, Ray</div> <div style="padding: 2px;">2, steveC, steveC, Cavin, Steve</div> <div style="padding: 2px;">2, mankit, mankit, Sze, Mankit</div> <div style="padding: 2px;">2, bobkidding, bobkidding, Kidding, Bob</div> <div style="padding: 2px;">2, hanming, hanming, Ong, Han Ming</div> <div style="padding: 2px;">All... (9 items)</div> </div>
First Name	Todd
Last Name	Fernandez
Listings	▸ 4 Listings
Ratings	▸ 0 AgentRatings
User Type	1
<div style="display: flex; justify-content: space-around;"> <div style="text-align: center;"> Delete</div> <div style="text-align: center;"> Return</div> <div style="text-align: center;"> Edit</div> </div>	

This property is configured with the DisplayToManyTable component. For more on how this is done, see “Representation of Relationships” (page 30).

An edit page (or edit component) looks like this:

Agent	
Contact Info	todd@todd.com 650 865 2431, 9 AM - 7 PM Edit
Customers	2, ray, ray, Kiddy, Ray 2, steveC, steveC, Cavin, Steve 2, mankit, mankit, Sze, Mankit 2, bobkidding, bobkidding, Kidding, Bob Edit
First Name	Todd
Last Name	Fernandez
Listings	LN #123456, \$300,000.00, 1, 2, 5000, 800, 1943 LN #777787, \$300,000.00, 2, 3, 6000, 1400, 1962 LN #642871, true, \$355,000.00, \$362,000.00, 2, 3, 6800, 1300 LN #479524, false, \$400,000.00, 2, 4, 8000, 1900, 1982 Edit
Ratings	Edit
User Type	1
<div style="display: flex; justify-content: space-around;"> Delete Cancel Save </div>	

It is similar to the inspect page, except that it has a Save button (for saving changes to the database) instead of an Edit button. If you click the Edit button next to the list of Customers, an edit-relationship page is displayed for editing the records in the to-many relationship. Edit components can occur in multiple-component pages, such as the master-detail page.

Edit-Relationship Pages

An edit-relationship page allows users to add records to a relationship and remove records from the relationship. Users typically come to these pages when they click an Edit button next to a relationship in an edit page. Edit-relationship pages consist of three separate components, of which two are shown at any one time. The first component lists the relationships of a particular property and contains several controls. In addition, a query component initially appears for locating another object to link to for that property. The third component, a select component, appears after you have specified a query and is discussed in the following pages.

Current Customers(s)

9 Customer(s)	2, ray, ray, Kiddy, Ray 2, steveC, steveC, Cavin, Steve 2, mankit, mankit, Sze, Mankit 2, bobkidding, bobkidding, Kidding, Bob	<input type="button" value="Remove"/>
---------------	---	---------------------------------------

Customer Query

Agent	where Last Name is <input style="width: 80%;" type="text"/>
Contact Info	where Notes is <input style="width: 80%;" type="text"/>
Defaults	where User Defaults is <input style="width: 80%;" type="text"/>
First Name	starts with <input style="width: 80%;" type="text"/>
Last Name	starts with <input style="width: 80%;" type="text"/>
Login	starts with <input style="width: 80%;" type="text"/>
Password	starts with <input style="width: 80%;" type="text"/>
Suggested Listings	where Virtual Tour URL is <input style="width: 80%;" type="text"/>
User Type	<input style="width: 20%;" type="text"/> to <input style="width: 20%;" type="text"/> (Number)

This user interface facilitates the following tasks:

- To remove a record from the property, select the key identifying the record in the browser and click Remove.
- To add a new record to the property, click New Record. An edit component appears underneath the list of relationships; fill out the fields of the edit component and click Save to add the new record to the database *and* the new relationship to the property above.
- To locate an existing record to add to the relationship, enter the properties to search on in the query component and click Query DB.

When a query is executed (assuming matching records are found) a select component replaces the query component.

Current Customers(s)

9 Customer(s)	2, ray, ray, Kiddy, Ray 2, steveC, steveC, Cavin, Steve 2, mankit, mankit, Sze, Mankit 2, bobkidding, bobkidding, Kidding, Bob	<input type="button" value="Remove"/>
---------------	---	---------------------------------------

1 Customer(s)

	First Name	Last Name	Login	Password	User Type
<input type="button" value="Add"/>	Karl	Hsu	karl	karl	2

To add a listed record to the to-many relationship, click the Select button. To construct a new query, click the Build Query button.

When you have finished editing a relationship, click the Return button under the browser to return to the original edit page. You must click the Save button in this page to store the changed relationship in the database.

Master-Detail Pages

Master-detail pages put a select component and an edit component on the same page, thereby allowing users to select and edit records without having to go to another page. The following is an example of a master-detail page:

3 Customer(s)

	First Name	Last Name	Login	Password	User Type
Select	Bob	Kidding	bobkidding	bobkidding	2
Select	Ernest	Kim	ernest	ernest	2
Select	Ray	Kiddy	ray	ray	2

Return

Customer

Agent: [1, todd, todd, Fernandez, Todd](#) Edit

Contact Info: Edit

First Name:

Last Name:

Suggested Listings:

LN #123456, \$300,000.00, 1, 2, 5000, 800, 1943

LN #642871, true, \$355,000.00, \$362,000.00, 2, 3, 6800, 1300

LN #796380, \$700,000.00, 4, 4, 9000, 3000, 1992

Edit

User Type:

Delete Cancel Save

To use a master-detail page, click Select next to a record in the list component. The record is displayed in an edit component. See [“Inspect and Edit Pages”](#) (page 20) for usage information.

The master-detail page does not appear under Tasks in the WebAssistant (expert mode). This is because it is defined as a type of list page (BASMasterDetailPage, NEUMasterDetailPage, or WOLMasterDetail page depending on the look) of the list task.

Customizing Your Application With the WebAssistant

The WebAssistant allows you to customize each page of your application. You can specify

- which entities of the model the application displays and, of these, which are read-only
- global attributes of pages, such as style, color, and border thickness
- which properties are displayed, and in what order

By default, an entity's properties are listed in alphabetical order. Often, you'll want to change the order, as well as hiding some properties.

- how number and date strings should be represented
- how relationships should be represented

Running the WebAssistant With appletviewer

You can launch WebAssistant using the Java program `appletviewer`. You might need to do this if your web browser is unable to launch WebAssistant itself. Follow these steps:

1. Launch your application with the command-line option `D2WebAssistantEnabled` set to YES. This is covered in ["Launching a Direct to Web Application"](#) (page 14).
2. In the "Run" output of Project Builder, look for a line similar to the following:

```
DirectToWeb WebAssistant launch line:appletviewer
http://localhost:51508/cgi-bin/WebObjects/D2WTutorial.woa/wa/D2WActions/openWebAssistant
```

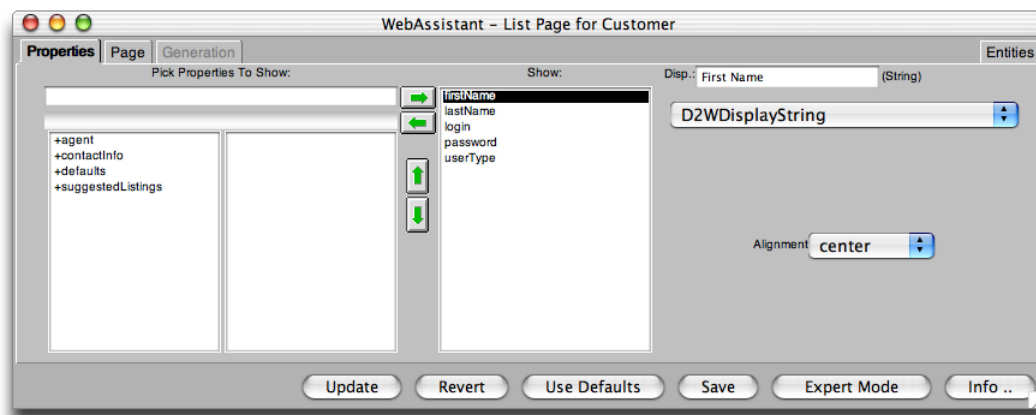
3. Open the Terminal application provided on Mac OS X.
4. Copy the string beginning at "appletviewer" up to "openWebAssistant" to the Terminal and press Return.

When you complete this procedure, WebAssistant launches and connects to your application. If you stop and restart the Direct to Web application, the WebAssistant will re-connect to it—provided it is running on the same port.

A standalone WebAssistant has exactly the same functionality as one launched inside your browser. However, if the browser you are using is not Java-enabled, your pages are not automatically refreshed after you click Update. You must either click your browser's "reload" or "refresh" button or (when you are picking a new type of page, such as a MasterDetails page instead of a ListPage), you will have to re-navigate to the same page.

WebAssistant Overview

When the Web Assistant applet is launched, it appears in a window whose title indicates the current page and entity:



The WebAssistant has four displays, each selectable by clicking a tab:

- **Properties** Allows you to set which properties of an entity are shown in a page, the order in which they're displayed, and the display characteristics of properties (for example, color and alignment).
- **Page** Allows you to customize global page properties, such as template, overall style, color, and border thickness.
- **Generation** Only available in expert mode, this display allows you to generate templates and “freeze” customized pages as reusable components.
- **Entities** Allows you to select which entities of the model are hidden, which are shown, and which are read-only.

The WebAssistant stays synchronous with your browser. When you navigate to a new page, it displays the settings for that page.

The Web Assistant has two modes, Standard mode and Expert mode. By default the Web Assistant opens in Standard mode, which lets you customize the current page in your application. When you customize a page in Standard mode, the changes apply to all occurrences of that page, and that page only. For example, if you change the order of properties in an edit page for the Customer entity, then any time a Customer edit page is displayed, those changes are in effect (provided you have clicked Update or Save). However, the changes don't apply to a Customer query, list, or inspect page; if you want to customize those in the same way, you must do so explicitly.

Using Web Assistant's Expert mode, you can customize any page in the application, regardless of whether it is currently displayed. Thus, by specifying the “*all*” setting in Expert mode, you could change all pages of a given entity at once. In addition, you can generate a template or “freeze” a page as a reusable component. For more information, see [“WebAssistant Expert Mode”](#) (page 32).

When you've made changes to a page, you can use the buttons at the bottom of the WebAssistant to apply them:

- **Update** sends your changes to the server. On some systems this causes the page to be refreshed in your browser.
- **Revert** causes all settings to revert to their last saved values.
- **Use Defaults** reverts all settings to the values they had when the project was created.

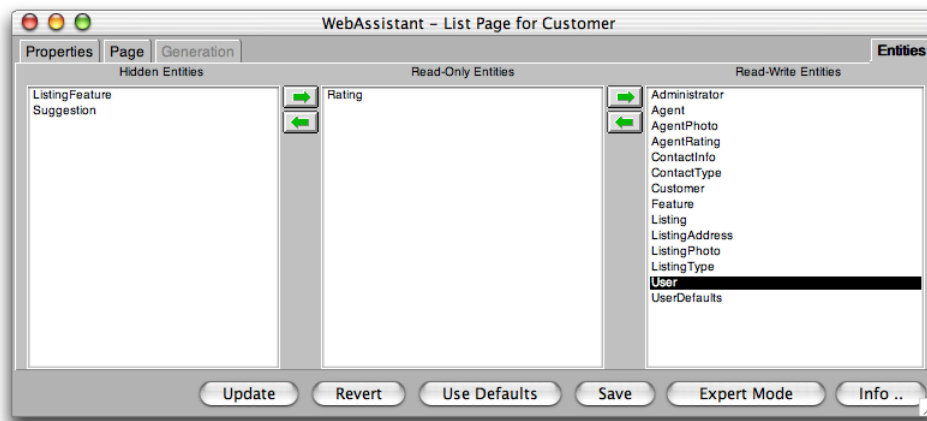
- **Save** saves the changes to disk. You need to save your changes in order for them to persist beyond the current session.

The “Info...” button displays a brief description of the currently selected Direct to Web component.

Restricting Access to Entities

The Entities display of the WebAssistant enables you to specify which entities of the database model appear in the application. Of those entities, it further allows you to specify which are read-only and which the user can write data to. Records from read-only entities are restricted from appearing in edit pages.

The user interface for accomplishing these tasks is simple, as the following example illustrates:

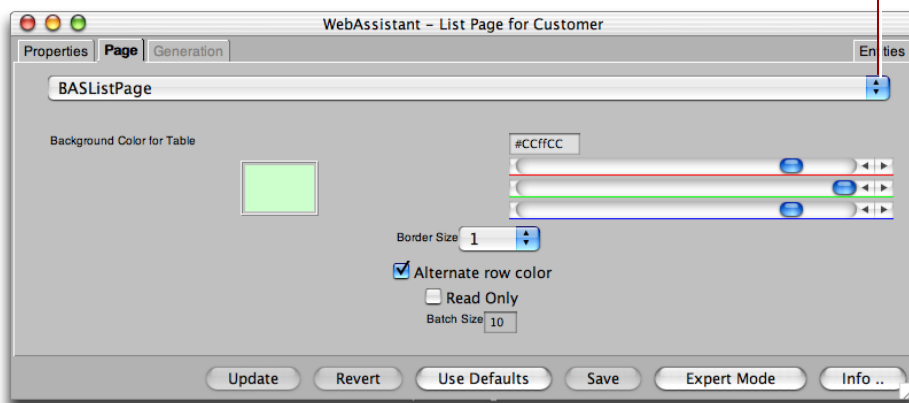


To specify an entity that shouldn't appear in the application, select it and use the arrow keys to move it to the Hidden Entities column. The above example shows that the entities ListingFeature and Suggestions are hidden. To specify an entity that should be read-only, select it and use the arrow keys to move it to the Read-only Entities column. The above example shows that the Rating entity has been moved the the Read-Only column. By default, all entities initially appear in the Read/Write Entities column.

Customizing Pages

The Customize Page display of the WebAssistant enables you to set global attributes for the current page. These attributes include the page template, the color of the table, whether this color alternates with another color in lists, and the size of the border enclosing the page. The following is an example of the Customize Page display:

Click here to choose the template for this page.



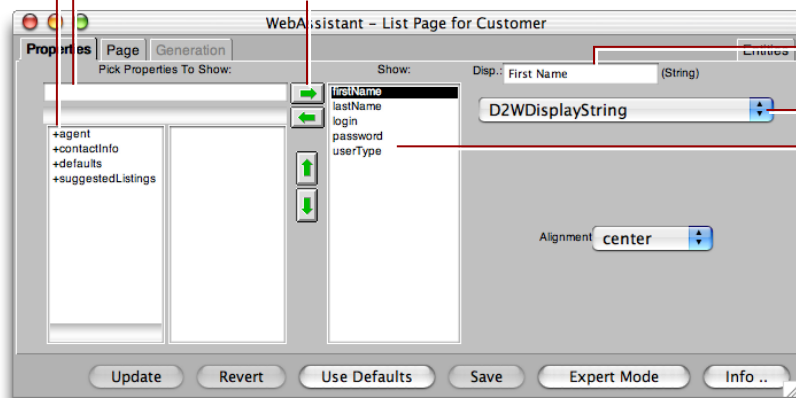
- To change the template defining the page style, choose another template from the pop-up list.
- To change the thickness of the border around the page, choose a value from the Border Size pop-up list, replacing the current number. You can specify a border thickness of 0 to 5 pixels.
- To change the color of the table, move the sliders to the right of the sample color. The color specification is RGB-based (that is, a specific mixture of red, green, and blue). The top slider manipulates red saturation, the middle slider is for green, and the bottom slider is for blue. The three pairs of hexadecimal digits after the number sign in the field represent (left to right) saturation levels of red, green, and blue.

Setting Which Properties are Displayed

The Properties display of the WebAssistant enables you to specify which properties of an entity appear in a page (or component) and the order in which these properties appear. Most of the user-interface elements for accomplishing these things are in the left half of the display as shown in the following example:

Choose a property to show with the key path browser...
or type it into this field.

Use the arrow buttons to show or hide properties, or to change their order.



Enter a label for the property here.

Component pop-up list.

These properties appear on the page

The entity's properties (attributes and relationships) in the Show column are displayed in the page. To the left of the arrows is a key browser that shows relationships (which appear with a "+") and hidden attributes. You can click on a relationship to show its attributes and relationships in the next column of the key browser.

The WebAssistant displays the keys that can be found in the entity's source code. If you want to show a key or key path that doesn't appear, you can type in the text field.

For each property, you can

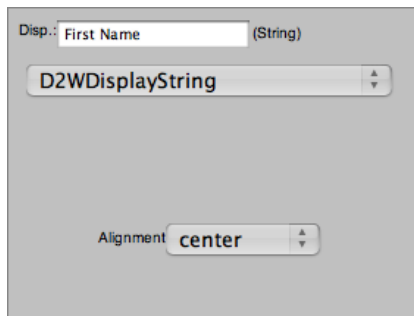
- Move it to the key browser by selecting it and clicking the left arrow. This hides the property. Similarly, if a property is hidden, you can show it by selecting it and clicking the right arrow.
- Move it up or down in the list by clicking the up and down arrows. This changes the order of appearance of the properties in the page (left to right or top to bottom, depending on the component).

By default, the WebAssistant shows only class properties. If you want to show a custom method or a keypath, type it into the textfield under the "Pick Properties To Show" label (for example, "agent.photo").

You can also change the title for a property by editing the string in the Display (Disp.) field. This change only affects the way the property is labeled in the page and has no effect on the actual property name.

Changing How Properties Are Displayed

You can use the Properties display of the WebAssistant to specify various display characteristics of properties, such as formatting, color, alignment, and the representation of relationships. The fields and controls for setting these characteristics are on the right half of the display. Here is an example:



Let's go over the various elements of this part of the user interface:

- At the top is the Display field, which holds the title of the property for the current page and entity. As discussed in "[Setting Which Properties are Displayed](#)" (page 28), you can edit this string.
- Next to the Display field, in parentheses, is its data type. The data type determines the set of display components available for use. You cannot edit this information directly (however, you can edit the model file, which specifies the data type, using EOModeler).
- The pop-up list shows the name of the component that is used to display the selected property in the current page. From this menu you can choose a different component to display the property. When you choose a display component, the set of controls and fields in the right side of the Properties display can change.

The items in the pop-up list identify reusable components in the Direct to Web framework called property-level components, which are used to render the properties on the pages you see in your application. Each property in a page of any type is initially shown in a default way for that type using a default property-level component.

Textual Attributes and Formatting

The display components available for the currently selected property offer characteristics suitable to the data type and function of the attribute. A few examples might help to clarify this statement:

- If the data type of the attribute is a String, but it is also a URL, then the DisplayHyperlink or DisplayMailTo components could be what you want.
- If the attribute is a date (NSTimestamp), then you might choose the DisplayString component and provide format specifiers to have the date formatted in a certain way.
- Similarly, if the attribute is a currency value (BigDecimal), you might want to use the DisplayNumber component and format the display of the attribute with two decimal positions and a leading dollar sign.
- If you want to highlight a certain column of values in a table by giving them a different color, then you could choose the DisplayStyledString component that lets you apply a color to a property.

You can click the Info button in the WebAssistant to get a short description of the currently selected display component.

The three most common display characteristics for properties are alignment, formatting, and color. Each of these has their own controls or fields in the right side of the Property display:

- **Alignment** Choose Right, Center, or Left from the pop-up list to specify the alignment of text within a cell of a table.
- **Formatter** You can have your application display some types of data, such as dates and numbers, as formatted strings. For example, the date “Sat 4 Jul 98” can be also represented as “July 4, 1998.” The number one thousand can be represented either as “1,000” or “1.000”, depending on the locale. There are different format specifiers for dates and numbers; check the reference documentation for the NSTimestampFormatter and NSNumberFormatter classes for details.
- **Color** To change the color of text, either move the sliders to the right of the sample color or enter hexadecimal numbers in the field above the sliders. The color specification is RGB-based (that is, a specific mixture of red, green, and blue). The top slider manipulates red saturation, the middle slider is for green, and the bottom slider is for blue. The three pairs of hexadecimal digits after the number sign in the field represent (left to right) saturation levels of red, green, and blue.

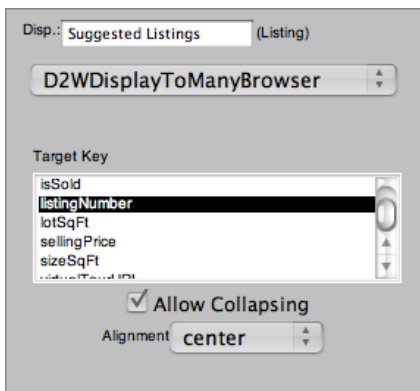
Representation of Relationships

Properties that are relationships (instead of attributes) have their own set of display components that you can use. Take the following list page as an example:

30 Customer(s)							
Display 5 items				Page 1 of 6			
	First Name	Last Name	Login	Password	User Type	Agent	Suggested Listings
Edit	Ben	Trumbull	ben	ben	2	Inspect	Inspect Delete
Edit	Bob	Kidding	bobkidding	bobkidding	2	Inspect	Inspect Delete
Edit	Brad	Haakenson	brad	brad	2	Inspect	Inspect Delete
Edit	Chris	Friesen	chris	chris	2	Inspect	Inspect Delete
Edit	Daryl	Lee	daryl	daryl	2	Inspect	Inspect Delete

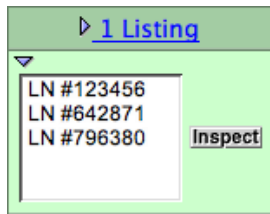
There are two relationships on this page. One is a to-one relationship (Agent) and the other is a to-many relationship (Suggested Listings). By default, all to-many relationships are displayed using `DisplayToManyFault`, and to-one relationships are displayed using `DisplayToOneFault`. “Fault” indicates that the records in the relationship aren’t displayed until they are asked for; that is, until the user clicks `Inspect`. When you click `Inspect`, a list page appears, showing all the records in the relationship (such as all suggested listings for the customer).

You can change the display component for the relationship to get a different presentation. Consider the Suggested Listings relationship in the Customer-List page example above. Using your browser, navigate to the list page for the Customer entity. Move the suggestedListings property to the Show column using the WebAssistant and choose `D2WDisplayToManyBrowser` from the component pop-up list. The right side of the WebAssistant should look similar to the following example:

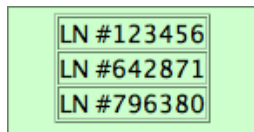


In addition to the Alignment pop-up list, the `WOComponent` group includes two controls specific to the display of relationships. The items in the Target Key browser are selected attributes of the destination entity; these “target keys” refer to a string identifying a to-many relationship. In this case the Suggested Listings entity has many target keys to choose from. In addition, Direct to Web provides a default key called `userPresentableDescription`, which is usually a combination of the relationship’s keys, if there are multiple keys.

The Allow Collapsing checkbox, when checked, causes the relationship initially to be presented as a disclosure triangle followed by a number and the plural form of the display name for the destination entity (for example, “3 Listings”). When the user clicks the triangle, the table cell expands to display the items in the form appropriate to the display component; in this case, a browser:



To get a better sense of the control you have over the presentation of relationships, set the display component for the Suggested Listings relationship to DisplayToMany and uncheck the Allow Collapsing checkbox. When you update your browser, a cell in the Suggested Listings column should look similar to this:



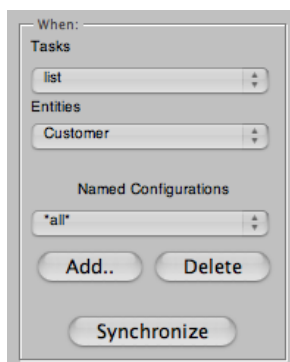
To-one relationships offer five possibilities for presentation. However, only three are relevant. The DisplayToOneFault component presents an Inspect button which, when clicked, displays the relationship record in an inspect page. The DisplayToOne component displays the target key for the single destination record as a hyperlink which, when clicked, brings you to the same inspect page. Finally, there is always the option of using a D2WCustomComponent. For more information on creating custom components, see “Creating a Custom Property-Level Component” (page 83).

A note of caution: The type of display component appropriate to a relationship depends on the likely number of records in that relationship. For example, the Agent entity has a Listings to-many relationship; if some agents have numerous listings, it might make more sense to use DisplayToManyFault (that is, the Inspect button) rather than display the listing numbers of all those listings in a cell in the table.

To find out more about a display component for a relationship, click the Info button in the WebAssistant after selecting the component.

WebAssistant Expert Mode

Expert mode is similar to standard mode, except that it allows you to make changes to any page in your application whether it is currently displayed in your browser or not. If you click the Expert mode button at the bottom of the WebAssistant, the window expands to include the following interface:

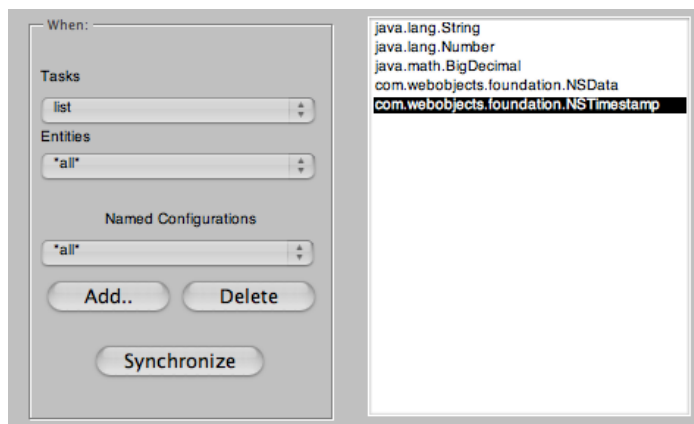


The tasks pop-up list shows the types of pages available in Direct to Web. The Entities pop-up list shows the entities in the model.

Choosing a Page to Customize

To customize any page in your application, simply select the type of page and the entity. The figure above shows an example of choosing the list page for the Customer entity, making the WebAssistant focus on this page rather than the page currently showing in the browser.

If you select `**all**` under Tasks, any changes you make affect all customizable pages for the selected entity. If you select `**all**` under Entities, you'll see a list of data types that exist in the application, as shown in the following figure.



Any changes you make affect all occurrences of that type. For example, the figure shows `NSTimestamp` selected. You can specify a formatter, and pick a component to use anywhere in the application that an `NSTimestamp` object is displayed.

If you click `Synchronize`, the task and entity for the current browser page are selected in the WebAssistant.

You can also select the Page pane in the WebAssistant window while in Expert mode and change the underlying component, color, and border thickness of whatever page for whatever entity you select in the Tasks and Entities pop-up menus.

Named Configurations

Once you have customized a page, you can capture the settings in a *named configuration*. Named configurations are used when you need to display more than one type of page for a particular task and entity. Consider a page that lists properties for a real estate agency. A potential buyer would want to see the things like amount of rooms and bathrooms along with the asking price. A real estate agent might want to see how many times the property has been looked at and what the selling price is. In addition, the buyer should not be able to edit any information, while the agent might be able to edit some properties. To set up such a system, you create two named configurations for listing properties: one for the potential buyer and another for the real estate agent.

Named configurations can only be displayed programmatically; the WebAssistant can edit named configurations but can't display the changes in your browser.

To create a new named configuration follow these steps:

1. In the expert mode interface, click Add.

A panel appears with a text field containing a default name for the configuration (the page name followed by the entity name).

2. Enter a new name for the configuration if you choose.
3. Click Ok.
4. Configure the Properties and Page panes for the named configuration.
5. Click Save.

To edit a named configuration, select it from the Named Configurations pop-up list, make the changes, and click Save.

To delete a named configuration, select it from the Named Configurations pop-up list and click Delete.

Generating Components

When you have worked with the WebAssistant and customized your pages to your liking, you may still want to add more features to your application. To do so, you can “freeze” a page; that is, save it as a WebObjects component. When you do this, the component becomes part of your project and is no longer created “on the fly” by Direct to Web. This has several advantages:

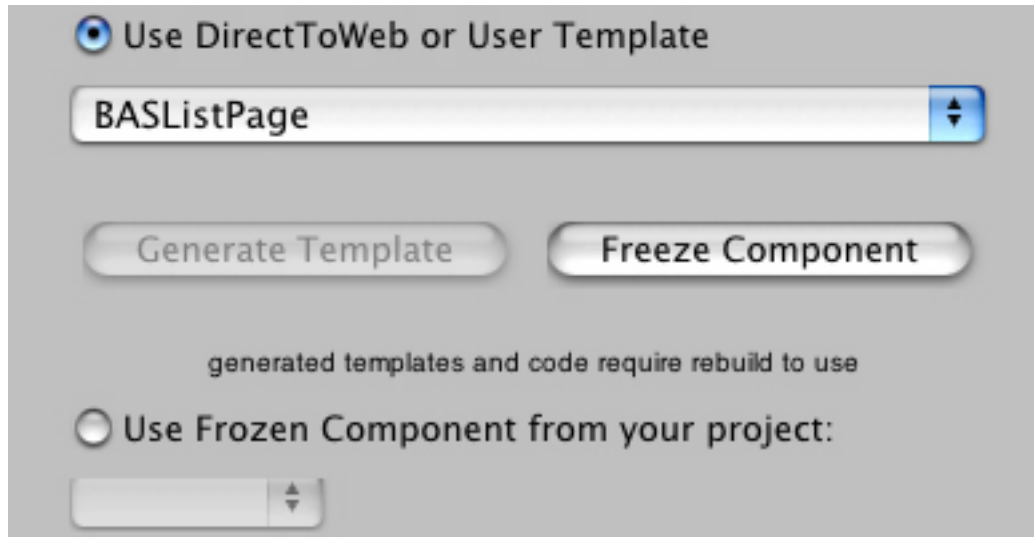
- You have complete control over the visual appearance of the page. You can add any static or dynamic HTML elements you like, using a tool such as WebObjects Builder.
- You can add functionality to the page by editing the component’s Java code, as well as by editing the bindings of the page’s dynamic elements.
- Your application’s performance improves because Direct to Web doesn’t have to go through the process of creating the page “on the fly.”

The main disadvantage of generating components is that you lose the ability to modify settings with the WebAssistant since the entity, property settings, and page configuration are stored directly in the generated component. To modify the page, you must edit the component or its corresponding `.java` file. Therefore, you should try to get your settings as close as possible to what you want before generating the component.

To generate a component:

1. Click the Expert Mode button at the bottom of the WebAssistant to enter Expert mode.

2. Click the Generation tab at the top of the WebAssistant.

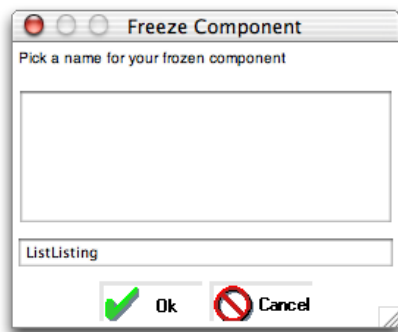


3. Select the task and entity corresponding to the page you want to generate.

You can't select "*"all*" to generate multiple components. You must generate the components one at a time.

4. In the Advanced Options group of controls, make sure the "Use DirectToWeb or User Template" radio button is selected.
5. Click Freeze Component.

The Freeze Component window appears. It contains a text field with a default name for your page (the page name followed by the entity name). You can edit the name if you choose.



6. Click the Ok button.

Direct to Web generates a component (with extension `.wo`) and a corresponding `.java` file and adds them to your project. You may have to wait a few moments for this process to complete. Your settings are automatically saved.

7. Rebuild and run your project, and restart the WebAssistant.

If you decide not to use the frozen component and have Direct to Web build the page “on the fly,” select the “Use DirectToWeb or User Template” option.

When you generate a page and click Update, the browser’s current page doesn’t reflect the changes. To use the new component, you must rebuild the application, relaunch it, and then navigate to a new instance of the page. For example, if the current page is a Movie query page, and you use the WebAssistant to freeze it, you must rebuild the project with the frozen component, then launch the application and navigate to a new instance of Movie query (by clicking Build Query); the new instance uses the frozen component.

The generated component is like any other WebObjects component. You can edit your component graphically using WebObjects Builder. You can also examine the HTML and bindings (.wod file) of the new component in Project Builder.

Direct to Web also generates Java code for your component, which you can modify appropriately to your needs. Each component implements an interface that is appropriate to the page: QueryPageInterface, ListPageInterface, InspectPageInterface, and EditPageInterface. For example, the QueryAgent.java file in Listing 1-1 (page 36) implements the QueryPageInterface. For example, it contains an action method called queryAction that returns a component when the Query DB button is clicked. (Note that the component’s submit button is bound to queryAction in QueryAgent.wod.)

Listing 1-1 QueryMovieRole.java generated by the Web Assistant

```
import com.webobjects.appserver.*;
import com.webobjects.eocontrol.*;
import com.webobjects.directtoweb.*;
import com.webobjects.eoaccess.*;
import java.util.*;

public class QueryAgent extends WOComponent implements QueryPageInterface {

    protected EODatabaseDataSource_queryDataSource;

    protected WODisplayGroupdisplayGroup;

    protected NextPageDelegate_nextPageDelegate;

    public WOComponent queryAction(){
        _queryDataSource =new
EODatabaseDataSource(session().defaultEditingContext(), "Agent");
        _queryDataSource.setAuxiliaryQualifier(qualifier());
        _queryDataSource.fetchSpecification().setIsDeep(true);
        _queryDataSource.fetchSpecification().setUsesDistinct(false);
        _queryDataSource.fetchSpecification().setRefreshesRefetchedObjects(false);

        if (_nextPageDelegate==null){
            ListPageInterface
listPage=D2W.factory().listPageForEntityNamed("Agent",session());
            listPage.setDataSource(_queryDataSource);
            listPage.setNextPage(this);
            return (WOComponent)listPage;
        } else
            return _nextPageDelegate.nextPage(this);
    }

    public EOQualifier qualifier(){
```

```
return displayGroup.qualifierFromQueryValues();}

public void setNextPageDelegate(NextPageDelegateddelegate) {
    _nextPageDelegate=delegate;
}

public E0DataSource queryDataSource(){ return _queryDataSource; }

public String entity() {
    return "Agent";
}

public QueryAgent(WOContextaContext) {
    super(aContext);
}
}
```

User Templates

Sometimes you need to change the appearance of a Direct to Web page without freezing the component. You might want to change all of the pages for a particular task (list pages for example) without freezing components for every entity. Or you might want to use the Web Assistant to fine-tune pages having your custom appearance.

Direct to Web allows you to generate, modify, and use templates. Templates are WebObjects components that Direct to Web can use to generate pages (list pages for example) for any entity. Direct to Web provides a number of prebuilt templates from which you generate *user templates*. User templates appear together with the prebuilt templates in the Web Assistant and you can apply them to pages in your project.

These are the advantages of using templates:

- You can use a template for any entity.
- You can modify the properties and page appearance with the Web Assistant.
- Since the template is a WebObjects component, you have control over its visual appearance. You can add any static or dynamic HTML elements you like, using a tool such as WebObjects Builder.
- You can add functionality to the template by editing the component's Java code, as well as by editing the bindings of the component's dynamic elements.

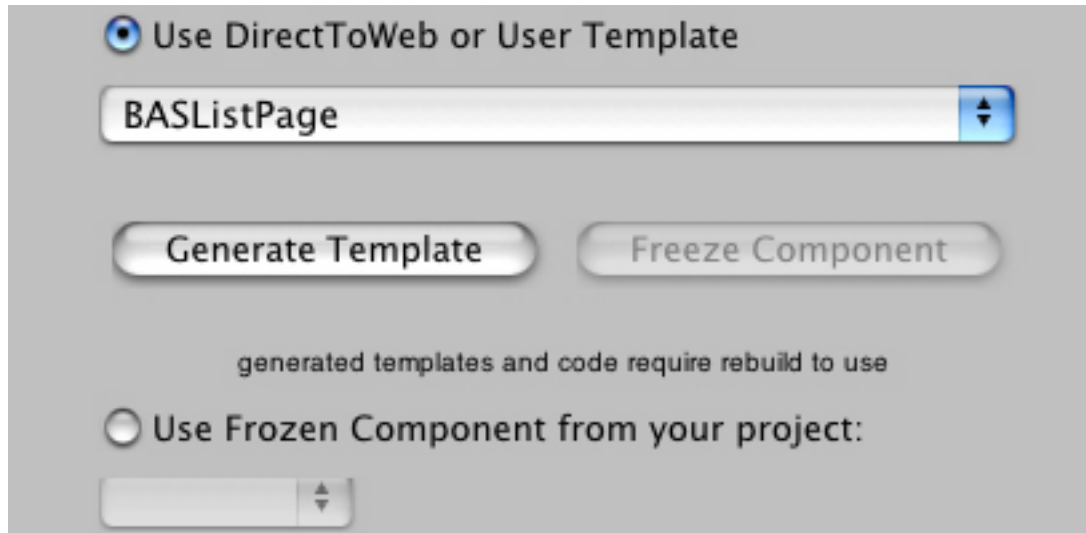
Templates are slower than frozen components since Direct to Web generates the pages that the user sees in the browser.

Generating a Template

Follow these steps to generate a template:

1. Click Expert Mode at the bottom of the Web Assistant window to enter Expert mode.

2. Click the Generation tab at the top of the WebAssistant.



3. Select the task corresponding to the page you want to generate.
4. Select “*all*” in the Entities pop-up list.

You must select “*all*” for the entities because the template is independent of the entity. You cannot select “*all*” for the task to generate multiple templates. You must generate the templates one at a time.

5. Make sure the “Use DirectToWeb or User Template” radio button is selected.
6. Click Generate Template.

The Generate Template window appears. It contains a text field with a default name for your template. You can edit the name if you choose.

7. Click the Ok button.

Direct to Web copies a component (with extension .wo) and a corresponding .java file from a predefined template and adds them to your project. You may have to wait a few moments for this process to complete. Your settings are automatically saved.

8. Rebuild and run your project, and restart the WebAssistant.

After you generate the template and rebuild your project, you can use the Web Assistant to apply the template to a Direct to Web page. See [“Customizing Pages”](#) (page 27).

Direct to Web Architecture

The Direct to Web framework works together with the WebObjects framework to generate web pages for nine database tasks including querying, editing, and listing. To do this, Direct to Web uses a task-specific component called a *Direct to Web template* that can perform the task on any entity. Direct to Web also translates the information that the Enterprise Objects Framework provides about the entity into values the Direct to Web template needs to render the page.

This chapter discusses the Direct to Web architecture and how Direct to Web generates a page. More specifically, it describes

- the different types of components that Direct to Web uses to render the pages
- the *Direct to Web context*, an instance of the `D2WContext` class that resolves the bindings in the Direct to Web template's binding file
- the *Direct to Web factory*, an instance of the `D2W` class that creates the Direct to Web pages
- how Direct to Web generates a query page
- the Direct to Web rule system, which contains application configuration information

Direct to Web Components

Direct to Web provides nine types of Web pages to perform the tasks shown in Table 1. See “[Dynamically Generated Pages](#)” (page 16) for more information about these tasks.

Table 2-1 Direct to Web tasks

Task	Description
Query	Allows the user to construct a query for a particular entity.
Query All	Displays all entities and lets the user construct queries on their attributes.
Inspect	Displays a single record of a given entity.
Edit	Displays a single record of a given entity and allows the user to change the record and save it to the database.
List	Displays several records of a particular entity in tabular form.
Select	Displays several records of a particular entity in tabular form and allows the user to choose one of them.
Edit relationship	Adds and removes objects from a relationship.
Confirm	Prompts the user to confirm that a record should be deleted.

Task	Description
Error	Displays information related to exceptions and other errors.

To render these pages, Direct to Web uses three types of components: Direct to Web templates, Direct to Web reusable components, and property-level components.

Direct to Web Templates

Direct to Web generates the task Web pages using instances of the `D2WPage` class (itself a descendent of the `WOComponent` class) called Direct to Web templates. A Direct to Web template defines the basic layout for the task's user interface. Direct to Web includes 29 templates: nine for the Basic look, ten for the Neutral look, and ten for the WebObjects look. For more information about looks, see [“The Different Looks for WebObjects Applications”](#) (page 11). The Direct to Web templates are listed in [Table 2-2](#) (page 40).

Table 2-2 Direct to Web Templates

Task	Basic Look	Neutral Look	WebObjects Look
Confirm	BASConfirmPage	NEUConfirmPage	WOLConfirmPage
Edit relationship	BASEditRelationshipPage	NEUEditRelationshipPage	WOLEditRelationship- Page
Error	BASErrorPage	NEUErrorPage	WOLErrorPage
Edit, Inspect	BASInspectPage	NEUInspectPage	WOLInspectPage
List, Select	BASListPage	NEUListPage	WOLListPage
List	BASMasterDetailPage	NEUMasterDetailPage	WOLMasterDetailPage
List, Select	BASPlainListPage	NEUPlainListPage	WOLPlainListPage
Query all	BASQueryAllEntitiesPage	NEUQueryAllEntitiesPage	WOLQueryAllEntities- Page
Query	BASQueryPage	NEUQueryPage	WOLQueryPage
Edit, Inspect		NEUTabInspectPage	WOLTabInspectPage

Some Direct to Web templates perform multiple tasks. For example, an `InspectPage` template also edits. For some tasks, there are multiple Direct to Web templates in a given look that you can use. For example, you can use a `ListPage`, a `PlainListPage`, or a `MasterDetailPage` template to perform the list task.

Like any other `WOComponent`, a Direct to Web template has an HTML template (`.html`) file and a bindings (`.wod`) file. What differentiates a Direct to Web template from other components is that it resolves its bindings with the help of the Direct to Web framework at runtime.

Note that a Direct to Web template is different from a component's HTML template (`.html`) file: a Direct to Web template is a special type of component while an HTML template is a file containing the HTML code that defines a component's appearance.

Direct to Web Reusable Components

Some Direct to Web templates can be viewed as implementing more than one task:

- A MasterDetailPage template consists of a select component at the top and an edit component at the bottom.
- An EditRelationshipPage template consists of a select component at the top and query, select, and edit components at the bottom.

Direct to Web displays these subcomponents with Direct to Web templates. For example, a NEUMasterDetailPage displays its select component using a NEUListPage and its edit component with a NEUInspectPage. However, the Direct to Web templates are not designed to be nested directly within other Direct to Web templates. To permit nesting, Direct to Web uses another type of component called a *Direct to Web reusable component*, which acts as an interface between the outer template and the inner template. There are five types of Direct to Web reusable components; they are listed in [Table 2-3](#) (page 41).

Table 2-3 Reusable components

Name	Task
D2WEdit	Edit
D2WInspect	Inspect
D2WList	List
D2WQuery	Query
D2WSelect	Select

[Table 2-4](#) (page 41) shows how the reusable components are used in the Direct to Web templates containing multiple tasks. The remaining templates do not contain Direct to Web reusable components.

Table 2-4 Direct to Web templates and reusable components

Direct to Web Template	Direct to Web Reusable Components Used
BASMasterDetailPageNEUMasterDetailPageWOLMasterDetailPage	D2WSelect, D2WEdit
BASEditRelationshipPageNEUEditRelationshipPageWOLEditRelationshipPage	D2WSelect, D2WQuery, D2WEdit

In addition to allowing the nesting of Direct to Web templates, Direct to Web reusable components can also be embedded in your own components; they are available on a palette in WebObjects Builder. See the *Direct to Web Reference* for more information about the individual Direct to Web reusable components.

Property-Level Components

Direct to Web uses *property-level components* to display, query, and edit individual properties of an entity. A property is an attribute or relationship of an entity. Direct to Web defines components for manipulating strings, dates, numbers, to-one relationships, to-many relationships, and other objects. For example, [Table 2-5](#) (page 42) lists some property-level components; these components work with numbers.

Table 2-5 Number property-level components (java.lang.Number, java.math.BigDecimal)

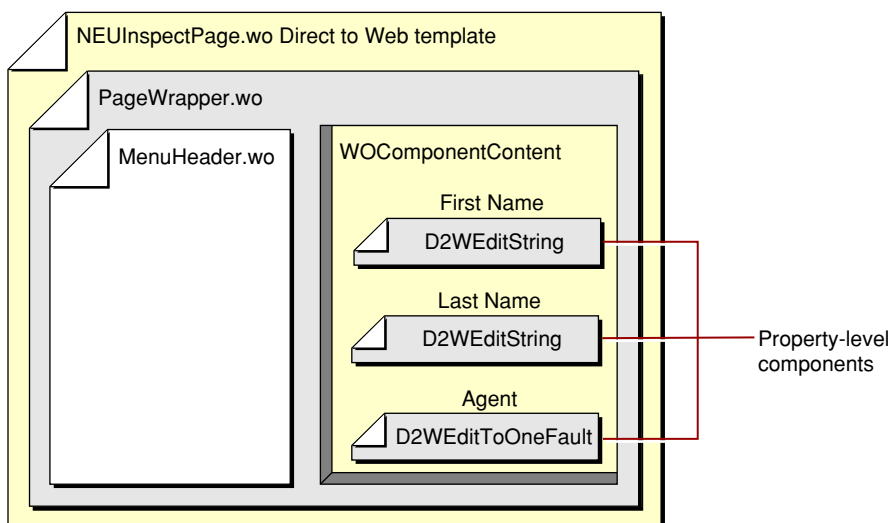
Display	Edit	Query
D2WDisplayNumber	D2WEditNumber	D2WQueryNumberOperator
D2WDisplayStyledNumber		D2WQueryNumberRange
D2WDisplayBoolean	D2WEditBoolean	D2WQueryBoolean

At runtime when a template displays a property, Direct to Web chooses which property-level component should display the property. The choice depends on the property's data type and how you configure the application with the Web Assistant.

Direct to Web Component Organization

[Figure 2-1](#) (page 42) shows the components in an edit page for the Customer entity in the Neutral look, with only the attributes `firstName`, `lastName`, and `agent` showing. The top-level component is a Direct to Web template called `NEUInspectPage.wo`. It contains the project's `PageWrapper.wo` component, which defines the overall layout of the page. The `PageWrapper.wo` component contains the `MenuHeader.wo` component, which defines the Direct to Web navigation menu. See [“The Structure of a Direct to Web Project”](#) (page 13) for more information about `PageWrapper.wo` and `MenuHeader.wo`.

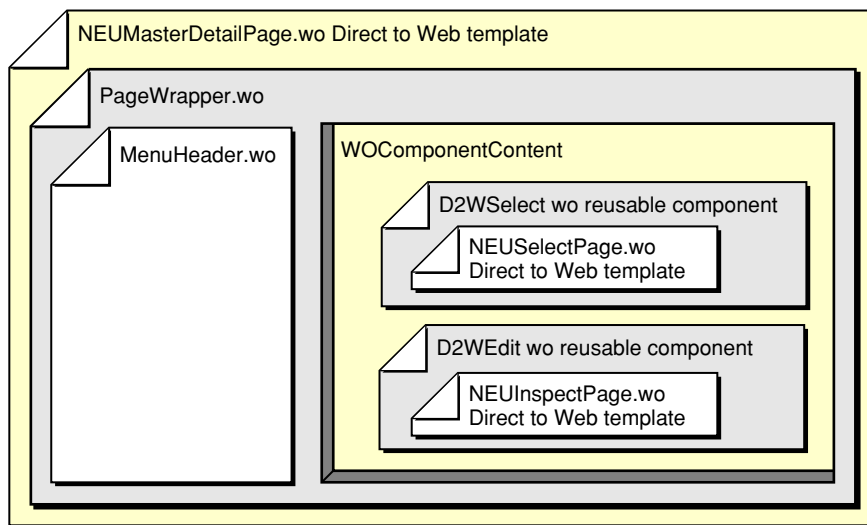
Figure 2-1 Edit page component organization



The `PageWrapper.wo` component content comes from the `NEUInspectPage` Direct to Web template. This content includes HTML input elements for the visible attributes of an entity. Each attribute appears in a separate property-level component that depends on the attribute's type. The `firstName` and `lastName` attributes are displayed using `D2WEditString` components. The `agent` relationship displays using a `D2WEditToOneFault` component.

Pages with nested Direct to Web templates contain Direct to Web reusable components. [Figure 2-2](#) (page 43) shows a master detail page in the Neutral look. The `NEUMasterDetailPage.wo` Direct to Web template contains `PageWrapper.wo` which in turn, contains `MenuHeader.wo`. `NEUMasterDetailPage.wo` also contains two reusable components that act as interfaces to the templates they display: a `D2WSelect` component and a `D2WEdit` component. Each of these components is actually a `WOSwitchComponent` that displays a template—the `D2WSelect` component displays a `NEUListPage` Direct to Web template and the `D2WEdit` component displays a `NEUInspectPage` Direct to Web template.

Figure 2-2 Master-detail page component organization



The Direct to Web Context

As mentioned earlier, a Direct to Web template is rendered using runtime information about the entities it displays. To translate that information into something you can bind to the template's dynamic elements, Direct to Web uses an instance of the `D2WContext` class called the Direct to Web context. This object has two functions: it maintains a *state dictionary* that holds the state of a Direct to Web template as it renders, and it provides values that you can bind directly to attributes of dynamic elements. Each instance of a Direct to Web template has an associated Direct to Web context.

Maintaining State

As the Direct to Web template changes state as it is rendered, the Direct to Web context changes state with it. Specifically, the Direct to Web context uses an `NSDictionary` containing

- the current task

- the current entity
- the current property (attribute or relationship)

The task and the entity remain constant while the template renders (with the exception of the query all template, for which only the task remains constant). The property does not stay constant, however. Consider an edit page. It displays the entity name and the entity's visible properties. To display the properties, the Direct to Web template iterates through them using a `WORepetition`. As it iterates, the Direct to Web context updates the information about the current property in its dictionary.

Providing Binding Values

Each Direct to Web template has a Direct to Web context called `d2wContext`, which implements the `EOKeyValueCoding` interface. Thus you can bind directly to keys that the context responds to. For example, [Listing 2-1](#) (page 44) shows the bindings file for a Direct to Web template that displays the name of the entity.

Listing 2-1 Bindings file for a Direct to Web template

```
String1: WOString {
    value = d2wContext.entity.name;
};
```

The Direct to Web context determines the values for the keys (`d2wContext.entity.name` for example) in one of three ways:

- it looks it up in its state dictionary
- it accesses application configuration information. (The Web Assistant is the primary way to modify the application configuration.)
- it derives values from the state and configuration information

Resolving Keys With the State Dictionary

The state dictionary contains the following entries:

Key	Description of Value
<code>task</code>	A string representing the current task.
<code>entity</code>	An <code>EOEntity</code> representing the current entity.
<code>propertyKey</code>	A string representing the key of the current property.
<code>attribute</code>	An <code>EOAttribute</code> representing the current attribute (<code>null</code> if the current property is a relationship.)
<code>relationship</code>	An <code>EORelationship</code> representing the current relationship (<code>null</code> if the current property is an attribute.)

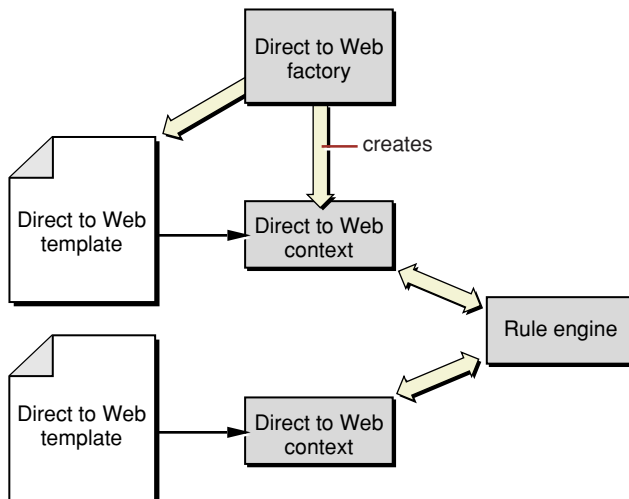
If the dictionary contains the key the template needs, the Direct to Web context resolves the key by returning the value in the dictionary. Otherwise the Direct to Web context resolves the key using one of the other ways.

Resolving Keys With the Application Configuration

Some keys can only be resolved using the application configuration information, which is stored as a database of rules. For example, a rule to determine the property-level component for the `dateReleased` attribute might be “If the task is ‘edit’, the entity name is ‘Customer’, and the property key is ‘lastName’ then the value for the `componentName` key is ‘D2WEditString.’”

The Direct to Web context uses the *rule engine* to resolve keys that aren’t in its dictionary. [Figure 2-3](#) (page 45) shows how the rule engine relates to the Direct to Web context. “[The Rule System](#)” (page 49) contains detailed information on how the rule engine works.

Figure 2-3 Direct to Web Architecture



Resolving Derived Values

The rule engine also provides objects that have methods that derive values from the Direct to Web context’s state dictionary. An example of a derived value is the name displayed for a property: a method converts a property name like `lastName` to a display string “Last Name”. Using derived values is discussed in more detail in “[The Rule System](#)” (page 49).

The Direct to Web Factory

Direct to Web pages are created by the Direct to Web factory, an instance of the D2W class (see [Figure 2-3](#) (page 45)). This object creates pages by instantiating a Direct to Web context and a Direct to Web template for each page. “[Rendering a Direct to Web Page: An Example](#)” (page 46), shows how the factory generates a query page.

Rendering a Direct to Web Page: An Example

This section describes how the components, the Direct to Web context, and the Direct to Web factory interact while creating and rendering a query page for the Customer entity. This example begins with the user viewing a query all page in the Basic look and clicking a hyperlink labeled “more..” in the Customer row, which links to the query page. The query all page is rendered using a `BASQueryAllEntitiesPage.wotemplate`.

Creating the Query Page

When the user clicks the hyperlink, the hyperlink invokes the `showRegularQueryAction` method defined in the `D2WQueryAllEntitiesPage` class, the superclass of the `BASQueryAllEntitiesPage` class. [Listing 2-2](#) (page 46) shows the implementation of the `showRegularQueryAction` method.

Listing 2-2 `D2WQueryAllEntitiesPage.showRegularQueryAction`

```
public WComponent showRegularQueryAction()
{
    QueryPageInterface newQueryPage= D2W.factory().queryPageForEntityNamed
        (entity().name(), session());
    return (WComponent)newQueryPage;
}
```

The Direct to Web factory object creates a Direct to Web context and initializes its `NSDictionary` by setting the value for the `task` key to “query” and the value for the `entity` key to the Customer `EOEntity`.

Table 2-6 Initial Direct to Web context dictionary

Key	Value
task	“query”
entity	<EOEntity Customer>

To determine which Direct to Web template to create, the Direct to Web factory asks the Direct to Web context for the value of the `pageName` key. Since this key is neither in the dictionary nor a derived value from the dictionary, the Direct to Web context enlists the aid of the rule engine. [Listing 2-3](#) (page 46) shows the rules that resolve the key.

Listing 2-3 Rules used to resolve the `pageName` key

```
((look= "BasicLook") and (task = "query")) => pageName = "BASQueryPage"

*true* => look = "BasicLook"
```

A rule has a left-hand side and a right-hand side separated by “=>”. The left-hand side specifies a condition that must be met for the rule to be a candidate to “fire,” or resolve a key. The right-hand side specifies the key-value assignment that takes place when the rule fires. Since the left-hand side for the second rule is true, the rule always fires when the Direct to Web context wants the value for the `look` key.

There are three sources for the rules that the rule engine uses:

- **The Direct to Web framework** defines rules that determine the default application behavior.
- **You** can write your own rules that override the framework's defaults rules.
- **The Web Assistant** generates rules involving the specific application information. These rules are generated automatically as you configure the application with the Web Assistant; you don't have to write them.

See [“The Rule System”](#) (page 49) for more information about rules.

The two rules that resolve the `pageName` key (defined in the Direct to Web Framework) fire and the Direct to Web context returns “BASQueryPage” as the value for the `pageName` key.

Knowing the template name, the Direct to Web factory object creates a page using the `WOComponent.pageWithName` method. Then it attaches the Direct to Web context to the newly generated page.

Rendering the Direct to Web Template

Now the WebObjects framework begins to render the template. [Listing 2-4](#) (page 47) shows excerpts from the HTML template for the BASQueryPage Direct to Web template; the listed portions of the file are discussed in this example. [Listing 2-5](#) (page 48) shows the corresponding sections in the bindings file.

First, the Direct to Web template displays the page wrapper. To do so, it needs to resolve the `WOComponentName` binding for the `PageWrapper` `WOSwitchComponent` (see [Listing 2-5](#) (page 48)). A `WOSwitchComponent` displays a nested component that has the name specified by its `WOComponentName` binding, which in this case is bound to the `d2wContext.pageWrapperName` key. Since the Direct to Web Context can't find it in its dictionary, it invokes the rule engine to resolve the key, which fires the rule:

```
*true*=> pageWrapperName = "PageWrapper"
```

The Direct to Web context returns “PageWrapper” for the `WOComponentName` binding and the `WOSwitchComponent` displays the application's `PageWrapper.wo` component. The template continues to render, resolving its keys in a similar way.

Listing 2-4 BASQueryPage.html excerpts

```
<WEBOBJECTNAME=PageWrapper>
.
.
  <WEBOBJECT NAME=ResourceRepetition>
.
.
  ... <WEBOBJECT NAME=ResourceLabel>:... </WEBOBJECT>
.
.
  <WEBOBJECT NAME=ResourceInputRepresentation></WEBOBJECT>
.
.
  </WEBOBJECT>
</WEBOBJECT>
```

Listing 2-5 BASQueryPage.wod excerpts

```

PageWrapper:WOSwitchComponent {
    WOComponentName = pageWrapperName;
    ...
}

ResourceInputRepresentation:WOSwitchComponent {
    WOComponentName = d2wContext.componentName;
    ...
}

ResourceLabel: WOString {
    ...
    value = d2wContext.displayNameForProperty;
}

ResourceRepetition: WORepetition{
    ...
    item = d2wContext.propertyKey;
    list = d2wContext.displayPropertyKeys;
}

```

Setting the Property Key

When the template begins to render the query fields for the entity’s attributes and relationships (like the agent and contact info) it encounters the WORepetition labeled ResourceRepetition. See Listing 2-4 (page 47) and Listing 2-5 (page 48). The WORepetition’s list attribute is bound to d2wContext.displayPropertyKeys. Since displayPropertyKeys is not in its dictionary, the Direct to Web context resolves the key using the rule engine, which causes the following rule to fire:

```
*true*=> displayPropertyKeys = "defaultPropertyKeysFromEntity"
```

The defaultPropertyKeysFromEntity key refers to a method that derives a value based on the Direct to Web context’s dictionary. See “The Rule System” (page 49) for more information about the how derived values are handled. The defaultPropertyKeysFromEntity method returns an NSArray containing the Customer entity’s property keys, which resolves the WORepetition’s list binding.

As the repetition iterates, it sets the item attribute for each of the objects in the list. The first object is the string “agent”. Since item is bound to d2wContext.propertyKey, the Direct to Web context sets the value for propertyKey in its dictionary to “agent”. At the same time, it sets the value for the attribute key to null and the value for the relationship key to the agent EORelationship, since a Customer’s agent property is a relationship and not an attribute. Now the Direct to Web Context dictionary contains the information listed in Table 2-7 (page 48).

Table 2-7 Direct to Web context dictionary after setting propertyKey

Key	Value
task	“query”
entity	<EOEntity Customer>

Key	Value
propertyKey	"agent"
attribute	null
relationship	<EORelationship agent>

Resolving Keys That Depend on the Property

As the WebObjects framework continues to render the ResourceRepetition WORepetition, it encounters the ResourceLabel WOString. See Listing 2-4 (page 47). The value attribute is bound to `d2wContext.displayNameForProperty`. This causes the following rule to fire:

```
*true*=> displayNameForProperty = "defaultDisplayNameForProperty"
```

The derived value for the `defaultDisplayNameForProperty` key is implemented by a method that capitalizes the property key in the context's dictionary, inserts spaces between words with mixed case, and returns the resulting name "Agent", which the template displays.

Next, the template displays the property-level component that queries for the `agent` relationship. Since this component is known only at runtime, the Direct to Web template displays it with a WOSwitchComponent called ResourceInputRepresentation. See Listing 2-5 (page 48). The WOSwitchComponent's WComponentName attribute is bound to `d2wContext.componentName`. When the context evaluates this key, the following rule fires:

```
((task= "query") and (propertyType = "r")
  and (not (relationship.isToMany= (java.math.BigDecimal)"1")))
=> componentName = "D2WQueryToOneField"
```

Thus the WOSwitchComponent displays a `D2WQueryToOneField.wo` reusable component from the DirectToWeb framework.

The rest of the template renders in a similar way.

The Rule System

Direct to Web stores its configuration in the form of rules. The following is an example of a rule:

```
((task= "query") and (not (attribute = null))
  and (attribute.valueClassName= "java.lang.String"))
=> componentName = "D2WQueryStringComponent"
```

A rule consists of five parts, of which three are shown in the example:

- **a left-hand side**, which is separated from the right-hand side by "=>"

The left-hand side specifies a condition that must be true for the rule to be a candidate to fire. In this case, the task must be "query"; the attribute must not be `null` and its value must be a Java String.

- **a right-hand-side key** (`componentName` in this case)
The right-hand-side key must match the key the Direct to Web context is seeking for the rule to be a candidate to fire.
- **a right-hand-side value** ("`D2WQueryStringComponent`" in this case)
The right-hand-side value specifies the value for the right-hand-side key when the rule fires. It can be a constant value (as in this case) or a value computed by a method.
- **a priority**
The priority helps Direct to Web decide which rule should fire when there are several candidates. See "[Deciding Which Candidate Should Fire](#)" (page 50) for more information about the rule priority.
- **an assignment class specifier**
The assignment class sets the value for the right-hand-side key when the rule fires. The default assignment class, `Assignment`, defined in the Direct to Web framework, assigns a constant value like "`D2WQueryStringComponent`". When the right-hand-side value is derived using a method, the assignment class specifies a class that contains the method.

Deciding Which Candidate Should Fire

When the Direct to Web context asks for the value for a key, there are typically several rules that can fire. For example, consider the following rules to resolve the `componentName` key:

```
*true*=> componentName = "D2WUneditable"

(task = "inspect") =>componentName = "D2WDisplayString"

((task = "inspect") and
 (attribute.valueClassName= "com.webobjects.foundation.NSTimestamp"))
=> componentName = "D2WDisplayDate"
```

The left-hand side of the first rule is always true. Such rules are useful for providing “fallback” values when all other rules fail to fire. Note that if the left-hand side for the third rule is true, all three rules are candidates for firing. The rule engine must choose which rule will fire.

To make the choice, Direct to Web employs a priority system. Each rule has a priority. The single rule with the highest priority fires. By convention, the following priorities are used in Direct to Web.

Priority	Description
0-10	Reserved for Direct to Web framework and fallback rules
100-105	WebAssistant rules

If two or more rules have the same priority, the rule with the most specific left-hand side (applying to the least number of cases) fires. Direct to Web measures how specific a rule is by counting the number of clauses joined by an *and* operator; the more clauses the rule has, the more specific it is. Clauses joined by the *or* operator count as a single clause.

If two or more rules have the same priority and are equally specific, Direct to Web arbitrarily chooses one.

The rule system resolves keys recursively. In other words, the rule system can resolve a rule based on the outcome of another rule. Consider a rule for the `pageName` key:

```
((look= "BasicLook") and (task = "query")) => pageName = "BASQueryPage"
```

The `look` key is defined by a rule like this:

```
*true*=> look = "BasicLook"
```

To resolve the `pageName` key, the rule engine asks the Direct to Web context for values for the `look` and the `task` keys. The Direct to Web context then invokes the rule engine to resolve the `look` key. This extra step isn't necessary for the `task` key; it's already in the Direct to Web context's local dictionary. Although recursive rules are powerful, they can hamper Direct to Web's performance.

To see the rules that fire as Direct to Web renders pages, run your application with the switch `-D2WTraceRuleFiringEnabled YES`.

Rules and the Web Assistant

The Web Assistant defines rules that pertain to specific entities and properties in your application, unlike the rules from the Direct to Web framework. These rules have a priority of 100, which override the default rules in the Direct to Web framework. Consequently, if you want to define your own default rules that the Web Assistant can override, you need to specify them with a priority between 11 and 99.

When you click Update in the Web Assistant window, the settings are compared to the settings on the server and the appropriate rules are added or deleted from the rule database in memory. When you click Save in the Web Assistant window, the rule database is stored in the application's `user.d2wmodel` file.

To build the Web Assistant's list of available task pages and property-level components, Direct to Web uses the rule system differently from when it renders a page. Instead of firing the single best candidate rule to find a particular key, Direct to Web asks for all rules that can resolve the key given the state of the Direct to Web context and collects the resulting values into a list that the Web Assistant presents to you. Two special keys are handled this way: `pageAvailable` for collecting task pages and `componentAvailable` for collecting property-level components.

If you want to see which rules the Web Assistant creates and deletes at runtime, you can run your application with the switch `-D2WTraceRuleModificationsEnabled YES`.

Rule Firing Cache

When a rule fires, its right-hand-side value is cached to improve Direct to Web's rendering performance. Once the value is cached, subsequent requests for the key may cause the rule engine to access the cache to resolve its value instead of finding a rule to fire. The cache is retained for the life of the application or until you click Update, Save, or Revert in the Web Assistant.

It is important to note that the right-hand-side value is cached based on the values of a collection of keys that does not necessarily include all of the keys on the left-hand side of the rule. Only the values of a list of *significant keys* and the right-hand-side key are used to uniquely identify the cache entry. By default, the significant keys are

- task

- entity
- propertyKey
- configuration

The configuration key refers to the named configuration of the current page. You can add to this list using the D2W class's `newSignificantKey` method.

Consider this rule:

```
((task= "edit") and (entity.name = "Customer")
  and (propertyKey = "agent"))
  => componentName = "D2WEditToOneRelationship"
```

When it fires, Direct to Web creates the cache entry described in [Table 2-8](#) (page 52).

Table 2-8 Example of Cached Rule

task	"edit"
entity	<EOEntity Customer>
propertyKey	"agent"
configuration	null
key	componentName
value	"D2WEditToOneRelationship"

If the Direct to Web context is asked for the value of the `componentName` key again, and the context's values for `task`, `entity`, `propertyKey`, and `configuration` are unchanged, this rule does not fire. Instead, the rule system uses the cached value. On the other hand, if the value of any of these keys changes, the cache entry no longer applies and the rule engine must find a rule to fire to resolve the `componentName` key.

Caching Gotchas

If you watch the rules as they fire (with `-D2WTraceRuleFiringEnabled YES`), you may find rules that fire when you expect Direct to Web to use the values in the cache. Or rules you expect to fire don't appear because Direct to Web gets the values from the cache.

To see how a rule might fire when you expect its value to be cached, consider the rule that resolves the `look` key, which defines whether the application is using the Basic look, the Neutral look, or the WebObjects look. Suppose the rule is

```
*true*=> look = "NeutralLook"
```

The first time this rule fires is when the Direct to Web factory asks for the name of the Direct to Web template to display the QueryAll page. The following cache entry is created:

Table 2-9 Example of Cached Rule First Time it Fires

task	"queryAll"
------	------------

entity	null
propertyKey	null
configuration	null
key	look
value	"NeutralLook"

Note that the `entity` key is `null`. The next time Direct to Web asks for the `look` key is when it wants to know the background color for the Query form table. The entity is still `null` so Direct to Web gets the value from the cache.

Now the `QueryAll` template begins to iterate through the entities. It sets the first entity to the `AdministratorEOEntity`. This time the `entity` key is no longer `null` so the old cache entry does not apply. Consequently, the rule engine fires the `look` rule again.

What is more difficult to debug is when the rule engine resolves a key using the cache when you expect a rule to fire. This happens when the outcome of the rule depends on a key that is not cached (that is, not in the list of significant keys). This can arise in an application that has different behavior depending on the user's access privileges.

Consider an online real estate database application that behaves differently based on the user's access privileges. In particular, the real estate agent (access level 1) sees the `AgentListListing` template and the customer (access level 3) sees the `CustomerListListing` template. You can set this up with these rules:

```
((task= "list") and (entity = "Listing") and (session.user.accessLevel= 1))
=> pageName = "AgentListListing"
```

```
((task = "list") and (entity= "Listing") and (session.user.accessLevel = 3))
=> pageName = "CustomerListListing"
```

Suppose the real estate agent logs into the application and accesses the list page. Direct to Web creates this cache entry:

Table 2-10 Cached Rule for Real Estate Agent Login

task	"list"
entity	<EOEntity Listing>
propertyKey	null
configuration	null
key	pageName
value	"AgentListListing"

task	entity	propertyKey	configuration	key	value
"list"	<EOEntityListing>	null	null	pageName	"AgentListListing"

Later a customer logs on and accesses the list page. Instead of showing the customer list page, Direct to Web displays the real estate agent's list page, which is an obvious security violation. This happens because the second rule never fires. Instead, the cache entry from the first rule resolves the value for the `pageName` key.

To fix the application, you need to add `session.user.accessLevel` to the list of significant keys using the D2W class's `newSignificantKey` method. For example,

```
D2W.factory().newSignificantKey("session.user.accessLevel");
```

Customizing a Direct to Web Application

This chapter shows some of the ways you can customize the behavior of a Direct to Web application. Specifically, this chapter discusses how to

- add a logo to your Direct to Web pages
- use Direct to Web in other WebObjects applications
- modify the Direct to Web factory
- create a custom property-level component
- modify a Direct to Web task
- add a task to Direct to Web
- add methods to the Direct to Web rule engine

Adding a Logo to Your Direct to Web Pages

The Neutral look is well-suited for adding a logo because it doesn't already display the Apple or WebObjects logos, unlike the Basic and WebObjects looks. To add a company logo to your Direct to Web pages, you need to add the HTML code that displays the logo to two components: `Main.wo`, which implements the login page, and `PageWrapper.wo`, which provides the backdrop for all of the pages Direct to Web generates. In this example, the Apple logo is added to `PageWrapper.wo`.

1. Edit `PageWrapper.html`. Line five reads:

```
<TABLE BORDER=0 CELLPADDING=4 CELLSPACING=0 WIDTH="99%">
```

After this line add

```
<TR>
  <TD><WEBOBJECTNAME=Image1></WEBOBJECT></TD>
  <TD COLSPAN=3></TD>
</TR>
```

2. Edit `PageWrapper.wod` to specify the bindings for the `WOImage`. Add the following lines:

```
Image1:WOImage {
  filename = "Apple-fade.gif";
  framework = "JavaDirectToWeb";
}
```

Now all of the pages Direct to Web generates appear with the Apple logo. If you want to use a custom image already added to your WebObjects project, `filename` would be the image name, and `framework` would most likely be "app".

Using Direct to Web in Other WebObjects Applications

WebObjects applications that were not generated using the Direct to Web option in the wizard can use the Direct to Web framework to display query, list, edit, and other pages in the Direct to Web repertoire. Making use of Direct to Web can be a convenient shortcut for many applications when all they need is a standard database-related page. They can use Direct to Web in one of two ways:

- by embedding Direct to Web components in the pages of their application
- by linking to a dynamically created Direct to Web page and appropriately implementing the action method invoked when the link is clicked

Embedding Direct to Web Components

Using a Direct To Web component is not much different than using any other reusable component. In this section you create a D2WQuery that searches for Listing entities based on the address. The procedure is the following:

Note: The example below uses a WODisplayGroup with Listing as its entity from the RealEstate EOModel. The easiest way to put a WODisplayGroup in your component is using drag-and-drop. Drag the Listing entity in EOModeler onto your component in WebObjects Builder, and the WODisplayGroup will be created. You shouldn't need to configure anything, so just click Add when prompted.

1. Add the Direct to Web and the Direct to Web Generation frameworks to your project. The path for these frameworks is `/System/Library/Frameworks` and the file names are `JavaDirectToWeb.framework`, `JavaDTWGeneration.framework`, and `JavaEOPProject.framework`.

2. Decide which Direct to Web component you want to use and become familiar with its API.

See the “D2W” class reference in the WebObjects API Reference for summaries of these components.

3. Put a WebObjects tag for the component in the page that is to display it.

```
<WEBOBJECT NAME=MyD2WQuery></WEBOBJECT>
```

This is a step you can complete in WebObjects Builder. The reusable components are available from the “DirectToWeb” palette.

4. Make the appropriate bindings for the component.

```
MyD2WQuery:D2WQuery {
    entityName="Listing";
    displayKeys="(address.city,address.state, address.zip)";
    queryDataSource=listingDisplayGroup.dataSource;
}
```

All embedded components require an `entityName` binding to specify the entity the page works with. Extra bindings could be required, depending on the functionality of the page. For example, List pages require a `dataSource` binding. If you want to use a named configuration, specify it with a `pageConfiguration` binding. See “Named Configurations” (page 33) for more information about named configurations.

You can also complete this step in WebObjects Builder.

5. If necessary, implement the action method for the component.
6. You can customize embedded Direct to Web components using the Web Assistant. You can launch the Web Assistant using the `appletviewer` tool.

When the Web Assistant acts upon an application that was not generated using Direct to Web, it does not automatically track which page is displayed. Thus you must set the task and entity for the page you want to modify in Expert mode. For example, if you want to customize a list page for Customers, click Expert mode and select the list task and the Customer entity. You can then customize a component in the same way you customize Direct to Web pages. Also, when you click Update to send the new settings to the application, the browser does not automatically refresh your page. You must either click the Reload button in your browser or (especially when you select a new task or entity) you must renavigate to the page.

By default, your component appears in the Neutral look.

Note: If you set the display keys as shown above—hard-wire the values in the code—then those values are used not any rules you set using the WebAssistant.

Linking to a Direct to Web Page

The second way to use Direct to Web in your application is to link directly to a dynamically generated page of the appropriate type. The D2W class defines methods that create components (inspect, query, list, and so on) defined for an entity in a session. The returned component objects implement the appropriate interface:

```

QueryPageInterface queryPageForEntityNamed (String entity,
    WOSession session);
ListPageInterface listPageForEntityNamed (String entity,
    WOSession session);
EditPageInterface editPageForEntityNamed (String entity,
    WOSession session);
InspectPageInterface inspectPageForEntityNamed (String entity,
    WOSession session);
SelectPageInterface selectPageForEntityNamed (String entity,
    WOSession session);
EditRelationshipPageInterface editRelationshipPageForEntityNamed
    (String entity, WOSession session);
QueryAllPageInterface queryAllPage (WOSession session);

```

To create a named configuration page, you use the method

```

WOComponent pageForConfigurationNamed
    (String namedConfiguration, WOSession session);

```

To link to a Direct to Web page, you need to implement an action method that returns a Direct to Web component implementing the appropriate page interface.

Implementing the Action Method

To implement the action methods that link to Direct to Web pages, you must use methods of the D2W class and the page-specific Direct to Web interfaces. You also need to specify a hyperlink, active image, or similar HTML control that invokes the action method. The following example shows such a hyperlink; first, the `WEBOBJECT` tag in the HTML template file:

```
<WEBOBJECTname=D2WListPage>D2W list page</WEBOBJECT>
```

Then, in the `.wod` file, bind the hyperlink to the `d2wList` action method:

```
D2WListPage:WOWHyperlink {
    action = d2wList;
}
```

The action method must return a component (that is, a `WOComponent` object) that implements the interface appropriate to the required type of page. For example, if you want to link to a dynamically generated list page, the component returned must implement the `ListPageInterface` interface. The D2W class provides methods that create such components:

```
import com.webobjects.directtoweb.*;
.
.
.
public WOComponent d2wList(){
    ListPageInterface lpi =
        D2W.factory().listPageForEntityNamed("Listing", session());
    lpi.setDataSource(listingDisplayGroup.dataSource());
    lpi.setNextPage(this);
    return (WOComponent)lpi;
}
```

Notice that before you return the component, you must set things such as the data source for the component and the page to go to when users click Return (`setNextPage`). This example assumes that the data you want to display is contained in the `EODDataSource` for a `WODisplayGroup` called `listingDisplayGroup`.

Setting Up a Next-Page Delegate

For some pages, you need to specify the action method that navigates to the next page. Consider a component called `MyListPage` that displays a list of objects and has a hyperlink that adds objects to the list. To determine which objects to add, the component invokes a Direct to Web query page.

The query page behavior differs from the normal Direct to Web query page behavior in two ways:

- Normally a query page creates a Direct to Web list page when the user clicks the Search DB button. In this example, however, the query page jumps back to the `MyListPage` component.
- When the query executes, it adds the objects to the list of objects displayed by the `MyListPage` component, an action the normal query component does not perform.

To implement this custom behavior, you need to define and instantiate a delegate object in addition to creating the query component. The delegate object must implement the `NextPageDelegate` interface and include a method called `nextPage`, which is invoked when the user clicks the submit button for the query page (Query DB). The query component's next page delegate must be assigned to this object using the `nextPageDelegate` method.

Listing 3-1 (page 59) shows how this can be done.

Listing 3-1 Sample code that sets up a next-page delegate

```
public class MyListPage extends WOComponent {

    public WODisplayGroup myDisplayGroup;
    ...

    public WOComponent showD2WQuery(){
        QueryPageInterface qpi=
            D2W.factory().queryPageForEntityNamed("Customer",session());
        qpi.setNextPageDelegate(new NextPageDelegate() {
            // delegate implementation
            public WOComponent nextPage(WOComponent sender) {
                EODataSources ds;
                NSArray objectsToAdd;
                Enumeration e;
                ds = ((QueryPageInterface)sender).queryDataSource();
                objectsToAdd= ds.fetchObjects();
                e = objectsToAdd.objectEnumerator();
                while (e.hasMoreElements()){
                    EOEnterpriseObject eo;
                    eo = (EOEnterpriseObject)e.nextElement();
                    MyListPage.this.myDisplayGroup.
                        insertObjectAtIndex(eo,0);
                }
                return MyListPage.this;
            }
        });
        return (WOComponent)qpi;
    }
}
```

The `showD2WQuery` method first creates a query page component. It then creates a delegate object and sets the query page's next-page delegate to the newly created object. Finally, the method returns the new query page component, which causes WebObjects to display the query page.

The next-page delegate object contains an action method called `nextPage`, which is invoked when the user clicks Query DB on the query page. This method gets the query data source containing the query specification. Next, the `nextPage` method fetches the objects matching the query specification and adds them to the display group in the `MyListPage` component. Finally, it returns the `MyListPage` component, where you have access to the newly fetched objects in the `WODisplayGroup`. The `WODisplayGroup` could be hooked up to a `WORepetition` which then lists all the fetched objects.

Note: If you get a null pointer exception when trying this example, one thing to check is that the `WODisplayGroup myDisplayGroup` gets instantiated.

Setting Up the Page Wrapper

Every application that links to a dynamically created Direct to Web page should have a component called `PageWrapper.wo`. This component acts as a "wrapper" for the dynamically generated content and can have customized header and footer material. If your application does not have a page wrapper, Direct to Web

displays your pages in an empty page wrapper. [Listing 3-2](#) (page 60) and [Listing 3-3](#) (page 60) show how a `PageWrapper.wo` component might be set up. You can use a text editor, Project Builder, or WebObjects Builder to construct this component.

Listing 3-2 PageWrapper.html example

```
<HTML>
  <WEBOBJECT name=Head></WEBOBJECT>
  <BODY>
    <WEBOBJECT name=BodyContainer>
      <WEBOBJECT name=Body></WEBOBJECT>
    </WEBOBJECT>
  </BODY>
</HTML>
```

Listing 3-3 PageWrapper.wod example

```
BodyContainer:WBody {
  filename = "Background.gif";
  framework = "JavaDirectToWeb";
  bgcolor="#c0c0c0";
  TEXT = "#000000";
  LINK = "#0000F0";
  VLINK = "#0000F0";
  ALINK = "#FF0000";
}

Head : D2WHead {
  _unroll = true;
}

Body: WComponentContent {
  _unroll = true;
};
```

The only required component in `PageWrapper.wo` is the `WComponentContent`. The other components shown in the example are optional, and you can create your own header, footer, and body-container components for your dynamically generated pages.

The `_unroll` attribute, when set to `true`, enables the Web Assistant to generate a static component from the dynamically generated one.

Modifying the Direct to Web Factory

You can override the page-creation methods of the `D2W` class to customize the components they return. The `defaultPage` method of the `D2W` class is one you might want to override; this method returns the application's default page, which is the query-all page by default.

If you make a subclass of `D2W` to override or add certain methods, make sure you call the `setFactory` class method with an instance of the new class as the argument.

[Listing 3-4](#) (page 61) is an example of how to extend the `D2W` class.

Listing 3-4 Customizing the D2W class

```

import com.webobjects.foundation.*;
import com.webobjects.eocontrol.*;
import com.webobjects.directtoweb.*;
import com.webobjects.appserver.*;

public class D2WExtendedFactory extends D2W {

    public WOComponent defaultPage(WOSession session) {
        return WOApplication.application().
            pageWithName("MyDefaultPage", session.context());
    }
}

```

Add the following Java code to the application constructor in the `application.java` file:

```
D2W.setFactory(new D2WExtendedFactory());
```

Creating a Custom Property-Level Component

Sometimes you need a custom property-level component that implements specialized behavior or that works with a type of attribute that Direct to Web doesn't already support (such as a QuickTime movie). Direct to Web provides a property-level component called `D2WCustomComponent` to make it easier to create such a component. To use the `D2WCustomComponent`, you first create a custom component. Then you use the Web Assistant to tell Direct to Web to use it.

Specifying the Custom Component

The custom property-level component must be a reusable component that defines two keys: `object` and `key`. The value for `object` specifies the enterprise object that the reusable component manipulates, for example, a `Customer` object. The value for `key` specifies the key of the property that the component manipulates, for example, `firstName`.

You can get the property by defining two instance variables in your component:

```
EOEnterpriseObject object;
String key;
```

and using the `EOKeyValueCoding` method `valueForKey`:

```
String first = object.valueForKey(key);
```

To store a value for the property, you use `takeValueForKey`:

```
object.takeValueForKey(first, key);
```

If you are using a nonsynchronizing component, you need to get the values for the `object` and `key` bindings using the `valueForBinding` method before you get or store the property.

```
EOEnterpriseObject object = valueForBinding("object");
String key = valueForBinding("key");
```

“[EditStatePopup Listings](#)” (page 75) shows an example custom component called `EditStatePopup` that uses a pop-up list to edit US states. The example lists the `.html`, `.wod`, and `.java` files that specify the custom component.

Using the Custom Component With Direct to Web

Once the custom component has been compiled, you can use the Web Assistant to instruct your Direct to Web application to use it. Follow these steps to configure your application to use the pop-up list state editing component for the `state` attribute on the `ListingAddress` edit page:

1. Open the Web Assistant. See “[Customizing Your Application With the WebAssistant](#)” (page 24).
2. Click the “Expert mode” button.
In Expert mode, you can make changes that affect all pages for a given task.
3. Select the Properties tab.
4. Select the `edit` task and the `ListingAddress` entity.
5. Click `state` in the Show column.
6. In the right column, choose `D2WCustomComponent` from the pop-up list.
7. Enter the name `EditStatePopup` in the Component text box.

You can also use the Web Assistant to configure your application to use a custom component on every edit page by following these steps:

1. Select the `edit` task and “*all*” for the entity.
2. Select the data type your component handles in the type browser in the second column.
3. In the right column, choose `D2WCustomComponent` from the pop-up list.
4. Enter the name of your component in the Component text box.

You can also configure your application to use the custom component by setting rules with the rule editor instead of using Web Assistant. To do so, use the rules shown in [Listing 3-5](#) (page 62). (To open the `user.d2wmodel` in the Resources group of Project Builder’s main window, drag its icon in Project Builder onto the `RuleEditor` icon in the Dock.)

Listing 3-5 Setting Rules With Rule Editor

```
((task= "edit") and (entity.name = "ListingAddress")
  and (propertyKey = "state")
  => componentName = "D2WCustomComponent")

((task = "edit") and (entity.name= "ListingAddress")
  and (propertyKey = "state")
  => customComponentName= "EditDatePopup")
```

You also need to set the assignment class (Assignment) and the priority (100) for each rule. For more information about using the rule editor, see [“Adding Rules to Define the Default Behavior”](#) (page 66).

Modifying a Direct to Web Template

To change the appearance and function of all the pages Direct to Web generates for a particular task, you need to modify its template. You can use the Web Assistant to generate the template and WebObjects Builder to edit it.

To illustrate how to modify a template, this example shows how to add a hyperlink to the NEUListPage Direct to Web template. The hyperlink simply redraws the page for now, but in [“Adding a New Direct to Web Task to Your Application”](#) (page 64) it will be modified to navigate to a new task page. For this example, it is easiest to start by creating a new project.

1. Create a Direct to Web application using the Real Estate database. For the look, use the Neutral look. See [“Creating a Direct to Web Project”](#) (page 10).
2. Launch your application and run the Web Assistant. See [“Using Your Direct to Web Application”](#) (page 14) and [“Customizing Your Application With the WebAssistant”](#) (page 24).
3. Generate a Direct to Web template for the list task called “NEUListPage2” using the Web Assistant. [“Generating a Template”](#) (page 37).
4. From Project Builder, double-click `NEUListPage2.wo` in the project’s Web Components group. This opens the Direct to Web template in WebObjects Builder.

Add a hyperlink after the first instance of the metallic Return button. This instance is displayed when the list is not empty.

Add a `WOImage` inside the hyperlink.

Bind the following attributes to the `WOImage`:

Attribute	Value
filename	“EditMetalBtn.gif”
framework	“JavaDirectToWeb”
border	“0”

Add an action called `editList` that returns `null`.

Bind the action to the Edit List hyperlink’s `action` attribute.

5. Save your template.
6. Build and launch your application. Navigate to a list page. It should now appear with an Edit button.

Since the `editList` action returns `null`, the hyperlink just redraws the page. In [“Adding a New Direct to Web Task to Your Application”](#) (page 64), you will modify the action to create a new task page.

Freezing Your Modified Direct to Web Template

You can freeze a page based on your Direct to Web template using the Web Assistant. See “[Generating Components](#)” (page 34) for more details. Make sure that your template is selected in the template pop-up list in the third column of the Web Assistant.

Note: When you freeze a Direct to Web template, you lose any instance variables and methods you add to it. To avoid an unknown key exception when you display the frozen component, you need to add the same variables and methods to your frozen component.

In the example above, the `editList` action is missing from any component created by freezing the `NEUListPage2` template.

Sometimes you add components to your Direct to Web template that have child components. For example, tables, `WOConditionals`, `WORepetitions`, and any reusable components with a `WOComponentContent` dynamic element all have child components. If you add such components to your Direct to Web template, add an attribute called `_unroll` and bind it to `YES`. This attribute enables Direct to Web to include the component's children when you freeze the template.

Adding a New Direct to Web Task to Your Application

A Direct to Web application can be expanded to handle new tasks. This section describes how to add an example task called `edit-list`. The `edit-list` task page looks like a regular list page but renders the data in text fields so the user can edit the data. To create a task and use it in a Direct to Web application you need to

- create the Direct to Web template that executes the task
- add rules to configure the default behavior of the page and make it available to the Web Assistant
- add a hyperlink to an existing Direct to Web template that links to the new task page

Once you have added the task, you can use it with any entity and configure it with the Web Assistant.

Creating the Direct to Web Template

The easiest way to create a Direct to Web template is to modify one that the Web Assistant generates. The `edit-list` page most closely resembles the list page.

1. For this example, begin with the project you created in the last section, “[Modifying a Direct to Web Template](#)” (page 63). You should have a modified `NEUListPage2` Direct to Web template in your project.
2. Create a Direct to Web template called “`NEUEditListPage`” based on the `NEUListPage` by following steps 2–4 in “[Modifying a Direct to Web Template](#)” (page 63).
3. Open `NEUEditListPage.wo` in WebObjects Builder. Follow the directions below to modify the template.
4. Remove the first and last columns of the main table. The left column contains an icon that opens the edit page. The right column contains an icon that deletes the record. Since the list page provides these functions, they are not duplicated on the `edit-list` page.

- Now wrap the main table inside a WOForm. The easiest way to do this is to use the source view. Find the line

```
<WEBOBJECTNAME=NavBar></WEBOBJECT>
```

Change it to

```
<WEBOBJECT NAME=NavBar></WEBOBJECT><WEBOBJECTNAME=Form1>
```

Find the first `</CENTER>` tag in the file. Add a closing `</WEBOBJECT>` above this line.

- In the next three steps, you change the Return hyperlink into a submit button and add a Save submit button that saves the changes to the database. Find the line

```
<WEBOBJECT NAME>ShowCancel><WEBOBJECTNAME=BackLink><WEBOBJECT  
NAME=ReturnButton></WEBOBJECT></WEBOBJECT></WEBOBJECT>
```

Add

```
<WEBOBJECTNAME=SubmitChanges></WEBOBJECT>
```

between the last two `</WEBOBJECT>` tags. The line should now look like this:

```
<WEBOBJECTNAME>ShowCancel><WEBOBJECT NAME=BackLink><WEBOBJECT  
NAME=ReturnButton></WEBOBJECT></WEBOBJECT><WEBOBJECTNAME=SubmitChanges>  
</WEBOBJECT></WEBOBJECT>
```

- In the bindings file, find the bindings for the ReturnButton WebObjects element. Change it so it reads as follows:

```
ReturnButton:WOImage {  
    alt = "Cancels changes";  
    border = "0";  
    filename = "CancelMetalBtn.gif";  
    framework = "JavaDirectToWeb";  
    name = "Cancels changes";  
}
```

- Add the following bindings to the bindings file:

```
Form1:WOForm {  
    action = backAction;  
}  
  
SubmitChanges: WOImageButton{  
    action = saveChanges;  
    alt = "Saves your changes";  
    border = "0";  
    filename = "SaveMetalBtn.gif";  
    framework = "JavaDirectToWeb";  
    name = "Saves yourchanges";  
}
```

The `backAction` method is defined by the `NEUEditListPage` component's superclass, `D2WListPage`.

- Switch back to the layout view to check for HTML syntax errors.
- Save the `NEUEditListPage.wo` file.

11. Modify `NEUEditListPage.java` by adding the `saveChanges` method defined in [Listing 3-6](#) (page 66).

Listing 3-6 Implementation of the `saveChanges` method in `NEUEditListPage.java`

```
public WComponent saveChanges() {
    WComponent nextPage = this.nextPage();
    try {
        session().defaultEditingContext().saveChanges();
    } catch (Exception exception) {
        ErrorPageInterface epi = (ErrorPageInterface)
            D2W.factory().errorPage(session());
        epi.setMessage(exception.toString());
        epi.setNextPage(this);
        nextPage = (WComponent)epi;
    } finally {
        return nextPage;
    }
}
```

This Java code tries to save changes to the session's editing context. If it fails, it returns an error page. Otherwise, it returns to the page that called it.

Adding Rules to Define the Default Behavior

When a Direct to Web application launches, it looks for rules in two files in your project, `user.d2wmodel` and `d2w.d2wmodel`, and merges them with the rules defined in the Direct to Web framework. The Web Assistant modifies the rules in the `user.d2wmodel` file. For basic customization tasks you don't need to edit this file by hand. When you add rules that change the default behavior of your application, you modify the `d2w.d2wmodel` file.

Defining the default behavior of a new task page also informs the Web Assistant of the new task. Specifically, the Web Assistant collects a list of tasks based on the `task = "taskName"` clauses on the left-hand side of rules.

Direct to Web provides an application called `RuleEditor` to edit the rules.

For the edit-list page example, you need to

- revise the Web Assistant rules in the `user.d2wmodel` file

The Web Assistant created two rules when you generated the `NEUEditListPage` Direct to Web template. These rules state that the `NEUEditListPage` can and should be used with the list task. Since you are creating an edit-list task, which exclusively uses the `NEUEditListPage` Direct to Web template, these rules must be changed.
- create a `d2w.d2wmodel` file and put rules that define the edit-list page default behavior in it

Modifying the Web Assistant Rules

1. Quit the Web Assistant if it is still running. You will modify the `user.d2wmodel` file using `RuleEditor` instead of the Web Assistant because the Web Assistant can't perform the change.

2. Open the `user.d2wmodel` in the Resources group of Project Builder in RuleEditor.

The top half of the window lists the rules. The bottom half of the window edits the selected rule.

3. Select this rule:

```
((task = "list") and (look = "NeutralLook"))
=> pageAvailable = "NEUEditListPage"
```

by clicking on it, and click Delete. The `NEUEditListPage` is now no longer a candidate to perform the list task.

4. Select this rule:

```
(task = "list") => pageName= "NEUEditListPage"
```

In the Value text field in the bottom right corner of the screen, enter `"NEUListPage2"` (including the quotation marks) and press Return. Direct to Web displays a `NEUListPage2` component as the default list page.

5. Save the rule file.

Choose File > Save.

Adding New Default Rules

1. Create a new rule file.

In RuleEditor, choose File > New.

2. Click New to create a new rule.

The first rule you add is

```
(task = "editList") => displayPropertyKeys=
"defaultPropertyKeysFromEntityWithoutRelationships"
```

using `DefaultAssignment` as the custom class. The rule's priority is 50, which overrides the default Direct to Web framework rules but not the Web Assistant rules. The details of adding this rule are covered in the next step.

This rule specifies that the properties on the edit-list page include the entity's attributes but not its relationships. The `defaultPropertyKeysFromEntityWithoutRelationships` method is defined in the `DefaultAssignment` class in the Direct to Web framework.

3. In the first line of the browser labeled "Left-Hand Side," enter `(task = "editList")`, and press Enter.

Choose Custom from the Class pop-up list and enter

`com.webobjects.directtoweb.DefaultAssignment` in the Custom text box. The `DefaultAssignment` class contains methods that derive values from the state in the Direct to Web context.

Enter `displayPropertyKeys` in the Key text box.

Enter `defaultPropertyKeysFromEntityWithoutRelationships` in the Value text box.

Enter 50 in the Priority text box.

4. Add the following rules by repeating step 3. All of the rules have a priority of 50 and use the Assignment class (not the DefaultAssignment class). Assignment is available in the Class pop-up list; do not choose Custom.

This rule specifies that the NEUEditListPage Direct to Web template can be used to display an edit-list page:

```
(task = "editList") => pageAvailable = "NEUEditListPage"
```

This rule specifies that the default edit-list Direct to Web template is NEUEditListPage:

```
(task = "editList") => pageName= "NEUEditListPage"
```

This rule specifies the name of the banner for the edit-list page:

```
(task = "editList") => bannerFileName = "EditMetalBan.gif"
```

The following rule specifies that the D2WEditString property-level component can be used to edit strings on the edit-list page. To enter the left-hand side, use the And and Not buttons.

```
((task= "editList") and (not (attribute = null))
  and (attribute.className= "java.lang.String"))
=> componentAvailable= "D2WEditString"
```

This rule specifies that the default property-level component that edits strings on an edit-list page is D2WEditString:

```
((task= "editList") and (not (attribute = null))
  and (attribute.className= "java.lang.String"))
=> componentName = "D2WEditString"
```

This rule specifies that the D2WEditNumber property-level component can be used to edit numbers on the edit-list page:

```
((task= "editList") and (not (attribute = null))
  and ((attribute.className= "java.math.BigDecimal"
  or (attribute.className= "java.lang.Number"))))
=> componentAvailable= "D2WEditNumber"
```

This rule specifies that the default property-level component that edits numbers on an edit-list page is D2WEditNumber:

```
((task= "editList") and (not (attribute = null))
  and ((attribute.className= "java.math.BigDecimal"
  or (attribute.className= "java.lang.Number"))))
=> componentName = "D2WEditNumber"
```

If your EOModel had dates as some of the attributes, you would also add the two following rules:

```
((task = "editList") and(not (attribute = null))
  and (attribute.className= "com.webobjects.foundation.NSTimestamp"))
=> componentAvailable= "D2WEditDate"
```

```
((task= "editList") and (not (attribute = null))
  and (attribute.className= "com.webobjects.foundation.NSTimestamp"))
=> componentName = "D2WEditDate"
```

5. Save the file as `d2w.d2wmodel` in the top-level directory of your project.
6. Add the `d2w.d2wmodel` file to your project.

Choose Project > Add Files.

Select the `d2w.d2wmodel` file created in the previous step and click Open.

Select the Application Server target and click Add.

Adding a Hyperlink to the New Task Page

Now that Direct to Web can display the edit-list task page, you need to add a hyperlink to the list page to bring up the edit-list page.

1. Modify the `editList` method in `NEUListPage2.java` so it matches the implementation in listing below:

Listing 3-7 Implementation of the `editList` method in `NEUListPage2.java`

```
public ListPageInterface editList {
    ListPageInterface lpi =(ListPageInterface)D2W.factory().
        pageForTaskAndEntityNamed("editList",entity().name(),
            session());
    lpi.setNextPage(this);
    lpi.setDataSource(displayGroup().dataSource());
    return lpi;
}
```

The edit-list page uses the same interface as the list page (`ListPageInterface`) because the pages are very similar. The action method `editList` creates a new page using `pageForTaskAndEntityNamed` because the factory has no special method to create an edit-list page. You could add such a method if you like. See [“Modifying the Direct to Web Factory”](#) (page 60).

To specify the page to display when the user cancels the edits, the action method invokes `setNextPage`. When the user clicks the Cancel button, this list page will display. Finally, the action method sets the data source for the display group so it matches the list page’s data source. This ensures that the edit-list page displays the same objects the list page displays.

2. Build and launch your application. Navigate to a list page. Click the Edit button.

The edit-list page should appear. You can type in a field to edit its contents. If you click cancel, the edits are discarded. If you click save, the edits are saved to the database. You can page through the displayed objects with the navigation bar at the top of the screen. When you move from one page to another, the edit-list page discards the edits.

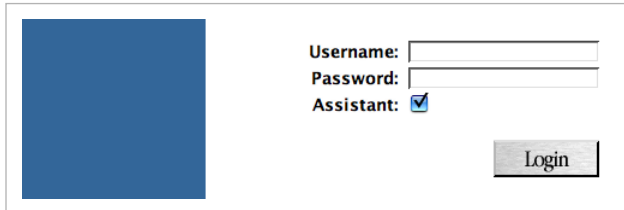
3. You can use the Web Assistant to further customize the edit-list page.

Adding Authentication to the Main Component

This section describes how to implement basic authentication in a Direct to Web application.

The default Main component in a Direct to Web application includes a login panel that resembles [Figure 3-1](#) (page 70).

Figure 3-1 Default Main component

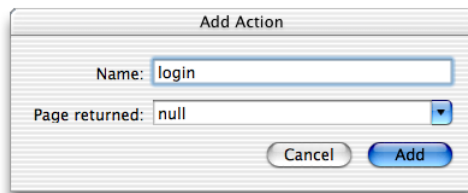


Unless you write custom code, clicking the Login button always lets the user into the application, regardless of what was entered into the Username and Password fields. Since the point of the login page is to protect the application from unauthorized users, it makes sense to implement a real authentication mechanism.

Hooking Up the Main Component

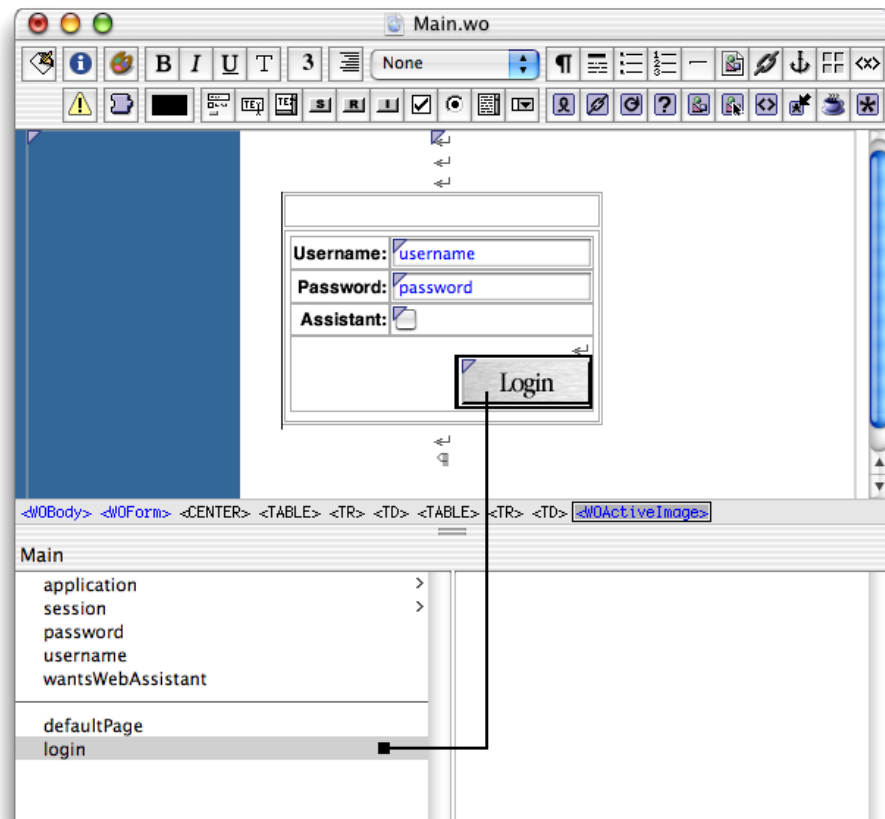
To implement login authentication, you need to edit the Main component and add some custom Java code. Open Main.wa in WebObjects Builder and add a new action called “login” returning `null`, as shown in [Figure 3-2](#) (page 70).

Figure 3-2 Add the login action



Change the binding for the `action` attribute of the Login button by dragging from the `login` action to the Login button, as shown in [Figure 3-3](#) (page 71). Choose “action” from the menu that pops up to bind it.

Figure 3-3 Dragging from the action method to the Login button



Add Logic

When you add an action to the Main component, WebObjects Builder generates Java code for the method based on the inputs specified in the Add Action dialog. In `Main.java`, you should see this code for the `login` action you added:

```
publicWOComponent login() {
    return null;
}
```

Clicking the Login button causes its `action` binding to be resolved. Since the action binding is "login", this calls the `login` method defined in the component. The `login` method simply returns `null`, which returns a new instance of the component calling the method in WebObjects. In our example, this means a new instance of the Main component is created, returned, and displayed when the Login button is clicked. See the WebObjects API Reference for the `WOComponent` class to better understand this behavior.

Rather than return the calling component (Main), you can add logic that returns a different page depending on the success of the authentication. To authenticate a user, you need to add some data to authenticate against. For now, simply add fields to `Main.java`. Later on, you'll implement authentication against a data source.

Add these fields:

```
private final String _username = "bucky";
```

```
private final String _password= "longfang";
```

You should notice that there are two instance variables already in `Main.java`, `username` and `password`. If you switch to WebObjects Builder, you can see that these variables are bound to the `value` attribute of the Username and Password WOTextFields, respectively. When the Login button is clicked, the strings contained in these textfields are automatically put into their corresponding variables, `username` and `password`. This means that you can add logic to test the value of each textfield against login information in a data store, or in this simple case, against variables in the class.

Before you add authentication logic, look at the `defaultPage` method in the class:

```
publicWOComponent defaultPage() {
    if (isAssistantCheckboxVisisble()){
        D2W.factory().setWebAssistantEnabled(wantsWebAssistant);
    }
    return D2W.factory().defaultPage(session());
}
```

In an uncustomized Direct to Web application, this returns the `queryAll` page. See “Query Pages” (page 17) for more information. If a user successfully authenticates, they will see this page. You will alter the `login` method to return this page if the user authenticates successfully. To do this, use a simple conditional:

```
if (_username.equals(username) && _password.equals(password))
```

If the conditional is true, return the default `queryAll` page. Otherwise, return the Main page with the login form. The complete code for `Main.java` is shown below:

```
import com.webobjects.foundation.*;
import com.webobjects.appserver.*;
import com.webobjects.directtoweb.*;

public class Main extends WOComponent{
    public String username;
    public String password;
    public boolean wantsWebAssistant= false;

    private final String _username= "bucky";
    private final String _password= "longfang";

    public Main(WOContext aContext){
        super(aContext);
    }

    public WOComponent defaultPage(){
        if (isAssistantCheckboxVisible()){
            D2W.factory().setWebAssistantEnabled(wantsWebAssistant);
        }
        return D2W.factory().defaultPage(session());
    }

    public boolean isAssistantCheckboxVisible(){
        return null == NSProperties.stringForKey("D2WebAssistantEnabled")
            || NSProperties.booleanForKey("D2WebAssistantEnabled");
    }

    public WOComponent login(){
        if (_username.equals(username)&& _password.equals(password))
```



```

        return defaultPage();
    }
    return null;
}

```

Add Better Logic

In a deployed application, you would likely want to authenticate users to information in a data store. Fortunately, WebObjects make this easy for you. By packaging the username and password into a dictionary object, you can use the EOUilities class to query a data store for a match of the given username and password. To use EOUilities, be sure to import the `com.webobjects.eoaccess` package.

The following code example uses the RealEstate EOModel. The `isAuthenticated` method checks to see if the given username and password match an Agent entity's `login` and `password` attributes, respectively. Here is the code:

```

public boolean isAuthenticated() {
    NSMutableDictionary userCredentials= new NSMutableDictionary();

    if (username == null || password == null)
        return false;

    userCredentials.setObjectForKey(username, "login");
    userCredentials.setObjectForKey(password, "password");    NSArray foundObjects
    =
    EOUilities.objectsMatchingValues(session().defaultEditingContext(),
    "Agent", userCredentials);
    if (foundObjects.count()== 1)        return true;

    NSLog.out.appendln("Authenticationfailed.");    return false;}

```

Lastly, you want to modify the `login` method as follows:

```

public WComponent login() {
    if (isAuthenticated())
        return defaultPage();

    return null;
}

```

When a user successfully authenticates, the default Direct to Web page is returned.

EditStatePopup Listings

The EditStatePopup.wo property-level component is listed here. The component has an HTML template (.html) file, a bindings (.wod) file, and a source (.java) file.

EditStatePopup.html

```
<WEBOBJECTNAME=PopUpButton1></WEBOBJECT>
```

EditStatePopup.wod

```
PopUpButton1:WOPopUpButton {
    list = stateList;
    selection = state;
}
```

EditStatePopup.java

```
import com.webobjects.foundation.*;
import com.webobjects.appserver.*;
import com.webobjects.eocontrol.*;
import com.webobjects.eoaccess.*;

public class EditStatePopup extends WOComponent {
    protected String state;
    protected EOEnterpriseObject object;
    protected String key;
    protected NSMutableArray stateList;

    public EditStatePopup(WOContext context) {
        super(context);
    }

    public void takeValuesFromRequest(WORequest request, WOContext context)
        throws NSValidation.ValidationException {
        super.takeValuesFromRequest(request, context);
        try {
            object.takeValueForKey(state, key);
        } catch (Exception exception) {
            throw (new NSValidation.ValidationException("Incorrect state
                input"));
        }
    }
}
```

EditStatePopup Listings

```
    }  
  }  
  
  public NSArray stateList(){  
    if (stateList == null){  
      stateList = newNSMutableArray(new Object[] {  
        "AL","AK", "AZ", "AR", "CA","CO", "CT", "DE", "FL","GA",  
        "HI","ID", "IL", "IN", "IA","KS", "KY", "LA", "ME","MD",  
        "MA","MI", "MN", "MS", "MO","MT", "NE", "NV", "NH","NJ",  
        "NM","NY", "NC", "ND", "OH","OK", "OR", "PA", "RI","SC",  
        "SD","TN", "TX", "UT", "VT","VA", "WA", "WV", "WI","WY"  
      });  
    }  
    return stateList;  
  }  
  
  public String state() throwsException {  
    state = (String)object.valueForKey(key);  
    return state;  
  }  
  
  public void setState(StringnewState) throws Exception {  
    state = newState;  
  }  
}
```