

Portalizing Domino Applications for WebSphere Portal

Integrating existing Domino applications into a portal

Portlet builders from IBM, Bowstreet, and CONET

Step-by-step integration techniques applied to practical scenarios



Tommi Tulisalo
Christopher Heltzel
Camilo Rojas
Michael Ticknor
Oliver Trabert
Marko Viksten



International Technical Support Organization

**Portalizing Domino Applications
for WebSphere Portal**

September 2003

Archived

Note: Before using this information and the product it supports, read the information in “Notices” on page vii.

First Edition (September 2003)

This edition applies to IBM Lotus Domino 6 and IBM WebSphere Portal 4.2.

© Copyright International Business Machines Corporation 2003. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	vii
Trademarks	viii
Preface	ix
The team that wrote this redbook	ix
Become a published author	xi
Comments welcome	xi
Chapter 1. Introduction to portalizing Domino applications	1
1.1 The portal vision	2
1.2 Introduction to WebSphere Portal	2
1.2.1 The workplaces idea	4
1.2.2 What the workplace means for Domino applications	6
1.2.3 Benefits of WebSphere Portal and Lotus Domino together	7
1.3 Integrating Domino applications into portlets and workplaces	9
1.3.1 Introduction to portlets	9
1.3.2 Portlet applications	10
1.3.3 Introduction to places	11
1.3.4 The portalizing process	14
1.3.5 The portalizing challenge	15
1.3.6 Domino applications	16
1.3.7 Portlet patterns	19
1.3.8 Considerations for the portlet design	22
1.3.9 Considerations	28
1.4 Portal architecture considerations	31
1.4.1 Page aggregation concept	31
1.4.2 Themes and styles	32
1.4.3 Page customization	33
1.4.4 Using Domino LDAP with WebSphere Portal	34
1.5 Summary	35
Chapter 2. Integration techniques	37
2.1 Choosing an integration technique	38
2.1.1 Step 1: Pre-project preparation and training	38
2.1.2 Step 2: Identify project requirements and considerations	39
2.1.3 Step 3: Select the appropriate portlet pattern	39
2.1.4 Step 4: Select the appropriate integration technique	40
2.2 Integration techniques and development options	41
2.2.1 Using existing portlets	42

2.2.2	Domino JSP tag libraries	44
2.2.3	Developing Domino portlets using Java	45
2.2.4	Portlet builders	46
2.3	Case study: A simple sales tracking application	47
2.4	Deploying the case study portlets	54
2.4.1	Install portlets	54
2.4.2	Creating a place	56
2.4.3	Creating a page	57
2.4.4	Adding portlets to a page	58
Chapter 3. Using existing portlets		63
3.1	Overview	64
3.1.1	Technologies involved.	64
3.1.2	Software and tools used	64
3.1.3	Integration techniques.	65
3.2	Integrate using the QuickLinks portlet.	65
3.2.1	Considerations	66
3.2.2	Implementation details	67
3.3	Integrate using the Web Page portlet	74
3.3.1	Considerations	74
3.3.2	Implementation details	75
3.4	Integrate using the Web Clipping portlet	79
3.4.1	Considerations	80
3.4.2	Implementation details	81
3.5	Integrate using Lotus Notes portlets	98
3.5.1	Lotus Notes portlets	98
3.5.2	Considerations	99
3.5.3	Implementation details	100
3.6	Integrate using XML helper and RSS portlets	107
3.6.1	Considerations	108
3.6.2	Implementation details	109
3.7	Integrate using multiple portlets	121
3.7.1	Considerations	121
3.8	Reference material	136
Chapter 4. Using custom Domino JSP tag libraries		137
4.1	Overview of the Domino custom JSP Tag option	138
4.2	Technologies involved.	138
4.2.1	J2EE overview	139
4.2.2	JavaServer Pages.	143
4.3	Software and tools used	149
4.3.1	WebSphere Studio Application Developer 5	150
4.3.2	WebSphere Portal Toolkit for WebSphere Studio 4.2.5	153

4.3.3 Lotus Domino Toolkit for WebSphere Studio 1.0	154
4.4 Integration techniques	156
4.5 Integration using Domino custom JSP Tag libraries	159
4.5.1 Overview	160
4.5.2 Considerations	167
4.5.3 Implementation example	168
4.5.4 Conclusions to the custom Domino tags integration technique	198
4.6 Integration via Click to Action	199
4.6.1 Click to Action	199
4.6.2 Considerations	203
4.6.3 Implementation of the technique	203
4.7 Integration via people awareness	215
4.7.1 People awareness	215
4.7.2 Implementation of the technique	219
4.8 Reference Material	222
Chapter 5. Portlet development using Java: Technology review	225
5.1 Overview	226
5.1.1 Technologies involved	226
5.2 Technical introduction to portlets	227
5.2.1 Basic portlet terms	227
5.2.2 Model-view-controller (MVC) design pattern	227
5.2.3 Portlet API overview	229
5.2.4 Portlets and the Servlet API	229
5.2.5 Portlet concepts	231
5.2.6 Portlet applications	232
5.2.7 Basic elements of the Portlet API	233
5.2.8 Frequently used objects	235
5.2.9 Configuration objects	240
5.2.10 Miscellaneous objects	242
5.2.11 Portlet events	244
5.3 Accessing Domino data from portlets using Java and CORBA	247
5.4 Domino objects for Java API	255
5.5 Domino Rich Text	264
5.6 Lotus Collaborative Components API	270
5.7 Domino 6 new features for DIOP	273
5.8 Object pooling	274
5.9 Logging from portlets	282
5.10 Struts Portal framework	287
5.11 General portlet development guidelines	295
Chapter 6. Portlet development using Java: Integration examples	305
6.1 Software and tools used	306

6.1.1 Domino Toolkit for Java version 2.1	306
6.1.2 Hybrid integration techniques	306
6.2 Search functionality	308
6.3 Paging through the view	310
6.4 HelloWorldFromDominoServer portlet	313
6.5 Using JavaBeans in the sample portlet	324
6.6 Browsing Domino ACL portlet	329
6.7 How to use log4j	338
6.8 Session pooling	342
6.9 Reference material, links, Redbooks	352
Chapter 7. Portlet builders	355
7.1 Overview of the portlet builders option	356
7.2 IBM Portlet Builder for Domino	356
7.2.1 Implementation details	359
7.2.2 Implementation example	371
7.3 Bowstreet Portlet Factory for WebSphere	372
7.3.1 Implementation details	377
7.3.2 Implementation example	390
7.4 CONET Portlet Factory for Domino	394
7.5 Implementing the Sales Workplace example	403
7.6 Other portlet builders	409
7.6.1 Sofor Interactive Portlet Builder for Domino	409
7.6.2 Aprtix Portlet Connector	411
Appendix A. Data dictionary for case study	415
A.1 Product Form	416
A.2 Sales Person Form	416
A.3 Customer Form	417
A.4 Customer Contact Form	418
A.5 Sales Activity Form	419
Appendix B. Additional material	421
Locating the Web material	421
Using the Web material	421
Related publications	423
IBM Redbooks	423
Other publications	423
Online resources	424
How to get IBM Redbooks	426
Help from IBM	426
Index	427

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:


This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server™
developerWorks®
ibm.com®
iNotes™
iSeries™
zSeries®
AIX®
CICS®

Domino™
Domino Designer®
DB2®
Everyplace®
IBM®
Lotus Discovery Server™
Lotus Enterprise Integrator®
Lotus Notes®

Lotus®
Notes®
QuickPlace®
Redbooks™
Redbooks (logo) ™
Sametime®
WebSphere®

The following terms are trademarks of other companies:

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

C-bus is a trademark of Corollary, Inc. in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

SET, SET Secure Electronic Transaction, and the SET Logo are trademarks owned by SET Secure Electronic Transaction LLC.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook describes how to integrate existing Domino™ applications into the IBM WebSphere® Portal. We have coined the term “portalizing” to describe this effort.

We begin by explaining why portal integration is so useful for any company that has a Domino environment, and the importance of integrating Domino applications into the WebSphere Portal. We also explain some of the key concepts of portals and Domino application integration, and outline some recognized design patterns for Domino application integration.

Next, we preview the recognized integration options which are described in detail later in the book. We also introduce the sample Domino application we used for our portalizing exercises throughout the book.

The following chapters present detailed discussions about the integration options currently available:

- ▶ Using existing portlets that ship with WebSphere Portal, including QuickLinks, Web page, Web Clipper, NotesView, XML/XSL Helper, and RSS portlet
- ▶ Using custom Domino JSP tag libraries
- ▶ Using Java programming
- ▶ Using portlet builders, including software products from IBM, Bowstreet, CONET, and others

For each of the integration options, we provide an overview of the technology, an introduction to the software and tools used, and step-by-step examples of using the techniques to portalize our sample Domino application.

This book is aimed at Domino application developers or anyone else who wants to learn how to portalize their Domino applications.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Cambridge Center.

Tommi Tulisalo is a Project Leader for the International Technical Support Organization at Cambridge, Massachusetts. He manages projects whose objective is to produce redbooks on all areas of Lotus® Software products.

Before joining the ITSO in 2001, he was an IT Architect for IBM Global Services in Finland, designing solutions for customers, often based on Lotus software.

Christopher Heltzel is an Advisory I/T Specialist with IBM Software Services for Lotus (ISSL). Christopher has over six years of IT consulting experience in the areas of application development, systems administration, and collaborative technologies. He is a Principal Certified Lotus Professional in System Administration R4.5 and a Principal Certified Lotus Professional in Application Development R4-5 and ND6. He received his B.S. in Microbiology from Texas A&M University.

Camilo Rojas is an Accredited IT Specialist at IBM Colombia. He has been working with the WebSphere family of products for 3 years. Currently he works the WebSphere Portal server and Foundation products. He is certified in WebSphere Application Server 4, WebSphere Studio Application Developer 4, WebSphere Portal 4.1 and he is a Principal Certified Lotus Professional in Application Development R5. His areas of expertise range from Java (J2EE) development, to analysis of systems architecture, open source software and solution design. Prior to joining IBM, Camilo worked with Global DataTel with Lotus Domino services. Camilo can be reached at: camilor@co.ibm.com

Michael Ticknor is an Advisory I/T Specialist for IBM Software Services for Lotus in Cincinnati, Ohio. He has over 9 years of experience designing and developing mission-critical applications using a wide range of Lotus, WebSphere, and Internet-based technologies. He is a Sun Certified Web Component Developer for the J2EE Platform and a Principal Certified Lotus Professional for Notes® and Domino Application Development (R4, R5, & Notes & Domino 6). He holds a B.S. in Computer Science from the University of Kentucky. Contact Michael at michael_ticknor@us.ibm.com.

Oliver Trabert is a Senior Consultant with WP-Experts in Cologne, Germany (<http://www.wpexperts.com>). Oliver trains and coaches IBM/Lotus Business Partners who want to build a WebSphere Portal business practice. He has been involved in portalizing Domino applications since WebSphere Portal 2.1. Previously he worked for CONET and developed the idea and concepts for the CONET Portlet Factory product. Oliver specializes in developing high performance Domino/Portal integrations using Domino object pooling and advanced data caching. Oliver holds a Master of Science degree from Rheinische Friedrich-Wilhelms-University Bonn. You can contact Oliver at otrabert@wpexperts.com.

Marko Viksten is an IT Architect with IBM Global Services, Helsinki, Finland. Working in Business Consulting Services, his primary responsibility is designing and implementing WebSphere Portal-based projects, which often have strong emphasis in integrating Domino and collaborative technologies. Marko is also a

Principal Certified Lotus Professional in Application Development. He has more than seven years of experience in the IT technology sector.

The following IBM Redbooks™ and Redpapers have been extremely useful to the team and we have adapted and reused some of the content from these publications:

- ▶ IBM WebSphere Portal Application Developers Handbook, SG24-6897.
- ▶ WebSphere Portal Collaboration Services, REDP0319.

The team would like to say special thanks to **Carl Kriger**, Sr. Marketing Manager, Lotus Workplaces, for direction and contributions.

Thanks to the following people for their contributions to this project:

Alan Lepofsky, Peter Janzen, Steve Leland, Lun Xiao - IBM Software Group

Jonathan Booth, Nicole Carrier, Brian Chaput, Ted Snyder, Dee Zepf - Bowstreet

Markus Marenbach - CONET AG

William Tworek - ITSO Cambridge Center

Alison Chandler - ITSO Poughkeepsie Center

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an Internet note to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HYJ Mail Station P099
2455 South Road
Poughkeepsie, New York 12601-5400



Introduction to portalizing Domino applications

In this chapter we introduce WebSphere Portal and discuss the important role of Domino integration into the Portal. We show that for companies moving towards WebSphere Portal it is essential to be able to seamlessly integrate Domino applications.

There are many options for Domino integration into the Portal, and no one method fits all needs. Using a portalizing methodology can assist you in making the right choice for your specific requirements.

We also discuss important aspects of the Portal architecture that you need to understand to successfully portalize Domino applications.

1.1 The portal vision

Portals serve as a simple, unified access point to applications. Portals also do much more: they provide valuable functions like security, search, collaboration, and workflow. A portal delivers integrated content and applications, plus a unified, collaborative workplace. Indeed, portals are the next-generation desktop, delivering e-business applications over the Web to all kinds of client devices.

A complete portal solution should provide users with convenient access to everything they need to get their tasks done anytime, anywhere, in a secure manner. IBM's vision is that portals are the key to reach users and give them the experience of an e-business application. That is, portals provide the tools and user interface to access information and applications, and to manage the selection and personalization of content.

1.2 Introduction to WebSphere Portal

IBM WebSphere Portal allows you to establish customized portals for your employees, Business Partners, and customers. As illustrated in Figure 1-1, the framework architecture implemented in this product provides a unified access point to internal and external Web applications as well as portal access to other legacy applications. In this way, users sign on to the portal and receive personalized Web pages.



Figure 1-1 Horizontal and vertical portals

IBM WebSphere Portal was designed in response to the following fundamental business objectives:

- ▶ A single point of access to all resources associated with the portal domain
- ▶ Personalized interaction with the portal services
- ▶ Federated access to hundreds of data types and repositories, aggregated and categorized
- ▶ Collaboration technologies that bring people together
- ▶ Integration with applications and workflow systems

IBM and many industry analysts have coalesced around the concept of horizontal and vertical portals. Horizontal portals are the primary infrastructure upon which a portal is built. Vertical portals are built upon the horizontal layer and represent a specific portal instance, usually defined by a major topic or domain.

As illustrated in Figure 1-2, the horizontal portal infrastructure consists of several modular subsystems including the following:

- ▶ Presentation layer - a Web user interface plus pervasive device support
- ▶ Personalization - the ability to serve dynamic response to the user based on personal profiles
- ▶ Collaboration - tools that allow e-mail, team rooms, shared places, and so forth to be exchanged
- ▶ Portlets - a framework for easily attaching software modules (portlets) and services
- ▶ Applications and workflow - integration of legacy and new applications
- ▶ Search and navigation - categorizing repositories of content and searching them for relevant information
- ▶ Publish and subscribe - the ability to author new content and publish it to subscribers
- ▶ Administration and security - basic Web site services such as page designers, performance monitors, cluster services, and metadata management Integration (metadata sharing, XML, connectors, standards, EAI).

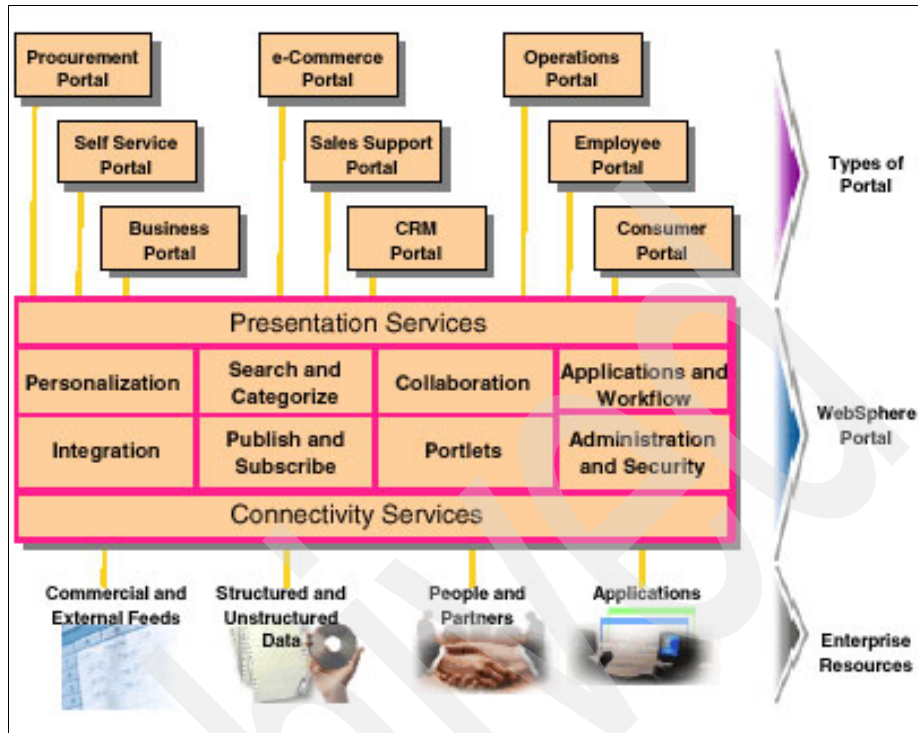


Figure 1-2 WebSphere Portal architecture

WebSphere Portal provides additional services such as single sign-on, security, Web content publishing, search, personalization, collaboration services, enterprise application integration, support for mobile devices, and site analysis.

Note: WebSphere Portal provides an extensible framework for interacting with enterprise applications, content, people, and processes. Self-service features allow end users to personalize and organize their own view of the portal, to manage their own profiles, and to publish and share documents with their colleagues.

1.2.1 The workplaces idea

The *workplace* is a central concept of the WebSphere Portal. A workplace offers a portal user a view that is role-based, and delivers information based on the business function need within the organization. That means it gives us a single point of access to all the applications and information that we need to do our job. This implies that different users will get access to different applications. And within the application different functions and content. Bottom line, the workplace

provides us with a tailored view that helps us to perform our specific business tasks and functions.

A workplace can include a large variety of applications. A few of those are very obvious - mail and calendar, for example - but you also see other business applications like HR, CRM, Learning and many more systems.

Figure 1-3 shows an example workplace.

The screenshot shows a dynamic workplace interface with several application windows and annotations. The main window displays a welcome message and a news article about IBM's transition to e-business. Other windows include a sidebar with navigation links, a calendar, a stock market scoreboard, and a list of team members. Annotations with arrows point to various parts of the interface, such as 'Content Management', 'Corporate Messages', 'Tailored News', 'People', 'Instant Messaging', 'Expert Location', 'e-Learning', 'Mail & Calendar', 'Business Apps', 'e-HR', 'Team Tools', and 'Applications'.

Figure 1-3 Dynamic workplace Example

In today's world many of us work in more than just one function. For example: the team members that wrote this book all have additional projects in the works. Team members can easily switch between the team workplace for the Redbook and their other project places. In the redbook project our role is content authors; in other projects we have a variety of other roles, from programmer to project leader to supervisor.

The workplace approach enables us to easily switch roles by just navigating between different workplace. IBM calls this concept the *Dynamic Work mode*.

Attention: The “workplaces” word and concept used in this book should not be confused with the new Lotus Workplace, which is the new platform for collaboration built on J2EE and relational database technology. Lotus Workplace combines market-leading collaborative capabilities with the WebSphere Portal framework to enable simplified access to people, information, and business processes on a single platform.

1.2.2 What the workplace means for Domino applications

It is important for us to understand why Domino integration into the Portal is so significant. Our experience with large Domino customers is that, over time, they utilized Domino for internal application development, and typically have developed dozens—even hundreds—of proprietary applications that contain critical data and business processes. If such a company now moves to WebSphere Portal, it is vital for the success of the Portal to be able to seamlessly integrate the Domino applications into the workplaces.

The history of Lotus Notes® and Domino

Let's look at the history of Domino to get a better understanding of the importance of Domino integration. In version 2, Notes was used primarily as an e-mail system. In subsequent versions Lotus enhanced programming elements, like the @Formula language, and views and forms that allowed users to build simple applications. Companies were impressed by the rapid application development concept that Notes introduced. Notes made it easy for developers or even power users to prototype applications. This new paradigm in application development was widely accepted by line of business (LOB) managers and users, and helped build the success of Notes/Domino. Since version 4.5, Notes could also be used for Web development; since then, it was ultimately accepted by many IT departments as a full-blown development platform.

Today many companies rely on their Domino infrastructure for business and mission-critical applications.

These range from document stores, to complex workflow applications supporting supply chains, to B2B business exchange Web sites.

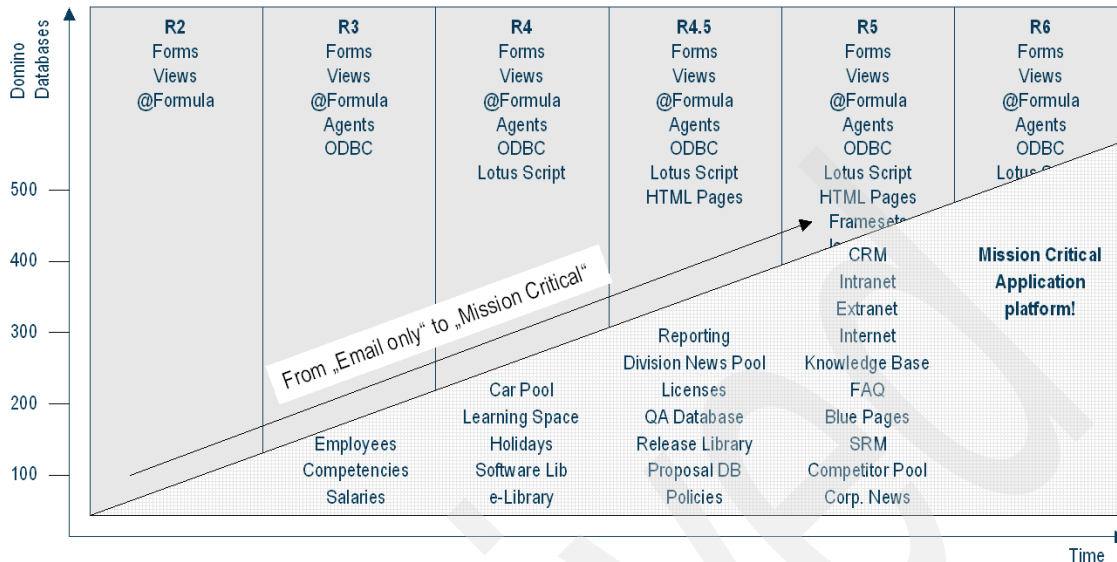


Figure 1-4 Domino's history: From e-mail only to a mission-critical platform

1.2.3 Benefits of WebSphere Portal and Lotus Domino together

If we consider the current situation—Domino is recognized as a successful and reliable application infrastructure—we naturally need to raise the question why not use it as our Portal Infrastructure as well? Why should we move to the WebSphere Portal framework?

If you have worked with Domino to build larger applications, you might have experienced a few limitations of the current Domino platform.

One of the most common challenges in developing large-scale Domino Web applications is to make the Domino HTTP access scale and perform well under a heavy load. Not that it isn't possible, but it requires a lot of experience, and sometimes you have to spend a lot of time, for example, to implement caching strategies.

A workplace also has the need to implement personalization. On the one hand you need to deliver content that is relevant to the user's role, on the other you also want to allow the user to filter the information that is presented to them. Implementing this on Domino, for example executing queries on the fly, puts heavy load on the Domino server and calls for a big investment in the server hardware.

A portal most often serves as a single point of access into all systems of an organization and therefore it is important to be able to implement interfaces into other applications, besides Domino.

Taking these considerations into account, it makes sense to look at an alternative scenario for a portal implementation. Not one where Domino is completely out of the picture, but one that combines the strength and the scalability of a J2EE application server platform with the strong collaboration and document-centric capabilities of Domino.

Such a combination can have lots of advantages since it is:

- ▶ Deeply integrated: WebSphere Portal provides the most robust integration with IBM's entire software portfolio, including your Lotus infrastructure; WebSphere Portal comes with out-of-the-box Portlets for IBM Lotus Notes and Domino, IBM Lotus Instant Messaging (Sametime®), IBM Lotus Discovery Server™, and much more.
- ▶ Open: WebSphere Portal lets you integrate best-of-breed applications. IBM is helping define the key open standards for the portal industry.
- ▶ Comprehensive: WebSphere Portal provides a complete framework that lets you answer all of your requirements for integrating your Lotus Domino assets with a single, powerful portal infrastructure.
- ▶ Right for Contextual Collaboration: WebSphere Portal weaves the advanced collaborative capabilities of Lotus into your portal, enabling your organization to achieve the highest level of productivity.
- ▶ Secure: WebSphere Portal provides granular security at the level you need for your applications and content. Use your Lotus Domino directory or other LDAP source.
- ▶ Flexible: WebSphere Portal lets you adjust the presentation of data for a wide range of client devices.
- ▶ Easily delegated: You can delegate sections of your portal to various “downstream” groups, enabling them to manage their segment of the portal.
- ▶ Personalized: Users can personalize content within portlets based on profiles and business rules.
- ▶ Global: WebSphere Portal supports multiple languages, enabling you to support your global organization.

1.3 Integrating Domino applications into portlets and workplaces

In this section we look at the methodology with which Domino applications can be portalized, that is, transformed and embedded into a portal.

To better understand this we need to first introduce the concepts of portlets and places.

1.3.1 Introduction to portlets

Portlets are the heart of a portal. The term portlet refers to a small portal application, usually depicted as a small box on the Web page. Figure 1-5 shows a sample Web page that contains six portlets.

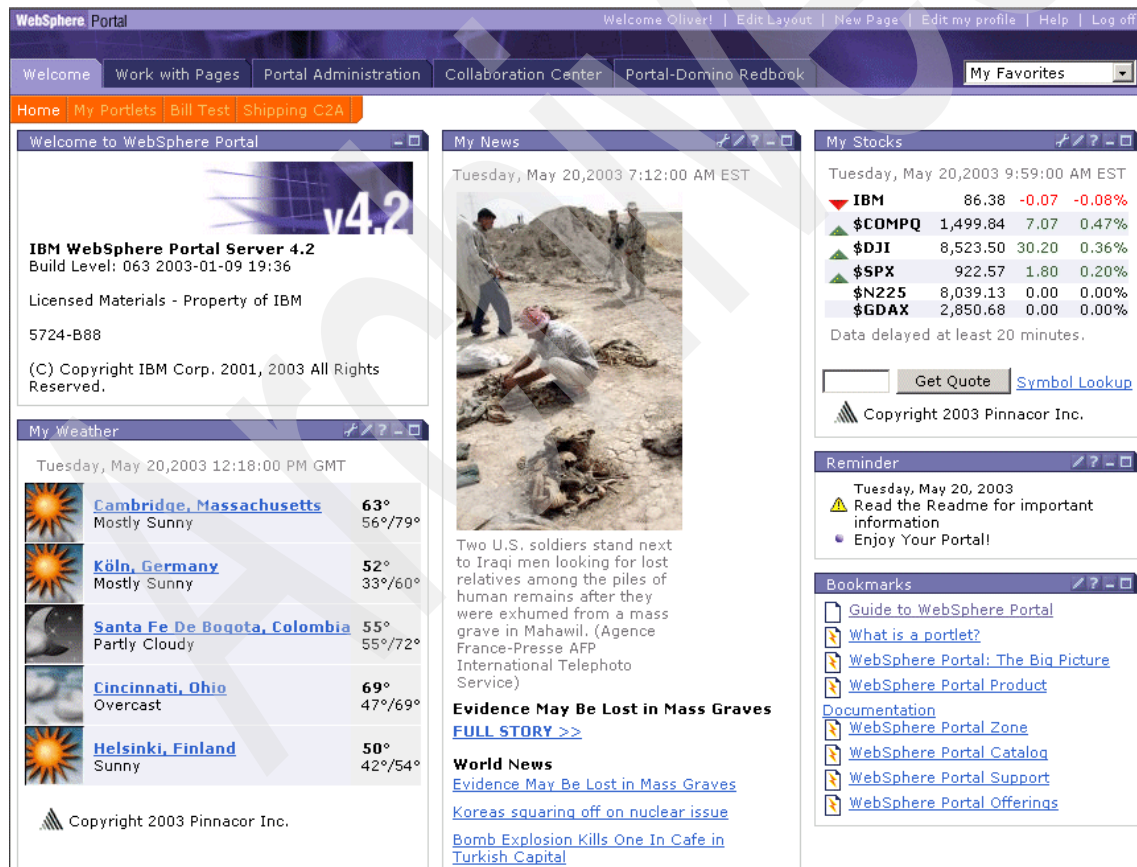


Figure 1-5 Portal page with several portlets displaying weather, news, and so forth

Portlets are reusable components that provide access to applications, Web-based content, and other resources. Web pages, Web services, applications, and syndicated content feeds can be accessed through portlets.

Portlet modes

Portlet modes allow a portlet to display a different user interface, depending on the task required of the portlet. A portlet has several modes of display, which can be invoked by icons on the portlet title bar: configure, edit, and help. The portlet shown in Figure 1-6 is currently displayed in view mode.

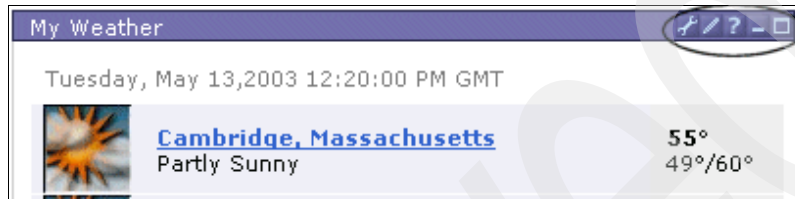


Figure 1-6 Portlet modes example

- ▶ A portlet is initially displayed in its view mode. As the user interacts with the portlet, it may display a sequence of view states, such as forms and responses, error messages, and other application-specific states.
- ▶ Help mode is used to provide user assistance about the portlet.
- ▶ Edit mode provides a page for users to change the portlet settings. For example, a weather portlet might provide an edit page for users to specify their location. Users must be logged into the portal to access edit mode.
- ▶ If Configure mode is supported by a portlet, it provides a page for portal administrators to configure portlet settings that are shared by all users.

Each portlet mode can be displayed in normal, maximized, or minimized state. When a portlet is maximized, it is displayed in the entire body of the portal page, replacing the view of other portlets. When a portlet is minimized, only the portlet title bar is displayed on the portal page.

1.3.2 Portlet applications

Portlets are more than simple views of existing Web content. A portlet is a complete application having multiple states and view modes, plus event and messaging capabilities.

Portlets run inside the portlet container of a portal server, similar to a servlet running on an application server. The portlet container provides a runtime environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the portal infrastructure to access user profile information,

participate in window and action events, communicate with other portlets, access remote content, look up credentials, and to store persistent data.

Generally, portlets are administered more dynamically than servlets. For example, portlet applications consisting of several portlets can be installed or removed while the server is running, and administrators can change the settings and access rights of a portlet while the portal is running, even in a production environment.

1.3.3 Introduction to places

The flexible design of WebSphere Portal allows users to initiate diverse places organized around specific projects, issues, or groups. It provides access to a wide range of tools, such as instant messaging, discussion databases, people awareness, and other collaborative tools that support teams. Best of all, users can create and modify these places to best meet their needs, without involvement from IT personnel. The result? Greater efficiency and faster results from any team effort.

Portal content is organized on *pages* that can be grouped. A page group becomes a *virtual place* when a user organizes content selectively and grants permission for other portal users to use the place. Within portal places, people can find other people and the right information quickly, building better teams and stronger ties to each other.

Places:

- ▶ Provide a new way to view, organize, and use portal resources, and are designed to improve communication, workflow, content management, and teamwork
- ▶ Present people and information in context with organizational or community needs
- ▶ Show links to individuals within places and portlets (people awareness)
- ▶ Have portlets that launch collaborative applications (for example instant messaging, Lotus Notes)

In addition, portlets have the ability to interact with each other. This functionality allows you to present information in a new context, which can greatly improve the user experience.

Let's look at this functionality in more detail.

Portlet cooperation

The portal server provides a mechanism for portlets to communicate with each other, exchanging data or other messages. In a production portal, portlet communication could be used to copy common data between portlets. This saves redundant typing by the user and makes the portal easier to use. For example, one portlet might display information about accounts while a second portlet displays information about transactions that have occurred for one of the accounts over the last 30 days. To do this, the transactions portlet needs to obtain the corresponding account information when it displays the transaction details. This is accomplished by communication between the two portlets, using portlet actions and portlet messages. In this example, the account portlet creates a portlet action and encodes it into the URL that is rendered for displaying transactions. When the link is clicked, the action listener is called, which then sends a portlet message to send the necessary data.

Programmatic messaging helps unify portlet applications that access different back-end applications. However, this is relatively static, and requires planning and design work in advance. The portlets exchanging messages must already know about each other in order to make the interchange work. Next we discuss more flexible means of portlet cooperation.

Brokered cooperation

Brokered cooperation allows independently developed portlets to exchange information. Portlets register their intent to cooperate with a broker, which facilitates the exchanges at runtime. The broker works by matching data types between the sources in one portlet and the actions of another portlet. When the types match, an exchange is possible. Portlets that exchange data in response to a user action are called “Click to Action” portlets.

The objective of the Click to Action portlets is to increase the productivity of portal users working with multiple portlets by enabling them to send information easily from one portlet to another. For example, users can click on information that is displayed in one portlet and transfer that information to another portlet. The portlet receiving the information processes it and updates its display.

Click to Action automatically matches the portlet information sources and possible actions based on their data type compatibility. Click to Action does not rely on drag and drop or other non-standard browser features. A unique advantage of Click to Action is that it is designed to work in different browsers, making it more accessible to users.

In WebSphere Portal 4.2, Click to Action is limited to Portlets that reside on the same page.

Example of a place with portlet cooperation

Figure 1-7 is an example of a place that incorporates portlet cooperation.

The screenshot displays a workplace interface with four portlets:

- Employee Directory:** A search form with fields for First Name, Last Name, and Employee #. Below it is a table of employees:

#	First Name	Last Name	Emp.#	Phone
1	Peter	Proman	4786	-272
2	Peter	Race	9555	-322
3	Peter	Cameron	9876	-456
4	Peter	Meier	2453	-380
5	Peter	Trabert	3214	-284
6	Peter	Dates	2761	-72
7	Peter	Miller	3844	-321

Below the table, it says "1 2 next" and "12 Employees".

- Employee Detail:** Shows information for Peter Proman, including a photo, general information (First Name, Last Name, Birthday, Room, Employee #), communication details (Office, Mobile, E-Mail, Private), and company information (Division, VS, PE, PK, Abbreviation, Join, W. hours, Employment, End, Price Cat.).
- Skills:** Shows a list of skills for Peter Proman, including Databases (DB/2, SQL Server, DB2, Lotus Notes, Oracle), Management Skills (Project Management, Coaching, Presentation), and Comments (Peter Proman is taking the lead in delivering enterprise web solutions to organizations around the world. Our products and services drive the way the industry thinks about web-based business practice for Domino).
- Projects:** Shows a list of projects, categorized into Open Projects and Closed Projects. The Open Projects list includes:

Nr.	Title	Division	Start	End
310-8282	Developing Intranet „Wondergo AG“	AIS	01/01/99	03/30/99
320-9393	Consulting Rice Braterhouser Woopers	AIS	01/01/99	03/30/99
320-8585	Portal Prototyp Gaman AG	AIS	01/01/99	03/30/99
320-6732	Internet Solution E-Business AG	AIS	01/01/99	03/30/99
320-7654	Internal Project: Building a Employee Portal	AIS	01/01/99	03/30/99

The Closed Projects list includes:

210-9882	Price Sheet „Cookie Cutter Machine 2003“	BSS	01/01/99	03/30/99
218-7221	Presentation Marketing Database	BSS	01/01/99	03/30/99
310-6666	Employment Information System „Army“	BSS	01/01/99	03/30/99
456-72882	Management-Information-System Gayer AG	EMWS	01/01/99	03/30/99

Figure 1-7 Example of a workplace with portlet cooperation

The example shows an HR workplace. In the upper left corner we have placed an employee directory. The other portlets show employee details (top right), skills (bottom left) and the projects (bottom right) the employee is currently working on. The skills and project information are coming from separate Domino databases. In previous times they were all developed and used separately. Now that we are moving to a workplace interface, we want to use the Portlet cooperation features to present the information in a new context.

If we search for an employee in the employee directory and click on the person entry we are looking for, we can have the other portlets react to our selection. The employee's detail information and skills, as well as the projects he is

currently working on, will be shown in the other portlets. All this is triggered by a simple click on the employee link.

This example illustrates why portlet cooperation is a great concept that helps us to build new kind of applications, just by exposing old Domino applications inside the portal.

1.3.4 The portalizing process

Before we talk about *portalizing*, we first need to establish a common understanding of what this means.

Figure 1-8 presents a high-level outline of the overall portalizing process.

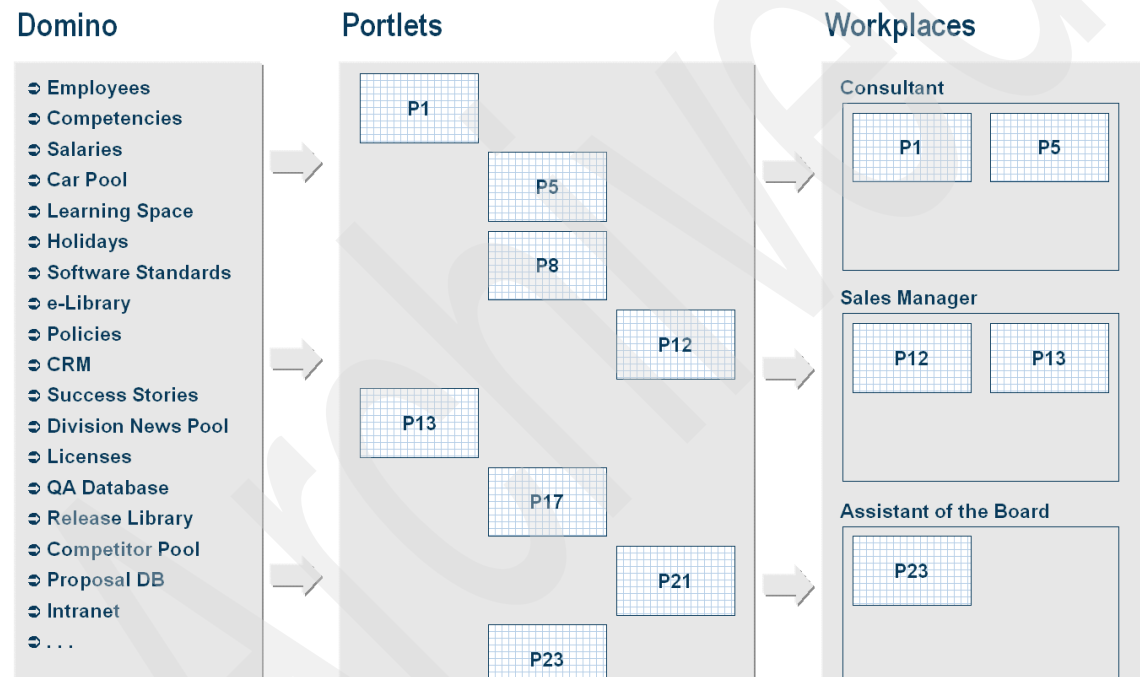


Figure 1-8 From Domino applications to portlets and workplaces

On the left is a list of typical Domino applications. They need to be portalized into portlets, which can then be made available for different workplaces.

A Domino application can be represented by a single portlet, but there is usually not a one-to-one relationship between applications and portlets. Most cases exhibit a one-to-many relationship. Portal applications are usually made up of multiple portlets that use portlet cooperation. This way, application functionality exposed in portlets can be combined in many different ways, sometimes allowing

the user to put information in a context that not even the developer of the individual portlet has thought about.

It is also important to understand that the same portlet can be used in different workplace contexts with different user roles.

1.3.5 The portalizing challenge

Looking at the portalizing process raises the following questions:

- ▶ What portlet development techniques are available?
- ▶ What is the effort required for portlet development?
- ▶ What functionalities can be implemented within a portal?
- ▶ How can scalability and performance be ensured?

In the following discussion we address these questions and describe the factors and dependencies of the portalization process.

Figure 1-9 illustrates the relevant factors.

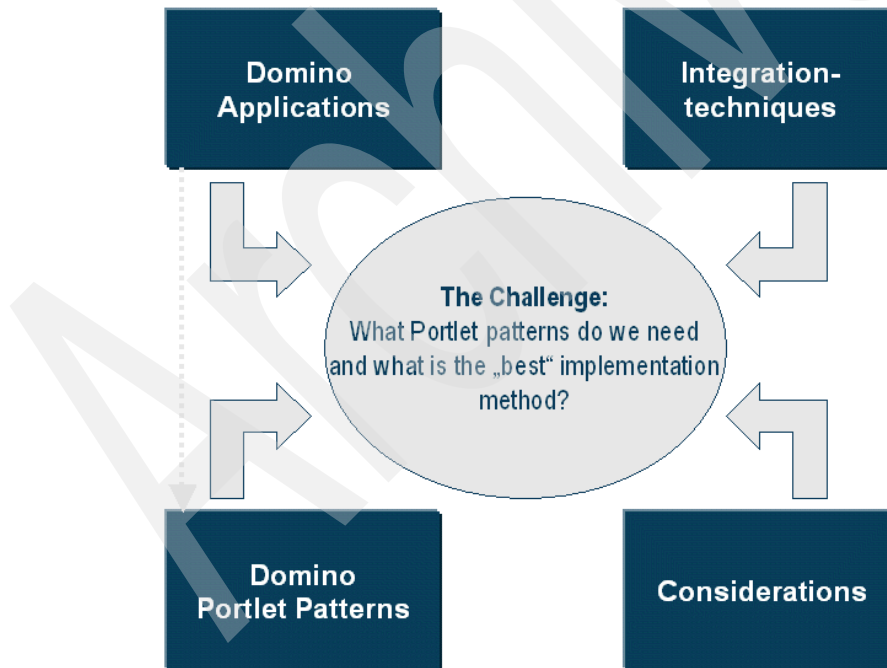


Figure 1-9 Factors and dependencies

As with any development project, there are multiple factors and dependencies that influence a portalizing project. The important parameters for a Domino portlet project are:

- ▶ Domino application: the characteristics of the Domino application.
- ▶ Integration techniques: knowledge of all the different options.
- ▶ Domino portlet pattern: defines the portal integration depth and user interface requirements for the portlet. This is discussed in greater detail later in this chapter.
- ▶ Considerations: overall project parameters like resources, skills, and so forth.

You need to develop a good understanding of the different factors and their dependencies to be able to successfully portalize a Domino application into portlets. Therefore, in the following sections we drill down deeper into the various factors and dependencies.

1.3.6 Domino applications

The most important piece in the portalizing process is the Domino application itself.

You need to consider the following parameters to characterize the application:

- ▶ Number of documents
- ▶ Number of users, number of maximum users
- ▶ Usage pattern, number of concurrent users
- ▶ Application complexity
- ▶ Single or multiple database application
- ▶ Web-enabled or Notes client-based

Figure 1-10 on page 17 can help to characterize a given Domino application.

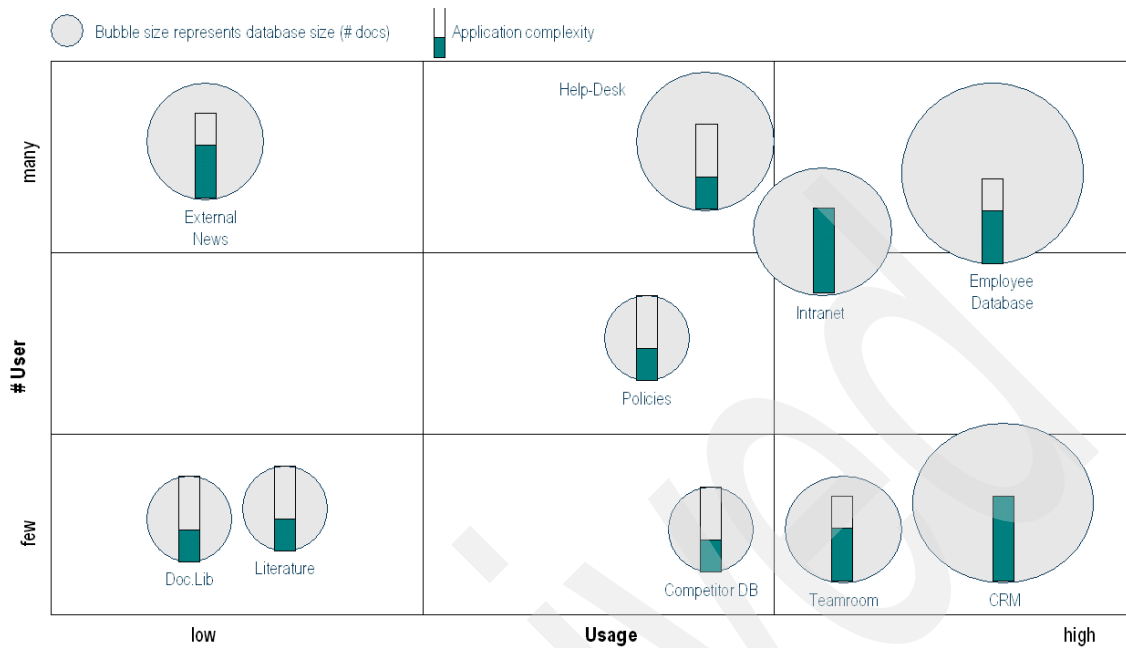


Figure 1-10 Domino application types

It shows several Domino applications, with their database usage on the horizontal axis and the number of users on the vertical axis. The size of each bubble represents the database size, and the bar in the middle indicates the application complexity.

Placing a given Domino application into this grid can help to develop a better understanding of the usage scenario for a Domino portlet.

This then allows us to estimate the workload on the portal and Domino servers.

Another important consideration is that the portal introduces new usage scenarios for your applications. With a Notes Client a user usually accesses Domino applications sequentially, meaning one at a time. On a workplace there are multiple portlets active, which will concurrently access your applications. This increases the server load significantly. In addition, the portal will broaden the user community for your applications because they are more visible now.

All this can have severe implications on your Domino backend and performance.

Performance considerations

From our experience, we know that portlet performance is the most important factor in a Domino portlet development project.

Let's imagine the following example. We are going to build an employee directory. The portlet is specified to look like this:

#	First Name	Last Name	Emp.#	Phone
1	Peter	Proman	4786	-272
2	Peter	Race	9555	-322
3	Peter	Cameron	9876	-456
4	Peter	Meier	2453	-380
5	Peter	Trabert	3214	-284
6	Peter	Daters	2761	-721
7	Peter	Miller	3844	-321

Figure 1-11 Example portlet “Employee Directory”

It has the following requirements:

- ▶ Employee directory with 100,000 entries
Initially the portlet shows the first 10 Employees of the A-name category. Users can page through the Employees like in a paper phone book, or search for an Employee by Name or Employee Number.
- ▶ Portal user community of 40,000 users
We assume a concurrency of 5% of the total portal user number for the workplace the portlet is placed on. That means we have approximately 2,000 concurrent users for the portlet.

Let's assume the following scenario:

The portlet is available on the portal home page for all users. Monday morning at 8 AM all employees are going to log on to the portal.

The interesting question now is: What is going to happen to the overall portal performance and server load?

Let's analyze: We are going to have 2,000 concurrent requests to display the initial view of the portlet. Creating 2,000 concurrent sessions to the Domino backend and rendering the first page for each user does not seem feasible.

If we planned for this scenario, we probably would have thought about caching strategies and overall session management for this use case. If not, this portlet can significantly decrease the overall portal performance and put a very high load on the Domino backend. Later in the book, we discuss caching strategies and session management in detail.

Note: Portlet performance and scalability is often underestimated in Domino portlet development projects. It is good best practice to do performance testing right from the start of the project. Performance is not only dependant on your coding or the Domino backend, but also on the overall portal/Domino infrastructure.

1.3.7 Portlet patterns

Portlets can be as simple as data display windows into existing applications, or as advanced as a replacement for complex workflow applications. They also can have a different level of integration with the portal.

Portlet *patterns* will help you to classify what type of integration level you are looking for; this information is relevant when deciding what integration technique to use.

In this section we describe the following patterns:

- ▶ Link
- ▶ Display
- ▶ Integrated
- ▶ Migrated

As for any development project, it is best practice to establish use cases for the portlet usage. We need to ask:

- ▶ What is the use case from a user's perspective?
- ▶ Does the use case vary with the role of the user?

With this information we can then decide what portlet pattern our application falls into.

Link

The *Link* type is the easiest of all integration types. It provides the user with a link that launches into the native application. This can be either a Notes-based or a browser-based application. Depending on the setup of the authentication system, the user may need to log on to the specific application. The Notes client will challenge you for your password if you have not configured it to use your

operating system login. If the application is browser-based and you did enable single sign-on (SSO), the re-authentication is not necessary.

A link type integration can be achieved by using, for example, the Bookmark Portlet or Quicklinks Portlet that ship with WebSphere Portal.

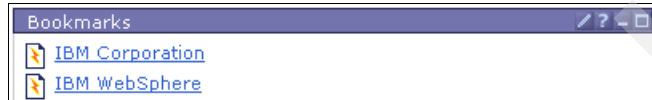
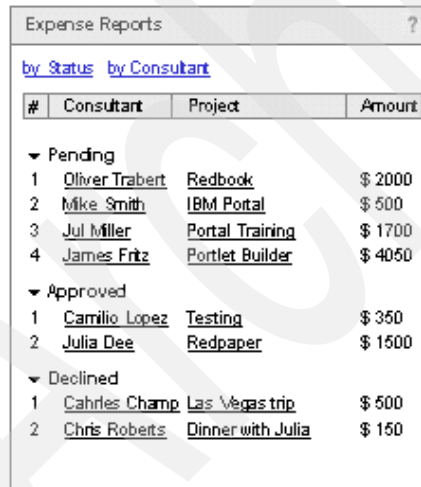


Figure 1-12 Link type example

Display

In the *display* category are portlets that only display information from Domino. If the user needs to interact with the application functionality, they must launch either into the Notes Client or a Browser interface. This can be a good option if an application is already Web-enabled. In the portal we can give the user an overview of, for instance, all the expense reports that he needs to approve, and then for the approval process launch the expense report application in a new browser window. Depending on the implementation, it would be possible to launch immediately into the expense report that the user wants to approve.

A screenshot of a web portlet titled "Expense Reports". At the top, there are two links: "by Status" and "by Consultant". Below the links is a table with four columns: "#", "Consultant", "Project", and "Amount". The table is organized into three sections: "Pending", "Approved", and "Declined", each with a dropdown arrow. The "Pending" section has 4 rows, "Approved" has 2 rows, and "Declined" has 2 rows. Each row contains a number, a consultant name, a project name, and an amount.

#	Consultant	Project	Amount
▼ Pending			
1	Oliver Trabert	Redbook	\$ 2000
2	Mike Smith	IBM Portal	\$ 500
3	Jul Miller	Portal Training	\$ 1700
4	James Fritz	Portlet Builder	\$ 4050
▼ Approved			
1	Camilio Lopez	Testing	\$ 350
2	Julia Dee	Redpaper	\$ 1500
▼ Declined			
1	Cahries Champ	Las Vegas trip	\$ 500
2	Chris Roberts	Dinner with Julia	\$ 150

Figure 1-13 Display type example

This example illustrates that it is not necessary to portalize the whole application to the portal. Sometimes a small view into the application might be enough.

Integrated

An *integrated* scenario lets the user perform tasks in other applications inside the portlet, rather than having to launch the other application.

In the previous scenario we could provide the user with a list of expense reports to approve, but to perform the approval task we had to launch into a different application. Launching into a separate application might not be a problem, but being able to perform the task inside of the portlet context enhances the user experience and the usability of the portlet.

In our example, we want to allow the user to approve or deny an expense report. In the denial case he should also be able to provide a reason.

For a different user role, we also want to allow the creation of new expense reports from within the portal. In these two use cases we have exposed only some of the functionality of the expense report application to the portal. Other workflow processing of the expenses and back-end integration with an HR system will remain untouched in the native Domino application.

This example shows the advantages of the coexistence of portlets and Domino Applications. Only the information and logic needed for certain use cases will be transferred into the portlet.

The screenshot displays two side-by-side windows. The left window, titled 'Expense Reports', contains a table with columns for '#', 'Consultant', 'Project', and 'Amount'. The reports are grouped by status: Pending, Approved, and Declined. A dashed arrow points from the 'Redbook' project in the Pending group to the right window. The right window, titled 'Expense Detail', shows the details for the selected report, including the project name, submitter, submitted date, amount, and a description. It also features an 'expenses1.xls' attachment icon.

#	Consultant	Project	Amount
▼ Pending			
1	Oliver Trabert	Redbook	\$ 2000
2	Mike Smith	IBM Portal	\$ 500
3	Jul Miller	Portal Training	\$ 1700
4	James Fritz	Portlet Builder	\$ 4050
▼ Approved			
1	Camillo Lopez	Testing	\$ 350
2	Julia Dee	Redpaper	\$ 1500
▼ Declined			
1	Cahrlies Champ	Las Vegas trip	\$ 500
2	Chris Roberts	Dinner with Julia	\$ 150

Expense Detail

[Approve](#) [Deny](#) [Comment](#)

Project: Working on the Redbook team

Submitter: Oliver Trabert Approver: Peter Proman

Submitted: 11/22/02 Status: Pending

Amount: \$ 2000

Description

I have been in Cambridge working in the Redbook project.
The attached spreadsheet shows the details for my expenses.

expenses1.xls

Figure 1-14 Integrated type example

Migrated

A *migrated* portlet is one that is used to replace an application and transform entire business processes into portlets. Usually an approach like this would involve a redesign of the application.

1.3.8 Considerations for the portlet design

In this section we outline a few design issues that are important to consider the requirement gathering process for the two portlet patterns.

We want to try to reuse as much of the Domino application functionality and code as possible. The reusability will greatly depend on the architecture of the original Domino application. If it is a Web-enabled application the chances for reuse are much higher.

What can be reused?

Usually you can reuse:

- ▶ Views
- ▶ Agents
- ▶ Forms
- ▶ Field validations

The options for application reuse vary with the chosen integration technique, and are discussed in detail in the corresponding chapters.

Figure 1-15 shows a portlet user interface that we are using to illustrate some common requirements for a Domino Portlet.

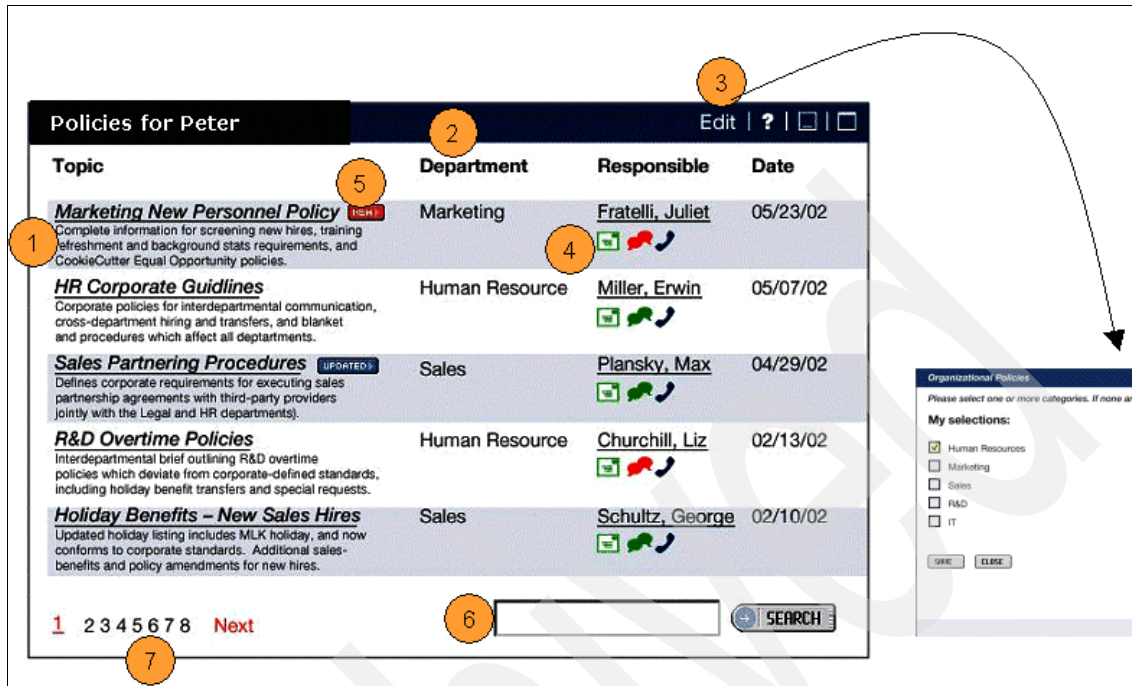


Figure 1-15 Elements of a Portlet

Layout and functional aspects

Key elements of the illustrated portlet are numbered, and have the following meanings:

1. Layout definition for views and forms
2. Retrieving and aggregating documents from multiple databases

Let's imagine that we have a product manager. He works with lots of departments; for instance, Marketing, Sales, Research and Development. Over time all these different departments have built their own policy databases. To find a particular policy, our product manager has to look into every single database. In the portlet context we now want to consolidate all the different policy databases into one portlet.

3. Edit-Mode: Implementing user personalization

This can have two different aspects. Since a workplace is tailored for a specific user role, we want to deliver only the subset of information that is relevant for a particular user to do his or her work. In addition, we want to allow the user to set a filter to further refine the information that is delivered to them.

4. Sametime integration
5. Advanced layout (buttons, alternating line colors, and so forth)
6. Search functionality
7. Pagination

Additional functionalities that may be needed in your portlet include:

- ▶ Categorization of data
- ▶ Write access to enable editing of existing data or creation of new documents
- ▶ Transactional handling, for example, calling Agents
- ▶ Search - full text or field level
- ▶ Rich Text Support for display and editing
- ▶ Page flow between different input screens/workflow
- ▶ Portlet cooperation

Considerations in choosing a portalizing type

Portlets can have view and form-like display, page flows (workflow), page navigation, read and write access, transactions (for example, using agents), and portlet cooperation.

Several levels of integration are possible. Each has a different degree of integration with the portal.

To decide on the portalizing type for a particular portlet, the answers to the following questions can be helpful:

- ▶ What functionality do you want to include/expose?
- ▶ What functionality do you need in the portlet?
- ▶ Do users perform transactions?
- ▶ Do you need simple write access to single fields or have complex forms with field validation?
- ▶ Do you have a page flow within the application?
- ▶ Is the portlet a replacement for a Domino application?
- ▶ Is your Domino application Web-enabled?
- ▶ Will the portlet be used to Web-enable an application?
- ▶ Are you going to have one complex portlet or many task-oriented portlets?

It is a good practice to develop a clear understanding of the purpose and use case for a portlet before starting the development. Portlets are true applications,

and therefore it is important to have detailed requirements for the development. Using portlet patterns can help in the requirement gathering process, and to develop a common understanding with the business users.

Portlet patterns can be characterized as requirement definitions for portlets.

Portlet pattern examples

In this section we look at a few more examples for portlet patterns. We are going to start with simple ones and move forward to more complex portlets.

Display example: News portlet

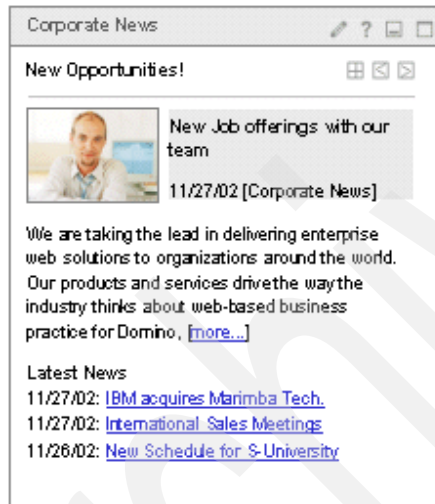


Figure 1-16 News portlet

Portlet description:

The portlet displays news from a high-volume news database.

Portlet functionalities:

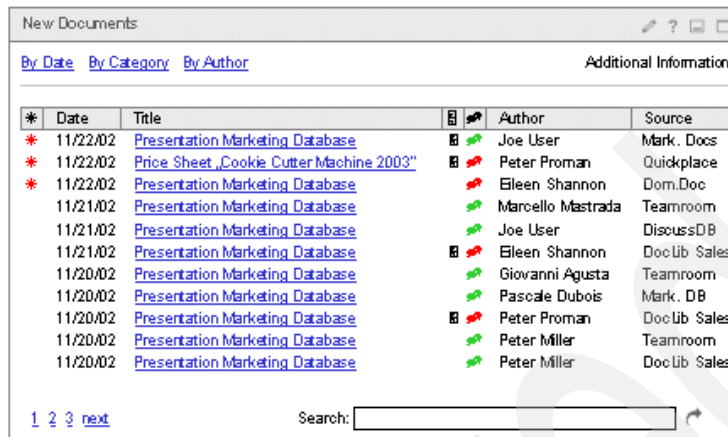
Displays data from *one* underlying Domino database.

Opens the content document in a new browser window.

Value for users:

The portlet will be a mass channel for combining internal news. As an enhancement, users can personalize which news categories they want to see.

Integrated example: New Documents



* Date	Title	Author	Source
11/22/02	Presentation Marketing Database	Joe User	Mark. Docs
11/22/02	Price Sheet „Cookie Cutter Machine 2003“	Peter Proman	Quickplace
11/22/02	Presentation Marketing Database	Eileen Shannon	Dom.Doc
11/21/02	Presentation Marketing Database	Marcello Mastrada	Teamroom
11/21/02	Presentation Marketing Database	Joe User	DiscussDB
11/21/02	Presentation Marketing Database	Eileen Shannon	DocLib Sales
11/20/02	Presentation Marketing Database	Giovanni Agusta	Teamroom
11/20/02	Presentation Marketing Database	Pascale Dubois	Mark. DB
11/20/02	Presentation Marketing Database	Peter Proman	DocLib Sales
11/20/02	Presentation Marketing Database	Peter Miller	Teamroom
11/20/02	Presentation Marketing Database	Peter Miller	DocLib Sales

Figure 1-17 New Documents portlet

Portlet description:

The portlet consolidates several enterprise knowledge databases based on Standard Document Library, Domino Doc, or Teamrooms. This is the key differentiation from the previous portlet. It adds the ability to select the view and a search option. Additionally, users have People awareness to get in touch with the author of the document.

Portlet functionalities:

- Combines data from several Domino databases

All databases have a different database structure, are not Web-enabled (only standard-view functionality is supported) and have high data volumes.

Tools for merging the different Domino databases into one consolidated database are available, including the Lotus Enterprise Integrator® (LEI).

- Searches inside of the consolidated database
- Content is opened in a form-like layout *embedded* in the portlet
- Form display also includes Rich Text fields

Displaying Rich Text inside of a portlet requires special attention. This is covered in detail in a later chapter.

- Allows the creation of new documents
- Has integrated people awareness

Value:

Users never miss published documents again. All relevant information and documents will be available in one portlet.

Integrated example: Policies portlet with write access and transactions

The image shows two views of the 'Organizational Policies' portlet. The top view is a list of policies, and the bottom view is a detailed view of a specific policy.

Topic	Department	Responsible	Date
Marketing New Personnel Policy <small>NEW</small> Complete information for screening new hires, training refreshment and background stats requirements, and CookieCutter Equal Opportunity policies.	Marketing	Fratelli, Juliet	05/23/02
HR Corporate Guidelines Corporate policies for interdepartmental communication, cross-department hiring and transfers, and blanket and procedures which affect all departments.	Human Resource	Miller, Erwin	05/07/02
Sales Partnering Procedures <small>UPDATED</small> Defines corporate requirements for executing sales partnership agreements with third-party providers (jointly with the Legal and HR departments).	Sales	Plansky, Max	04/29/02
R&D Overtime Policies Interdepartmental brief outlining R&D overtime policies which deviate from corporate-defined standards, including holiday benefit transfers and special requests.	Human Resource	Churchill, Liz	02/13/02
Holiday Benefits – New Sales Hires Updated holiday listing includes MLK holiday, and now conforms to corporate standards. Additional sales-benefits and policy amendments for new hires.	Sales	Schultz, George	02/10/02

Below the list is a pagination control: 1 2 3 4 5 6 7 8 Next and a search box with a SEARCH button.

The detailed view for 'Holiday Work Policy' shows:

- Topic: **Holiday Work Policy**
- Date: 01/12/01
- Department: **Human Resources**
- Responsible: **Claudia Johnson**
- Abstract: Corporate guidelines for requesting/requiring employee participation in project commitments that fall on recognized corporate holidays, including pay and benefits factors.
- Content: Working on holidays always involves special considerations. Please follow these guidelines to make sure employees are being taken care of. Everyone works hard enough as it is, and holiday overtime contributions need to be adequately compensated to prevent problems for the employee, and thereby prevent problems for the company.
 - Requisitioning development resources for critical time-sensitive projects is required VOLUNTARY compliance on the part of the resource.

Buttons for PRINT and CLOSE are visible at the bottom of the detailed view.

A separate window titled 'Organizational Policies' shows a selection interface: 'Please select one or more categories.' with 'My selections:' and checkboxes for Human Resources (checked), Marketing, Sales, R&D, and IT. It includes SAME and CLOSE buttons.

Figure 1-18 Portlet with personalization and form display

Portlet description:

The portlet displays the company's consolidated policy databases. Users have the option to select the policy categories they are interested in. The

portlet also has a built-in search function. The individual policies are opened in a form-like display and are embedded inside of the portlet. Additionally, users have *People awareness* to get in touch with subject matter experts.

The HR staff can create and modify policies from within the portlet. The HR manager who is responsible to approve policies can perform the approval process from within the portlet.

Portlet functionalities:

The portlet has the same functionality as the previous example, plus:

- Content creation and modification from within the Portlet
- Integrated approval workflow for policies

Value:

Users never miss new or updated policies. Through personalization they only see the policies that are relevant to them.

For the HR staff it is very convenient to update and create new policies. The integrated approval workflow expedites policy publishing.

Summary

These are just a few examples that should help you to develop a general understanding of portlet patterns and how they can help in the requirements gathering process.

1.3.9 Considerations

There are a number of factors that have great influence on a project. Some of these are:

- ▶ Time frame for the project and milestones
- ▶ Available people and resources
- ▶ Available or set products and versions
- ▶ Target platform, such as AIX®, Win 2000, and so forth
- ▶ Scalability and performance requirements
- ▶ Experience and development skills
- ▶ Portal/Domino infrastructure

Most conditions for a Domino integration project are not any different than for other software development projects. Therefore, you should apply the same project management and development methodology to a portalization project that you would use for other development project.

The Portal/Domino infrastructure

The Portal/Domino infrastructure is one condition that needs our special attention. It can have tremendous impact in a transformation project.

There are two aspects that are relevant in this context:

► **Server infrastructure**

Figure 1-19 shows the three portal/Domino infrastructure options that are used most often:

– **Single server configuration**

In this scenario the portal and Domino Server run on one machine. Databases from remote servers get replicated onto the local Domino server. This configuration is typical for the WebSphere Portal Express offering, as well as for Proof of Concept scenarios.

– **Domino hub configuration**

In this configuration the Domino and Portal servers are separate; a dedicated hub server is used for integration with the portal.

– **Distributed Domino configuration**

In this configuration all Domino servers hosting Domino applications are accessed directly from the portal.

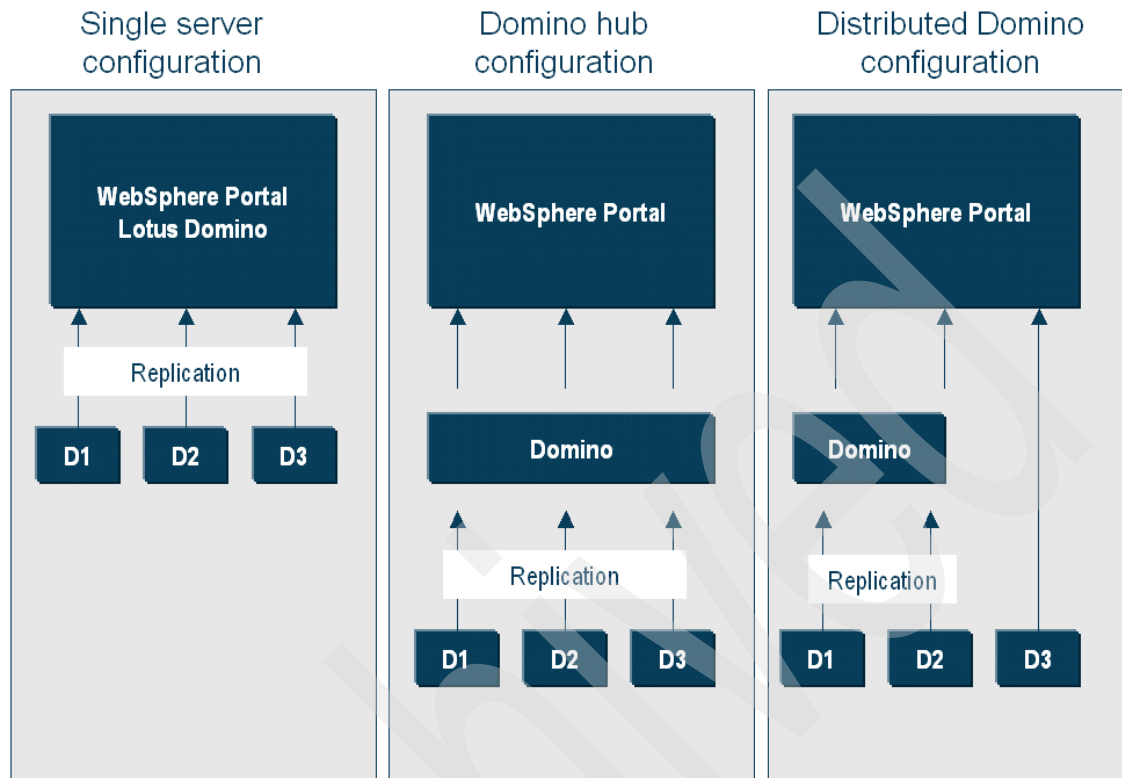


Figure 1-19 Portal/Domino Infrastructure options - D1, D2 and D3 represent Domino servers

Looking at the different infrastructure scenarios can help us to identify and avoid bottlenecks in the infrastructure.

Here are a few checkpoints from a Domino portlet point of view:

- Local Domino access is faster than remote access, but limits the scalability to one physical server
- Having a dedicated server to host the Domino applications for the portal makes the performance more predictable and scalable, but introducing a Domino cluster makes the programming more difficult. You then need to handle failover and load-balancing yourself. It is important to ensure a fast network connection between the Portal and Domino hub server.
- Accessing multiple Domino servers helps with balancing the load, but you need to check whether the Domino servers can handle the extra load from the portlet. Depending on the access method you choose, you might have to load the HTTP and/or IIOP task on the Domino server. Bandwidth constraints can also be problematic for portlet performance.

► Single sign-on

The portal server provides comprehensive single sign-on (SSO) support. Users want to be able to log on only once, and be known to the different parts of the portal server with the same consistent user credentials. Users should not be asked to do multiple logons simply because they access different portal applications.

The portal server supports single sign-on realms using WebSphere Application Server as well as authentication proxies. This means that the user needs to log on only once to gain access to all enterprise applications that are installed within the single sign-on realm.

With Domino there are two single sign-on options that you can use:

– LTPA Token authentication

This option requires that the WebSphere and Domino server are configured to share an LTPA session cookie. Using this method has the least impact on your portlet coding since the sign-on process is handled transparently for you.

For configuration information refer to the Portal InfoCenter.

– Active and passive credential vault

In the case that, for example, the portal server and the Domino server use different directories or are in different DNS domains, it is necessary to handle authentication manually. You need to get the user credentials either out of the current portal session or out of a slot in the credential vault, and then use these user credentials for the authentication with the Domino server.

For further information refer to the Portal InfoCenter.

1.4 Portal architecture considerations

In this section we discuss certain aspects of the Portal architecture that will be important to portalizing Domino applications.

For further information on the Portal architecture refer to the Portal InfoCenter.

1.4.1 Page aggregation concept

The portal should provide the user with a consistent view of portal applications and allow the user to define specific sets of applications which are presented to the user in a single context. Depending on the device of the user, the rendering of this application set has to vary to fulfill the requirements of the device. Consider, for example, a set of applications that include News, Stocks, Weather, and

Search, which have to be rendered to a conventional phone using voice interactions, a WML device with a limited display and keyboard, or a PC-based browser. The tasks of the aggregation, which are repeated with each request coming from the device, are:

- ▶ Gather information about the user, the device, and the selected language.
- ▶ Select the active portlets from the set of applications to which the user has access.
- ▶ Aggregate the output of the active portlets into a coherent, usable display.

Once the active page is determined, the layout of this page has to be used to aggregate the content of the defined applications, arrange the output, and integrate everything into a complete page. WebSphere Portal provides fully dynamic aggregation of pages from page descriptors held in the portal database.

Rendering of page components is done using JSPs, images, style sheets, and other resources. These resources are located in the file system, and are discussed in more detail in the next section. For all other resources refer to the Portal Infocenter, in the section entitled “Designing your portal.”

1.4.2 Themes and styles

The portal page is displayed using styles and themes defined by the Web designer or administrator of the portal. Themes define the look and feel of the portal overall, including colors and fonts. For example, the theme is used in the navigation bar to pick the correct color images for the corners on the page tabs. Themes make use of HTML style sheets to format the layout.

The style sheets used by WebSphere Portal contain classes that can be used by portlets to ensure visual consistency between portlets on the page.

Using these classes ensures that no matter what theme has been selected, the portlet’s look and feel matches that of other portlets and the portal page.

Examine the file `Styles.css` in the `wp_root/app/wps.ear/wps.war/themes/html/` directory to determine which classes to invoke in your portlet output. The file includes comments explaining the use of each class. To find the portlet classes, look for the following comment in the style sheet:

```
/*  
/* Styles used in portlets */  
*/
```

Styles.css is the default style sheet used by the portal and portlets. HelpStyles.css is used for portlet helps.

Portal aggregation looks up the correct copy of the style sheets based on the theme, locale, and client indicated in the request. You can change the tag definitions as well as the class definitions in the CSS style sheets. However, make sure you do not delete any style sheets or remove any style classes. Portlets require these style classes for JSP output.

We want to encourage you to use the style-definition for your own portlet markup. This way you will ensure that your portlet is going to comply with the overall look and feel of the portal if it is displayed on places using different themes.

1.4.3 Page customization

The portal allows you to define the layout and content of a portal page. During page creation, you can modify the number and placement of rows and columns on a page to your specifications. You then place the portlets into the resulting containers.

Figure 1-20 gives an example of a page that has the following layout:

- ▶ A row container that contains:
 - Two column containers, each containing:
 - Two portlet containers, with a portlet in each

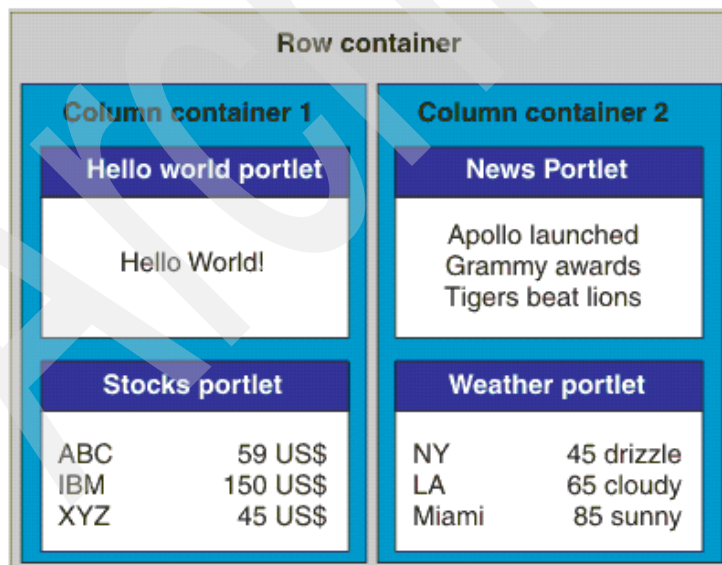


Figure 1-20 Page layout example

It is important to understand that the portal “owns” the entire HTML page. Our portlet will be placed inside of an HTML table cell, and therefore only owns the real estate within that cell.

This also means that we do not have to handle any opening or closing HTML or BODY tags. Those will be rendered by the portal page aggregator.

It is also important to keep in mind that the portlet can be displayed in column containers of different widths. Therefore we should not use absolute width settings in our own rendering. It is good practice to use relative width wherever possible.

Since portals support pervasive devices as well, we also have to consider support for other devices besides HTML browsers.

1.4.4 Using Domino LDAP with WebSphere Portal

The WebSphere Application Server uses Lightweight Third-Party Authentication (LTPA) tokens to provide single sign-on. When a user is authenticated, the portal server creates an LTPA single sign-on cookie containing the authenticated user credential. This encrypted cookie conforms to the format used by WebSphere Application Server and can be decrypted by all application servers in the shared domain, provided they all have the same cipher key. This cookie enables all servers in the cluster to access the user’s credentials without additional prompting, resulting in a seamless single sign-on experience for the user. To benefit from the LTPA method of single sign-on, the user’s browser must support cookies and have its support for session cookies enabled.

The portal server can be configured to use Domino as its LDAP directory.

The configuration process is explained in detail in the Portal Infocenter, section entitled “Configuring Domino LDAP.”

Here are just a few hints that we found useful:

- ▶ Make sure that you have created the users WPSADMIN and WPSBIND in the Domino directory. Also create the group WPSADMINS and add both users to it.
- ▶ Check that the group WPSADMINS has proper permissions and roles in the ACL of NAMES.NSF.
- ▶ Specify the correct LDAP settings in the global configuration document.
- ▶ If you plan to use the collaboration portlets, make sure that you have extended the list of fields that can be queried by anonymous users.

- ▶ When you install the portal, use the following settings for the LDAP:
 - user prefix="cn"
 - user suffix="o=yourco.com"
 - group prefix="cn"
 - group suffix=""
 - Portal administrator DN = "cn=wpsadmin,o=yourco.com"
 - Portal administrator group = "cn=wpsadmins"
- ▶ If you are installing WebSphere Application Server-Domino single sign-on, make sure you change the field for the LDAP server name in the Domino SSO document. You need to add a backslash after the colon and before the Port number.

1.5 Summary

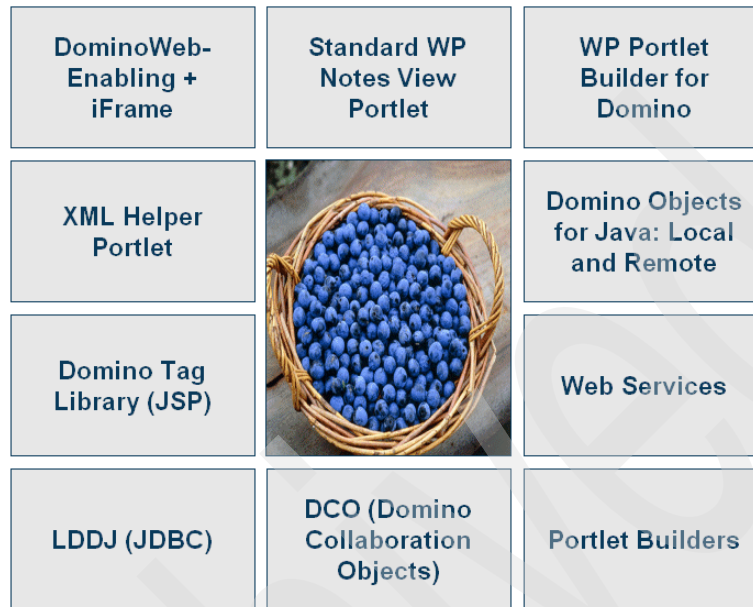
In this chapter we have shown that Domino integration into the portal is important for building workplaces. Existing Domino applications can be reused and exposed inside of the portal, allowing the display of existing information in new contexts. Bringing together the best of the Portal and Domino allows you to build a new class of hybrid applications.

We outlined a transformation methodology to achieve successful Domino-to-Portal transformation. It helps you to understand the integration challenge and to carefully choose the best integration technique and development approach for a given Domino application.

We also introduced you to basic concepts of the portal that are necessary to understand the role of the portal and the portlets in the integration process. Having a thorough understanding of the portal and its architecture is very helpful in preventing problems in the portlet development process.

In the following chapters of the book we will drill down more deeply into the vast set of Domino integration options.

Figure 1-21 *Basket of integration options*



Our goal is to brief you on all the different methods so that you can make an educated decision for your projects.



Integration techniques

This chapter provides a detailed overview of the integration techniques available for portalizing a Domino application. It also provides a guide to help you determine the best integration technique for your project.

The chapter concludes with the introduction of a fictitious Domino application case study that will be used in subsequent chapters to provide examples of how to implement each integration technique.

2.1 Choosing an integration technique

As discussed in the previous chapter, there are four main issues to consider when choosing an integration technique:

- ▶ Domino applications - the characteristics of the Domino applications
- ▶ Integration techniques - knowledge of all the different options
- ▶ Portlet pattern - concrete requirements for the portlet
- ▶ Conditions - overall project parameters like resources, skills, and so forth

Each of these issues must be addressed to ensure the successful transformation of a Domino application into a useful component of the WebSphere Portal workplace.

Each issue has many complex and potentially competing factors that must be considered. This can make it difficult to identify the integration technique that best satisfies the conditions at hand.

While every project is different, the use of a structured approach can help reduce the complexity of this decision-making process. The following sections detail a four-step approach to identify the appropriate integration technique. These steps are:

1. Pre-project preparation and training
2. Identify project requirements and considerations
3. Select the appropriate portlet pattern
4. Select the appropriate integration technique

2.1.1 Step 1: Pre-project preparation and training

Issues addressed:

- ▶ Portlet patterns
- ▶ Integration techniques

Description:

Before starting a portal integration project, obtain as much information and training as possible. Focus on understanding the portal patterns and integration techniques, and familiarize yourself with the skills and technologies required to implement each technique.

When learning new technologies, there is no substitute for hands-on training and experience. Try to implement the examples found in this book, or make up your own “case study” with which you can experiment.

2.1.2 Step 2: Identify project requirements and considerations

Issues addressed

- ▶ Domino application
- ▶ Conditions

Description

While in the requirements gathering phase of your project, obtain the Domino application characteristics as discussed in 1.3.6, “Domino applications” on page 16. It is important to have a solid understanding of the application design, size, performance, and usage patterns before you recommend a portlet pattern or integration technique.

Also, identify the miscellaneous project considerations outlined in 1.3.9, “Considerations” on page 28. The project budget, the skill set of the development team, and the application environment details (including SSO, software versions, and so forth) can each play a significant role in your final decision.

Finally, identify specific functional requirements that your portlet will be required to perform. These requirements may affect your selection of an integration technique.

Together, the Domino application characteristics, project considerations, and functional requirements define the capabilities that the selected integration technique for this project must be able to implement.

2.1.3 Step 3: Select the appropriate portlet pattern

Issues addressed:

- ▶ Portlet pattern

Description:

Use the client’s functional requirements to identify the best portlet pattern for this application. Table 2-1 on page 40 summarizes some of the main advantages and disadvantages of each portlet pattern.

Table 2-1 Portlet pattern advantages and disadvantages

Portlet pattern	Advantages	Disadvantages
Link	Quick and easy. Accesses existing applications. No modifications to existing functionality.	This is a simple Web link from a portlet. No content or functionality is available within the portal framework.
Display	Minimal enhancements to existing application functionality. Optional link outside portal to access existing application for more advanced functionality.	Minimal or no application functionality within the portal framework.
Integrated	Significant functionality within the portlet.	Requires more development time and might be more difficult to implement.
Migrated	Full application functionality within the portlet.	The most technically challenging and resource-intensive to implement.

A description of each portlet pattern can be found in 1.3.7, "Portlet patterns" on page 19.

2.1.4 Step 4: Select the appropriate integration technique

Issues addressed:

- ▶ Integration technique

Description:

Identify one or more integration techniques that will work well with the portlet pattern chosen in the previous step. Guidelines for this process are in the next section. Once the set of candidate integration techniques has been identified, compare each integration technique with the considerations identified in Step 2.

If none of the integration techniques satisfies all of the considerations for this application, identify the restricting factors that are feasible to change. While every project is different, some of the more flexible factors tend to be: estimated development time, developer skill set, and non-essential functional requirements. You may also consider a less complex portlet pattern. Some examples of factors which may be more difficult to change are: performance requirements and Domino application characteristics (database size, usage patterns, and so forth).

Once an integration technique that satisfies all project parameters has been identified, the first and most important step in the process is complete. You are now ready to design, develop, and deploy your portal application.

Considerations

Table 2-2 outlines many of the considerations that will influence your integration technique selection. In the following chapters, the discussion of each integration technique includes a brief discussion of each of these considerations. This will prove useful when comparing and contrasting the available techniques

Table 2-2 Considerations for integration technique selection

Consideration	Description
Portlet patterns	The portlet pattern or patterns most compatible with this integration technique.
Development time	Relative time/resources required to implement this integration technique.
Developer skill set	The technologies with which developers must be familiar to implement this integration technique.
Range of application functionality	The degree of control a developer has over the portlet features and functionality when implementing this technique.
Handle rich text	Can portlets implementing this technique easily display or edit the content of rich text fields?
Performance and scalability	How well do applications using this integration technique scale? Is session pooling used for Domino sessions? Can this technique be used in a clustered environment?
Requires single sign-on	Does this implementation technique require that single sign-on be enabled between WebSphere Portal and Lotus Domino servers?
Required software versions	Versions of server and development software are required to implement this technique?

2.2 Integration techniques and development options

Developers wishing to expose Domino content and functionality within a portal environment have many integration techniques at their disposal. The integration techniques presented in this book have been categorized into four main groups, called *development options*. These development options are:

1. Using existing portlets

2. Domino JSP tag library
3. Java
4. Portlet builders

The integration techniques within each development option share many similar characteristics. These characteristics include:

- ▶ The technologies and technical skill set required for implementation
- ▶ Development time
- ▶ The developer's level of control over the application functionality

This section provides a brief overview of the characteristics of each development option and outlines the integration techniques found within each option. Further details and implementation examples are provided for each option in subsequent chapters.

2.2.1 Using existing portlets

Note: Chapter 3 provides a full description of this development option.

Description

There are many general-purpose portlets provided with WebSphere Portal that can be used to access Domino applications. Other general purpose portlets are available from third-party sources. If a portlet is available that meets your Domino application's needs, this is the most expedient option.

Advantages

Using an existing portlet is by far the simplest option to implement and requires the least amount of development time. Developers do not need to be familiar with Java or WebSphere Portal, or often even Domino development technologies, to implement this option.

Limitations

The developer is limited to the functional capabilities provided by the portlet being used. Customizing the functionality of an out-of-the-box or third party portlet is not always possible.

Integration Techniques

QuickLinks portlet

The QuickLinks portlet simply stores a configurable collection of URLs and URL labels. It is very easy to configure and deploy.

Applicable portlet patterns

The QuickLinks portlet is ideal for implementing the Link portal pattern.

Web Page portlet (IFrame)

By using the <IFRAME> tag to create a nested frameset, the WebBrowser portlet allows us to embed almost any HTML content, including Web-enabled Domino applications, within the portal context. It is a quick and easy way to provide custom content within a portal page, but has all of the caveats of using framesets in general.

Applicable portlet patterns

If framesets can be tolerated, the WebBrowser portlet can be used for the Link, Display, and Hybrid patterns. The behavior of content in this portlet is essentially outside of the portal framework, and should not be used if access to portal resources or inter-portlet communication is required.

Web Clipping portlet

The Web Clipping portlet takes content from a specified URL, extracts a selected subset of this Web page, and embeds the content into a portlet. Unlike the Web Page portlet, content is rendered within the portlet context. However, this portlet does not always work as intended with existing HTML. For example, JavaScript does not always function as intended, and much of the header information is lost.

Applicable portlet patterns

Simple Link or Display portlets can be implemented using the Web Clipping portlet.

Notes View portlet

The Notes View portlet provides access to a specified view in any Domino database. It is simple to implement and can be applied to any application. However, the interface cannot be customized.

Applicable portlet patterns

The Notes View portlet can be used to implement a basic Display portal pattern. However, it should not be used if any customized interface or additional functionality is required.

XML helper and RSS portlets

The Rich Site Summary (RSS) news format is a simple, common format for delivering news to portals and other Web sites. By default, the RSS Portlet renders XML data that conforms to the RSS DTD. However, it can be easily configured to render any XML data using any XSLT stylesheet.

Currently, the RSS portlet only allows for a single, static URL to be specified when the portlet is configured. This significantly reduces the usefulness of this portlet as an effective integration technique.

An XSLT stylesheet can be written to render DXL, Domino's native XML format. Or, a custom Domino view or agent can be developed that generates XML in the RSS format.

An XML/XSL helper portlet can be configured to get the XML from Domino and format it for presentation. This set of methods produces portlets by simply modifying your Domino application to serve up XML to the XML/XSL helper portlet. The style sheet used to render the view creates an HTML table out of the Domino XML (DXL) resulting from a ReadViewEntries request to the Domino database.

Applicable portlet patterns

Link and Display portlet patterns using a very small data set can be implemented using this RSS portlet. XML/XSL helper portlet can be used to implement Link and Display portlet patterns.

Using multiple existing portlets together

By combining the Web Clipping and Web Page portlets with some custom JavaScript and Domino development, it is possible to create a basic portal application which implements the integrated portlet pattern without custom Java or JSP development. The main advantage of this integration technique is that it is relatively quick and easy to implement. Its main limitation is that the developer is tied to the features provided by the portlets being used.

Applicable portlet patterns

Display portlet pattern and Integrated portlet pattern (with limited functionality)

2.2.2 Domino JSP tag libraries

Note: Chapter 4 provides a full description of this development option and its integration techniques.

If the functionality provided by existing portlets is not adequate for your Domino portal integration requirements, the Domino JSP tag library is a good option to consider. JSPs and the Domino JSP tag library provide a rich set of development options that are quicker and easier to implement than the Java development option discussed in Chapter 5.

Note: At the time of writing, the Domino JSP tag library implementation did not fully support a portal environment. Refer to the release notes of the new Domino versions to see if the support has been added.

Integration Techniques

Domino JSP Tag Libraries

JSP Tag Libraries provide developers with a quick and easy technique for giving a Web application the ability to incorporate complex Lotus Domino interactions, simply by adding custom Domino JSP tags to a JSP page. Because the WebSphere Portal framework and JSPs are both based on the J2EE environment, a developer can easily use Domino JSP tags to expose Domino data and functionality in a WebSphere Portal portlet.

A key drawback of the Domino JSP tag library is that it does not currently support session pooling for connections to Domino servers. This can result in performance limitations for heavily used portal applications.

Applicable portlet patterns

Link and display portal patterns can be implemented using the JSP tags, but are often more easily implemented using techniques described earlier in this chapter. Generally JSP tags are used when implementing integrated and migrated portlets.

A great number of options and techniques are available to the developer. Some of the options are:

- ▶ J2EE development
- ▶ JSPs
- ▶ JSP tags
- ▶ Custom Domino JSP tags
- ▶ Click to Action
- ▶ Collaborative components

2.2.3 Developing Domino portlets using Java

Note: Chapter 5 provides a full description of the Java development option and its integration techniques.

While JSP tags provide developers significantly more power than using existing portlets, developers may still require specific functionality that is difficult or not possible to implement within the JSP tag model. Pure Java coding overcomes the development limitations of other integration technologies or options.

Java is the base programming language, ultimately used by all portlets running on WebSphere Portal. You are not limited by the functionality or the user interactions provided by portlet builders or existing portlets, or by any scalability limitations. When you are creating your own Java programs, you can deal with all of these issues, as well as others.

The disadvantages of developing in Java are that extensive Java programming skills and portlet programming skills are required, and in many cases the development time can be a lot longer than for other integration techniques.

A great number of options and techniques are available to the Java developer. Some of the options are:

- ▶ The WebSphere Portlet API
- ▶ Domino Java API, handling Rich Text in Domino
- ▶ Corba mechanism and IIOp access to Domino databases
- ▶ Implementing object pooling
- ▶ Using the Struts framework with Domino and WebSphere Portal
- ▶ Using JavaBeans
- ▶ Portlet logging using the server-based log and log4j

2.2.4 Portlet builders

Note: Chapter 6 provides a full description of the Portlet Builder development option and its integration techniques.

This option covers the tools and technologies available in the marketplace that attempt to bridge the gap between the limitations of the use of existing portlets and the complexity of the custom Java development options. It focuses on the technology offerings from vendors including IBM, Bowstreet, Conet, and others.

Portlet builder technologies provide a “middle-ground” approach to portlet development. They offer significantly more development capabilities than the use of existing portlet option. In addition, they promise a shorter development time and require less in-depth knowledge about custom portlet development than the more advanced custom development options of JSPs and Java.

The portlet builders that are included in our detailed discussion in Chapter 6 are:

- ▶ IBM Portlet Builder for Domino
- ▶ Bowstreet Portlet Factory for WebSphere
- ▶ Conet Portlet Factory for Domino

Other portlet builders introduced in that chapter are:

- ▶ Apatrix Portlet Connector
- ▶ Sofor Interactive Portlet Builder for Domino

These tools vary in capability, development approach, user interface, strengths, and weaknesses.

2.3 Case study: A simple sales tracking application

The remainder of this book discusses each development option and integration technique in greater detail. To facilitate this discussion, a sample application is used throughout the book to provide examples of how to implement each technique. This sample application is available for download from the IBM Redbooks Web site. For information on how to download this software, see Appendix B, “Additional material” on page 421.

Note: This application is intended only to provide portlet migration examples. It is not intended for use in a production setting.

Case study overview

The fictitious company Widget Corp. is using Lotus Domino to support the sales department. Over the years they have built a sales tracking application to help their sales force track interactions with their customers. The databases can be accessed from a Notes client or a Web browser.

There are two main databases in the sales tracking application: the Customers database and the Sales database. Figure 2-1 on page 48 and Figure 2-2 on page 49 illustrate the user interface when accessed from Lotus Notes. A third database, Products, is used for product keywords and is not accessed directly by the sales force.

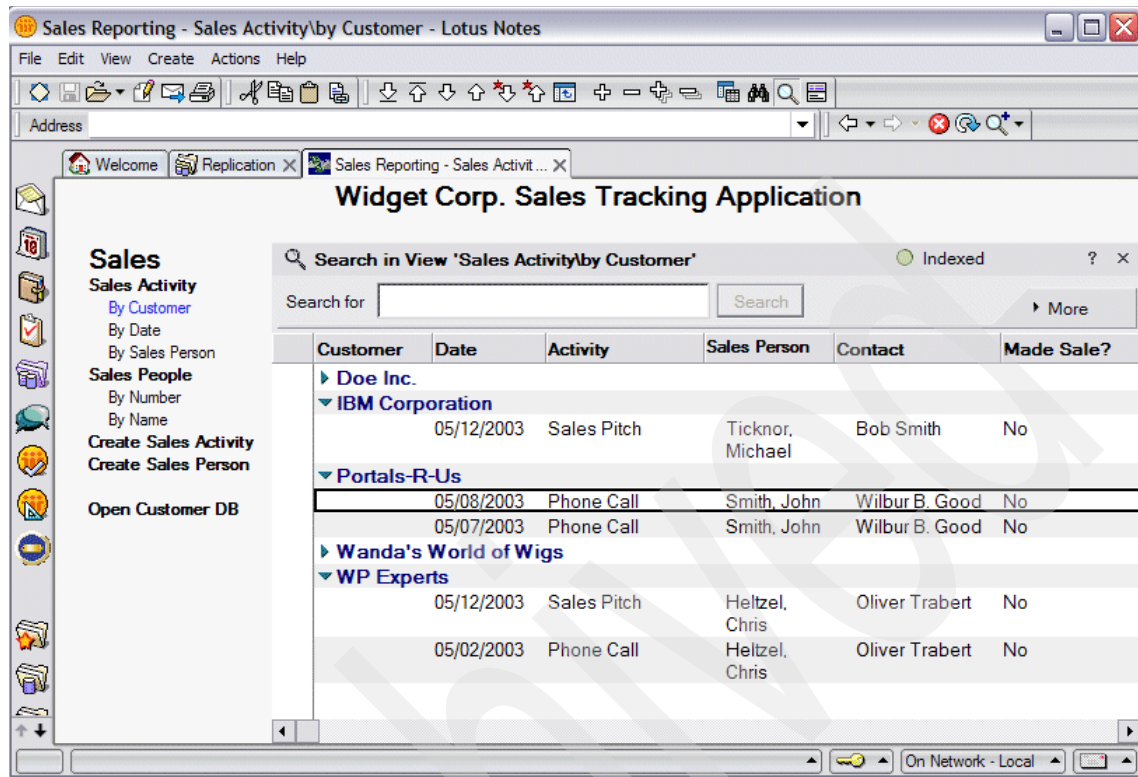


Figure 2-1 The Lotus Notes interface for the Sales Tracking application Sales DB

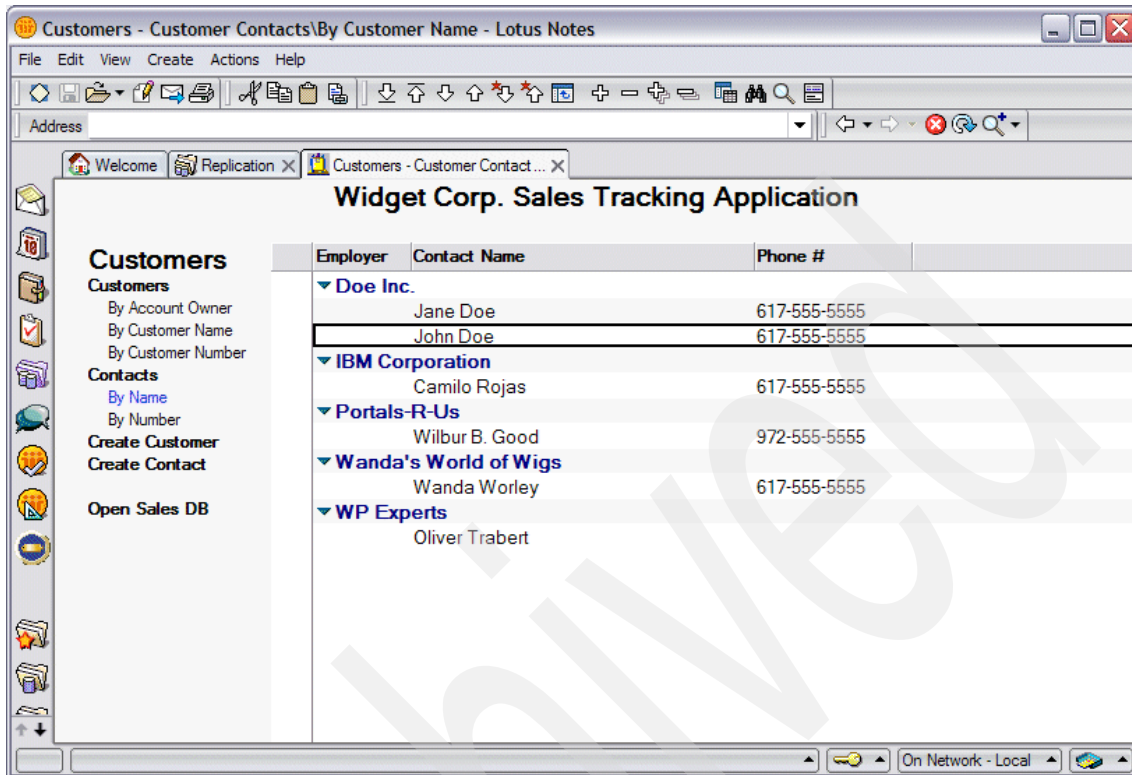


Figure 2-2 The Lotus Notes interface for the Sales Tracking application Customers DB

There are five main types of documents used by this application. They are defined in Table 2-3.

Table 2-3 Document types used by the example Sales Tracking application

Document type	Database	Description
Customer	Customers.nsf	An organization that is a customer or potential customer of Widget Corp.
Customer contact	Customers.nsf	An employee of a Widget Corp. customer who is responsible for a purchasing decision.
Product	Products.nsf	A product sold by Widget Corp.
Sales person	Sales.nsf	A member of Widget Corp.'s sales force.
Sales activity	Sales.nsf	A document tracking a specific interaction between a Sales Person and a Customer Contact.

Case study objective: Sales Workplace

Widget Corp. has decided to build a Sales Force Workplace using WebSphere Portal. On the workplace, they would like to expose the content of the Domino databases that are related to the sales process. They also want to use the portlet's cooperation functionality to create a seamless user experience. Their vision is to present the sales person all the information available on a given customer in context with the products the customer buys and the most recent sales activities with that customer. In addition, they want to use people awareness to allow better communication between the different sales people.

Figure 2-3 shows a prototype of the workplace that they want to build.

Customer Directory

Please select Customer

Name:

Location:

Customer #:

#	Company	Location	Company #
1	IBM-Lotus	Cambridge	4786
2	IBM-Tivoli	Rochester	9555
3	IBM-DB/2	Bogota	9876
4	IBM-IGS	Cologne	2453
5	IBM-ISSL	Dever	3214

1 2 [next](#) 12 Locations found

Company Detail

[Edit](#) [Delete](#)

Company: Address:

Name: IBM-Lotus 1 Roger St.
 Company #: 20030501 Cambridge, MA
 # Employees: 10000 USA

Communication Account Owner

Office: +1.617.693.1153 Name: Peter Salesman
 Fax: +1.617.693.1154 Phone: +1.617.693.1153
 Website: www.ibm.com Fax: +1.617.693.1154
 E-Mail: info@us.ibm.com Email: Pproman@widget.com

Customer contacts

#	First Name	Last Name	Phone#	-Ext.
1	David	Herbrand	617.693	1153
2	Michael	Heide	617.693	1154
3	Miker	Cameron	617.693	2267
4	Markus	Meier	617.693	8890
5	Tommi	Trabert	617.693	2330
6	Camillo	Daters	617.693	7865
7	Chris	Miller	617.693	9877
8	Tommi	Trabert	617.693	4413
9	Camillo	Daters	617.693	1234
10	Chris	Miller	617.693	7890
11	Camillo	Daters	617.693	4711
12	Chris	Miller	617.693	7865

12 contacts found

Recent Sales Activity

[By Date](#) [By Activity](#) [By Sales Rep.](#) Additional Information

* Contact Name	Title	Sales Rep.
▼ David Herbrand		
* <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	11/22/02 Price Sheet „Cookie Cutter Machine 2003“	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Price Sheet „Cookie Cutter Machine 2003“	<input checked="" type="checkbox"/> Tamara Mb Kinsey
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman
▶ Michael Heide		
<input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	11/22/02 Price Sheet „Cookie Cutter Machine 2003“	<input checked="" type="checkbox"/> Peter Proman
<input checked="" type="checkbox"/>	11/22/02 Presentation Marketing Database	<input checked="" type="checkbox"/> Peter Proman

1 2 [next](#)

Figure 2-3 Example of a customer information workplace

The Sales Workplace is made up of four portlets:

▶ Customer Directory

This portlet allows the sales person to search for a customer by name or customer number. He can also browse through the customers page by page, just as he would through a paper directory.

▶ Customer Detail

This portlet displays detailed information from a customer record, like the address or the responsible sales person, in a form-like display.

▶ Customer Contacts

The Customer Contacts portlet lists all contacts for a given customer.

▶ Recent Sales Activities

The recent sales activities are displayed in this portlet. It allows the user to switch between different views of the date. Supported views are: by date, by activity, and by sales person.

The portlets on this Sales Workplace communicate with each other. Choosing a customer record automatically updates all the other portlets. They immediately display the customer details, all the contacts at this customer, and the most recent sales activities with this customer.

Case study: Application details

The relationships between document types in this application are defined in Table 2-4.

Table 2-4 Relationships between the record types in the sample application

1st Document	Relationship	2nd Document	Key
Customer	1 to many	Customer Contact	Customer Number
Sales Person	1 to many	Customer	Employee Number
Sales Person	1 to many	Sales Activity	Employee Number
Customer Contact	1 to many	Sales Activity	Contact Name
Product	0-3 to many	Sales Activity	Product Number

User Roles

The application is designed for two types of users, identified in Table 2-5.

Table 2-5 User roles

User type	Access rights	Access method
Sales person	Author access to all databases. Rights to create Customer, Customer Contact, and Sales Activity documents. No deletion rights.	Notes client or Web browser
Application owner	Full rights to create, edit, and delete all record types.	Notes client

Use Cases

Table 2-6 summarizes the ways in which sales people interact with the system. Many of these use cases are good candidates for use within a portal Environment.

Table 2-6 Use cases for the sales person role

Activity	Use case
Document creation	Create customer document Create customer contact Create sales activity document
Document edit	Edit customer document Edit customer contact document Edit sales activity document Edit sales person document
View/Lookup/Search documents	Customers by name Customer contacts by Name Customer contacts by customer Sales people by name Sales activities by sales person Sales activities by customer Sales activities by customer contact Sales activities by date
Delete documents	None

Application owners have a superset of the rights given to sales people. Table 2-7 summarizes the additional rights application owners have to the system. While technically feasible, there is a lower business need to provide this functionality in a portal environment.

Table 2-7 Additional use cases for the application owner role

Activity	Use case
Document creation	Create sales person document Create product document
Document edit	Edit product document
View/Lookup/Search documents	Products by name Products by number
Delete documents	Delete all record types

Additional application functionality

In order to provide an example as close to the real world as possible, this application uses many development features used by the majority of Notes/Domino applications. These include:

- ▶ Keyword lists based on @DbLookups and other @Functions
- ▶ Input validation and input translation formulas
- ▶ Computed fields, hidden fields, shared fields
- ▶ Hide-when formulas based on keyword changes
- ▶ A page with embedded outline for the sales and customer databases to maintain consistent feel with cross-database navigation
- ▶ Links between related documents from a Web browser. For example, a sales activity record has links to the corresponding customer and sales person records.
- ▶ Use of \$\$Return field on document save
- ▶ Rich text for additional information
- ▶ Action buttons with hide-when formulas on forms
- ▶ Use of sections
- ▶ Use of subforms
- ▶ Categorized, sorted views
- ▶ Full-text indexed databases

Data dictionary

A complete description of the fields in each document type can be found in Appendix A, "Data dictionary for case study" on page 415.

2.4 Deploying the case study portlets

Instructions for downloading case study Domino databases and many of the portlets used throughout this book can be found in Appendix B, “Additional material” on page 421.

The Domino databases should be added to the `/apps/` subdirectory of your Domino server. Verify that you have set Anonymous & Default access to at least editor, or that you have granted known user IDs with at least editor access.

Complete the following tasks to deploy the portlets:

1. Install the portlet WAR file.
2. Create a place for the demo portlets.
3. Create pages in this place to hold the various demo portlets.
4. Add the pages to this place.
5. If needed, configure the portlets.

The remainder of this section presents the detailed steps for performing each of these tasks.

2.4.1 Install portlets

A portlet application is installed through a Web Archive (WAR) file, or you can install remote portlets via UDDI directory (Web Services portlet). The WAR file used to install the portlet application can contain multiple portlets. The install process uploads the WAR file to the server, installs portlets, adds them to the list of available portlets and activates the portlets. Once you install a portlet, it is automatically activated, but with no permissions. Use the Access Control portlet to specify which users and groups can view, edit, or manage the new portlet.

Perform the following steps:

1. From the Portal Administration tab, select Portlets → Install Portlets. Browse for the WAR file, as shown in Figure 2-4. Click Next.

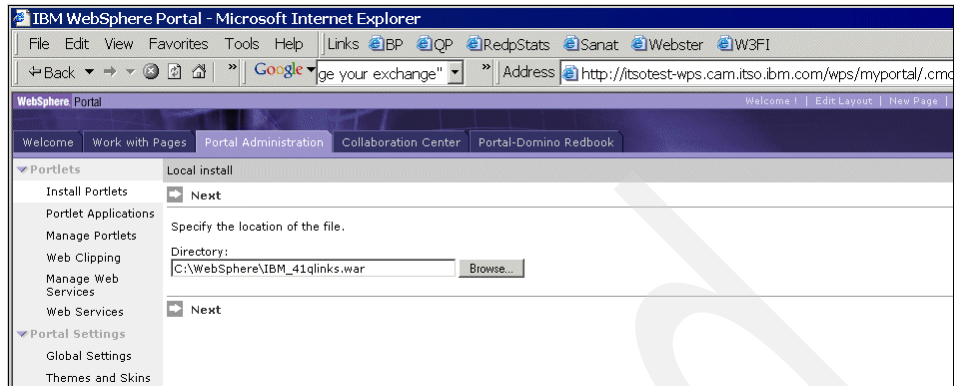


Figure 2-4 Browse for the WAR file

2. Check for the list of the Portlets included in the WAR file, as shown in Figure 2-5. In our example, QuickLinks is selected for installation. Click Install to begin the installation. You can click Cancel at any time to stop the installation process.

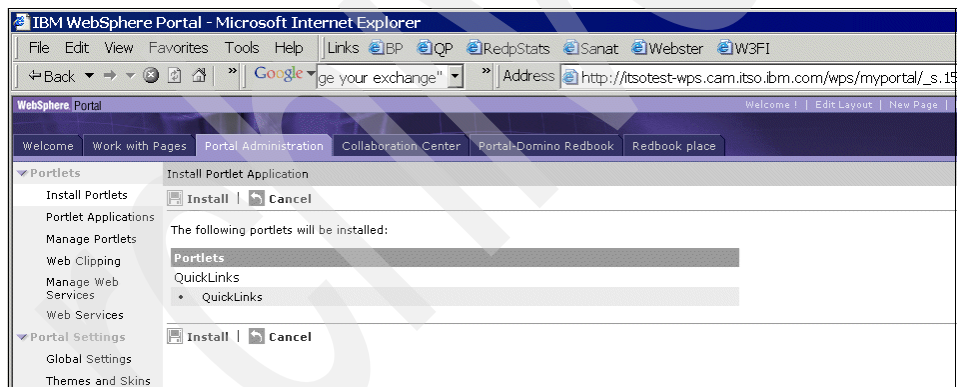


Figure 2-5 Check for the portlets that will be installed

3. When installation is complete, if successful, you should get the message Portlets Successfully Installed, as shown in Figure 2-6. Click Next if you want to install more portlets.

Tip: If portlet installation fails, check for the portal server logs directory and check the latest log file located under \WebSphere\PortalServer\logs\. The name of the log file can be determined with the append of the latest time and date stamp on it (for example, wps_2003.06.24-11.00.47.log).

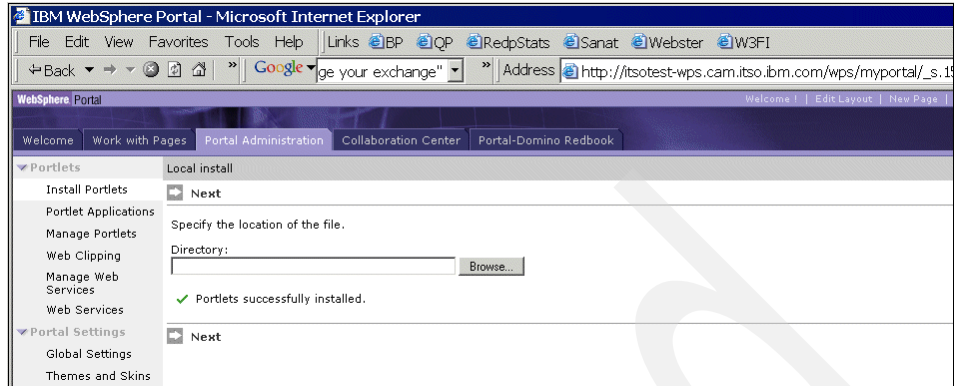


Figure 2-6 Portlet successfully installed

You have now successfully installed the portlet WAR. Portlets in this WAR are available for deployment on pages on this WebSphere Portal.

2.4.2 Creating a place

From the Portal Administration tab, you can use the Create place option to name and create a new place. To be able to use this option, you must have Create permission.

By default, any user in WebSphere Portal will have Create permission for Places and Pages.

Use the following steps to create a place:

1. Click the Create place icon in the Manage places and pages portlet. The screen shown in Figure 2-7 is displayed.

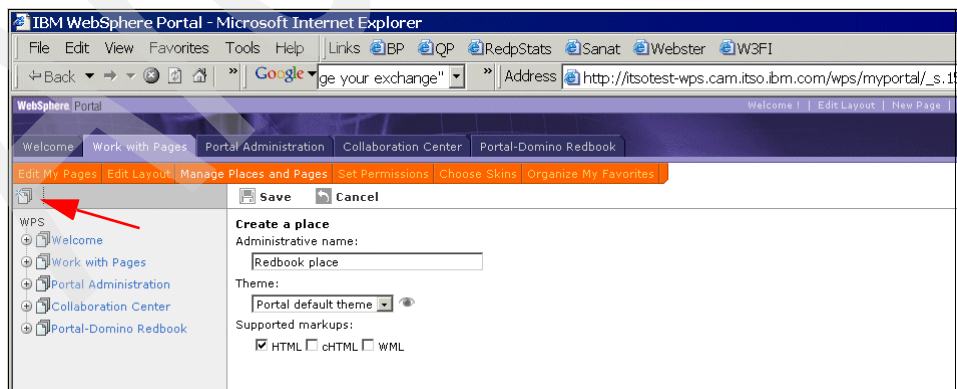


Figure 2-7 Creating a new place

2. Specify the new place name.
3. Select a theme for your place from the drop-down menu. The selected theme will appear in the theme preview on the right-hand side of the page. You can select from among various themes and, based on the preview, finalize the theme.

Note: You should *not* apply the Admin theme to a place. This theme is intended for administrative portlets and renders portlets without a title bar.

4. Make sure HTML is checked as a supported markup.
5. Click Save to create a place or Cancel to return.
6. The new place will be added to the list of places you can manage if you selected Save.

2.4.3 Creating a page

This section describes how to create a new page in an existing place. You must have Create permission to create a page in a place. Using this option, you can also reference an existing page, apply a layout, select supported markups, define a list of associated portlets, and specify locale-specific titles.

If you reference an existing page, the page name, layout, supported markups, locks, skins, and portlet list are predetermined by the referenced page. If you choose to reference an existing page, you must have Manage and Delegate permissions for the page that is referenced.

1. On the Work with Pages tab, click the Create page icon (third icon from the left).
2. The window shown in Figure 2-8 is displayed.

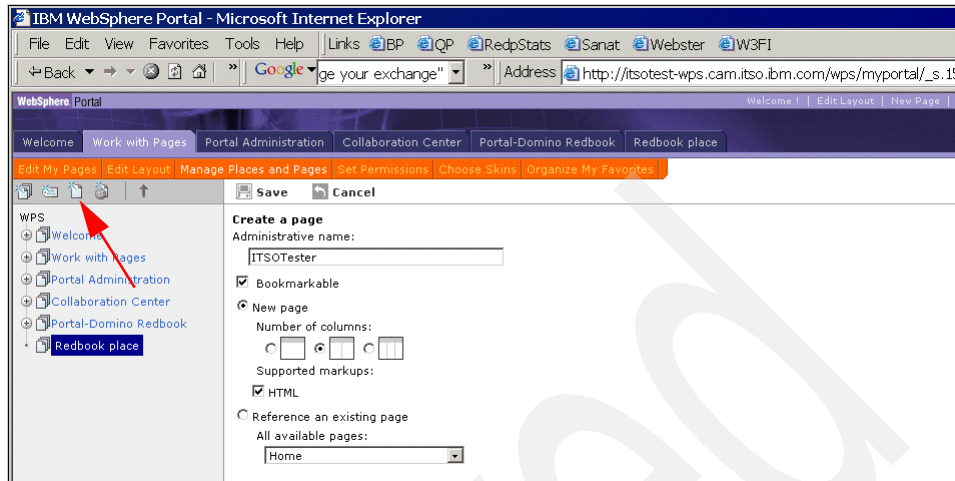


Figure 2-8 Create page options

3. Specify the administrative name. For this example, we specified the page name as ITSOTester.
4. Select a page layout you would like the page to use. The column layout can be changed later using the Edit layout and content portlet.
5. Specify the supported markup. In our example, we have only HTML as an option since we selected HTML as the only markup for our place.
6. Click Save to create a page or Cancel to return.
7. The page ITSOTESTER is added to the list of pages you can manage if you selected Save.

2.4.4 Adding portlets to a page

Open the page you want to add portlets to in edit mode using the following steps:

1. Click the Edit Layout page on the Work with pages tab.
2. Expand the place containing your page, Redbook place in our example.
3. Select the page you want to add portlets to, ITSOTester in our example.
4. Click the Add portlets icon to add the portlet to the page (Figure 2-9).

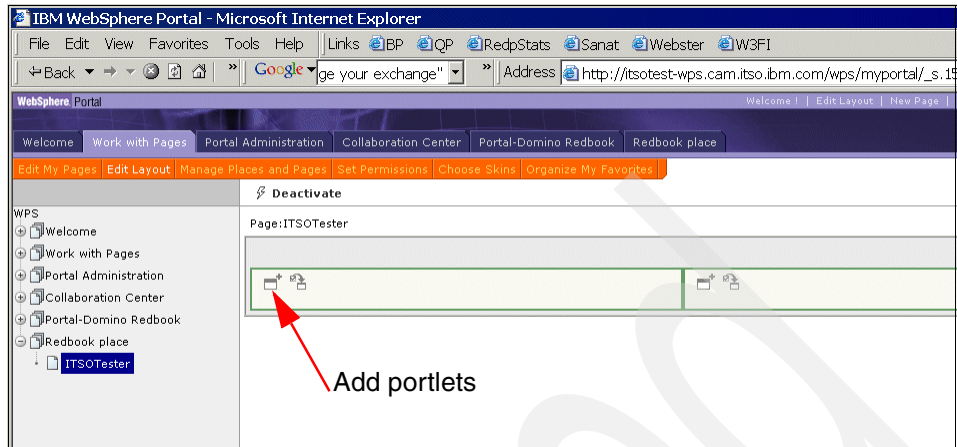


Figure 2-9 Adding portlets

5. Specify a search criteria and search for the portlets you want to add. You can search for all portlets, for a specific portlet by name, or for portlets modified on a particular date.
6. Click the Go button.
7. A list of portlets is displayed.

Note: If specific portlets are already associated with the selected page, the portlet list will be locked. Only portlets available in the portlet list can be placed on the selected page.

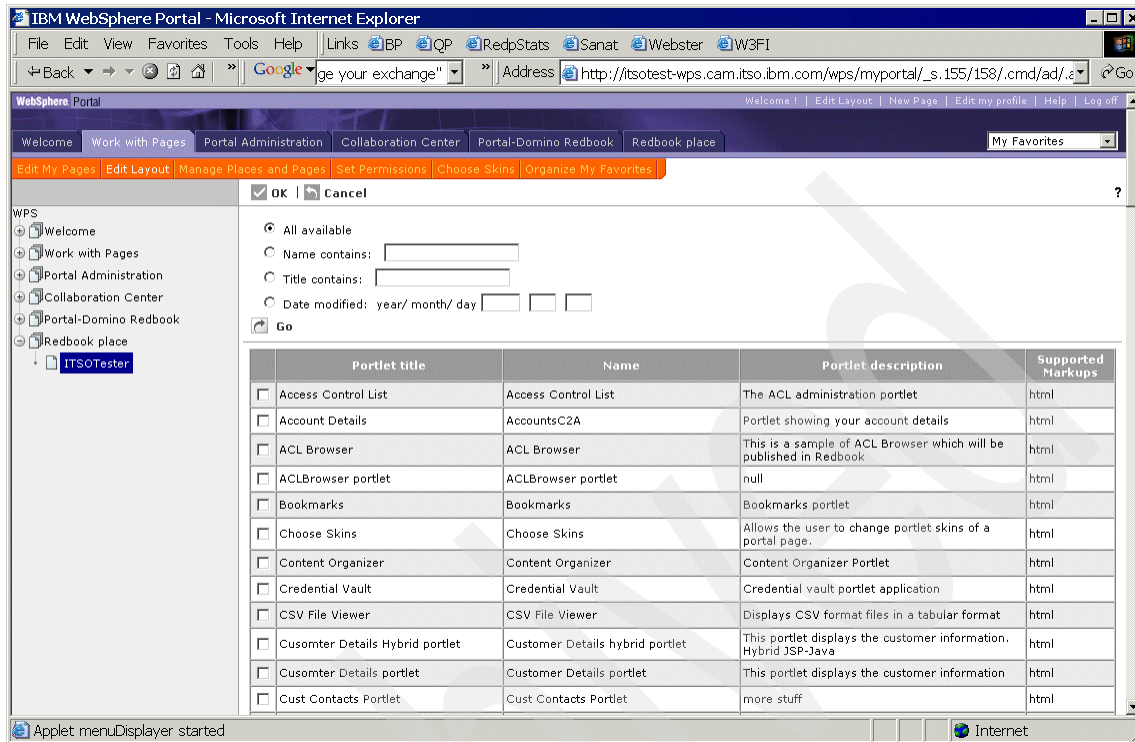


Figure 2-10 List of portlets

- Specify each portlet you wish to add to your page by selecting the check box next to the portlet name. Click OK to confirm your selection or Cancel to cancel the selection list. The portlets are added to your page if you clicked OK.

In our example, we added Welcome Portlet to the left side of the page (left column container) and World Clock to the right side (right column container), as shown in Figure 2-11.

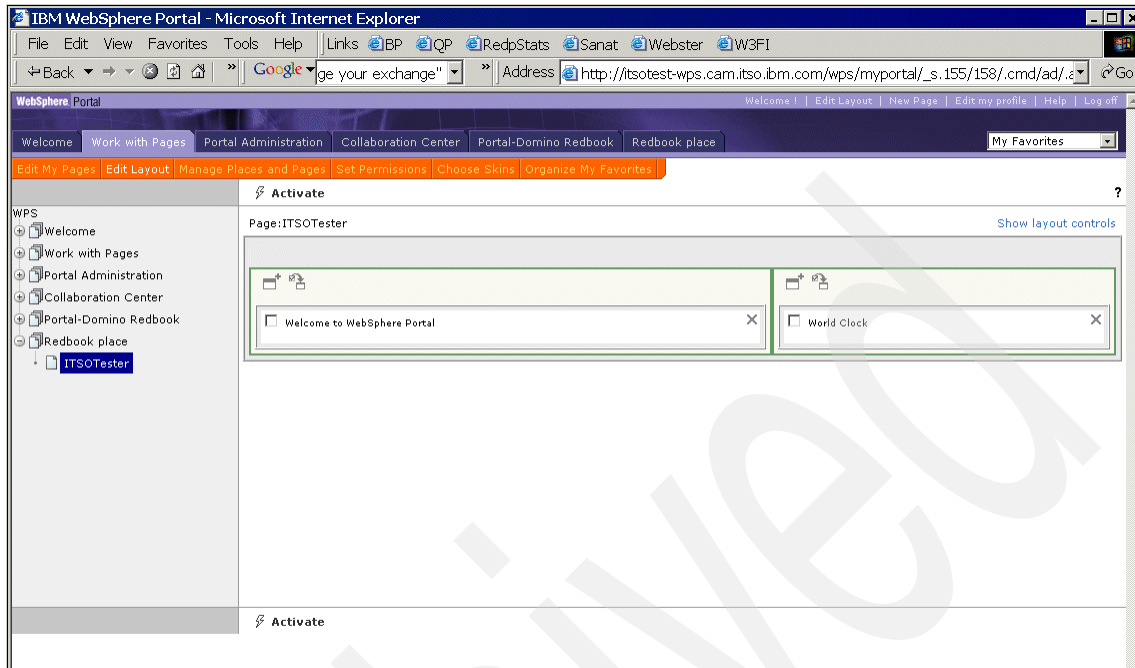


Figure 2-11 Portlets added to the ITSOTester page

When you select a page to modify, the page is deactivated to prevent users from seeing the page as you work on it. The tasks that you can perform on the page depend on the access permission.

9. Click the Activate icon to activate the page when you have finished, so that users can access it. You will notice that the page status button at the bottom of the screen will turn to Deactivate.

To test how the portlets are laid out on your page:

1. Select your place. In our example, we selected Redbook Place.
2. You will see a page, ITSOTester. This is the available page in our Redbook Place place.
3. Select the ITSOTester tab and you should see a window like that shown in Figure 2-12.

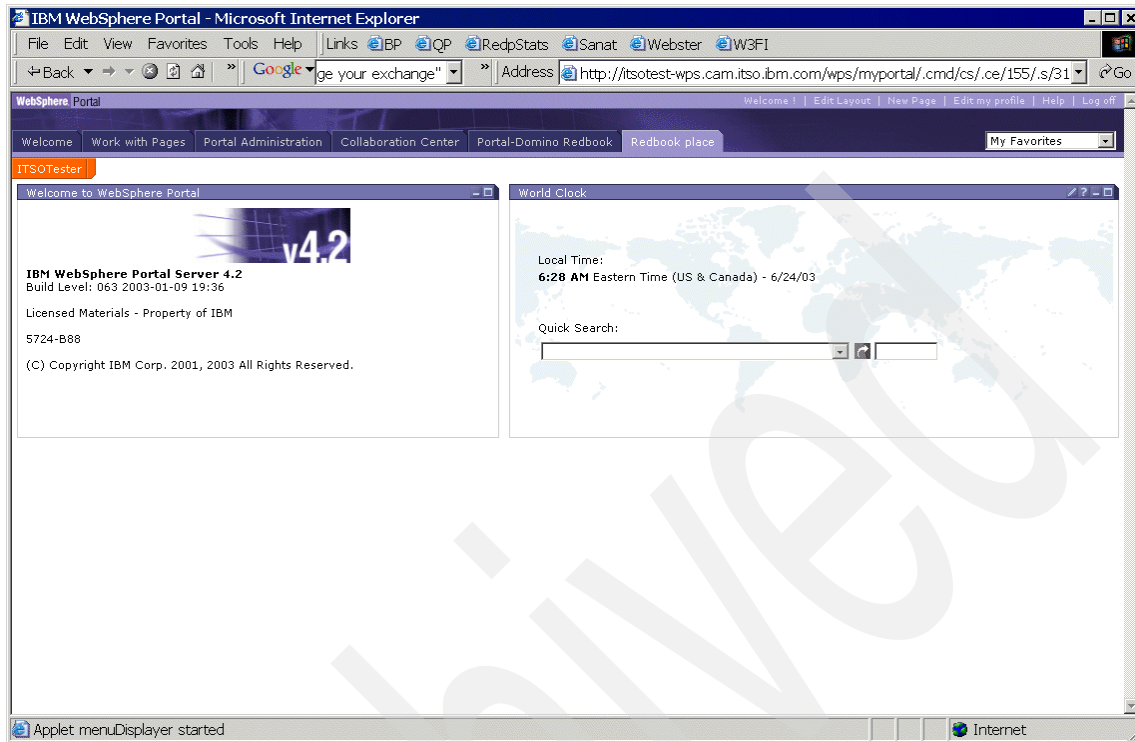


Figure 2-12 Portlets displayed on your page

The Welcome Portlet is displayed on the left side of the page and the World Clock portlet is on the right side.

Tip: If you open your page and do not see any portlet, make sure that you have activated the page. Any time you make changes on the page, you need to click Activate.

Using existing portlets

This chapter describes how to use some of the portlets that ship with WebSphere Portal to access Domino data. The portlets we have selected for this chapter utilize existing Domino applications to present information to the end user.

The outline of this chapter is as follows:

- ▶ Overview
- ▶ Why choose this option
- ▶ Technologies involved
- ▶ Software and tools used
- ▶ Integration techniques
- ▶ Reference material

3.1 Overview

This option relies on using existing prepackaged portlets to integrate existing Notes and Domino applications into the WebSphere Portal environment. Depending on the extent of your Notes and Domino applications, the techniques described in this chapter may provide a quick and effective option for integration into the portal environment. The programming skills required to use these techniques are minimal, and with some portlets, no programming skills are needed. Basic WebSphere Portal administration skills are needed. It is helpful to have Notes and Domino development skills to be able to recognize which options are beneficial, but it is not crucial.

3.1.1 Technologies involved

This option provides relatively simple techniques to integrate existing Notes and Domino applications into WebSphere Portal. There may be existing Notes and Domino applications that would not benefit from these technologies. The technologies presented involve taking Web-enabled data that is being generated in Domino and linking to or displaying this data using existing portlets in the portal environment.

3.1.2 Software and tools used

This section provides a brief description of the software and tools that are used in the implementation of the techniques included in this option. The software and tools used are as follows:

- ▶ Notes and Domino
- ▶ WebSphere Portal

Notes and Domino

A fully installed, configured, and running Domino server is a base requirement for this option. A workstation Notes client should be included because some techniques link to the supporting Notes client. Applicable versions are identified in the integration techniques section of this chapter.

WebSphere Portal

The base requirement for this option is for a fully installed, configured, and running WebSphere Portal Extend 4.2 environment, including the supporting infrastructure. This base environment will include a DB2® or Oracle database, and an LDAP authentication directory (which could be Domino), for use by WebSphere Portal. In addition, the Lotus Notes and collaborative portlets must

also be installed. A properly configured WebSphere Portal is vital for communication to the Domino server.

3.1.3 Integration techniques

This option includes techniques that enable data from Notes and Domino applications to be accessed via existing portlets that are included with WebSphere Portal. Our examples are based on an existing Notes and Domino application that resides in the apps subdirectory of our Domino server. The application includes two Web-enabled databases, Sales Reporting (Sales.nsf) and Customers (Customers.nsf), as well as an internal look-up database called Products (Products.nsf). The databases contain design elements that can be accessed through a Web browser as well as through the Notes client. The application and these databases are described in detail in Chapter 2.3, “Case study: A simple sales tracking application” on page 47.

The techniques included in this chapter are the following:

- ▶ QuickLinks portlet
- ▶ Web Page portlet
- ▶ Web Clipping portlets
- ▶ Domino portlets
- ▶ XML portlets
- ▶ Integrated portlets

For each technique, the discussion includes an overview, considerations regarding when its use is appropriate and details about the portlet, and complete implementation details, configuration options, and results.

3.2 Integrate using the QuickLinks portlet

Using the QuickLinks portlet is a very easy and powerful way to bring Notes and Domino applications into the portal environment. The QuickLinks portlet can be configured to provide links that open a Web browser window to display Web-enabled Domino data. Links can also be generated to open a Notes client on the workstation if certain conditions are met. Because this portlet is very flexible, it potentially could provide a link to any accessible database, view, or document. A close-up of the portlet is provided in Figure 3-1.

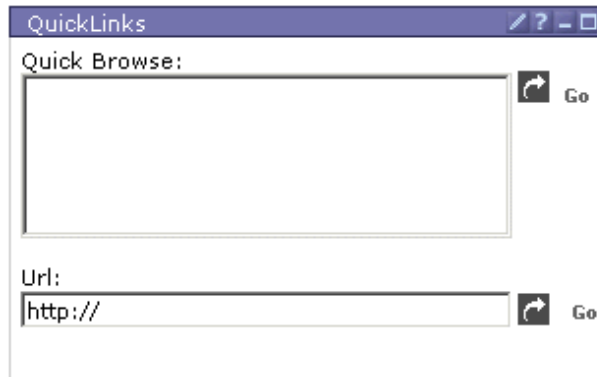


Figure 3-1 The QuickLinks portlet in display mode

The QuickLinks portlet has two ways to launch URLs to the user: the Quick Browse area and the URL area. They function as follows:

- ▶ Quick Browse area - This area provides a means to generate reusable links. To enable Quick Browse for use, the configuration mode must be entered. If you highlight a Quick Browse link and click the Go icon, you are taken to the URL associated with the link.
- ▶ URL area - If you enter a valid URL and click the Go icon, you will be taken to that URL.

Portlet listing

The QuickLinks portlet is listed in WebSphere Portal Extend using the nomenclature in Table 3-1.

Table 3-1 Portlet listing

QuickLinks Portlet	The name used in the documentation (for example, InfoCenter)
QuickLinks Portlet	Title in portlet selector (Edit Layout and Content)
QuickLinksPortlet	Name in portlet selector (installation)

3.2.1 Considerations

The QuickLinks portlet was designed to be a configurable way to store links to various Web sites. It potentially could provide a link to any accessible database, view, or document. Although implementing this technique is easy, Web site location changes, deletion of Web content, and removal of linked workstation-located applications could limit the usability of this type of portlet. A new browser or workstation session is opened when a user clicking on the links

in this portlet. There is no customization available for this portlet. Key aspects of the QuickLinks portlet are summarized in Table 3-2.

Table 3-2 QuickLinks details

Applicable portlet patterns	Link
Development time	Insignificant
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills would be useful but not required.
Range of applications	Narrow
Handle rich text	This technique launches either a Web browser or a Notes client. If a connection is made to the Domino HTTP server directly (out of the portlet context) then Rich Text is shown, otherwise no. If linking to Notes client, then this technique will handle rich text.
Performance	Limited
Session Management	No
Clustering	No
Scalability	No
Requires single sign-on	No. If Domino databases are accessed, users must have the appropriate authorization to open the database.
Required software versions	Client: Browser- This portlet supports Netscape Navigator 4.72 and higher and Internet Explorer 5.0 and higher. Notes client- The workstation should have a Notes client version of 5 or higher. Server: There are no special server requirements.

3.2.2 Implementation details

In this section we describe how to generate links and place them in the QuickLinks portlet's Quick Browse area located on a page in our portal. We also describe how to create URLs to launch a Web browser that displays Web-enable views in your databases, and how to create URLs to access views in your databases using a Notes client that you have installed on your workstation. To fully explain this implementation, we provide complete details concerning:

- ▶ Initial setup
- ▶ Configuration options
- ▶ Results

Initial setup

This example begins with accessing the QuickLinks portlet. Place a QuickLinks portlet onto a page and activate it. This process is described in detail in 2.4.4, “Adding portlets to a page” on page 58.

Open the portal page with the activated QuickLinks portlet. Figure 3-2 shows the initial portal page.

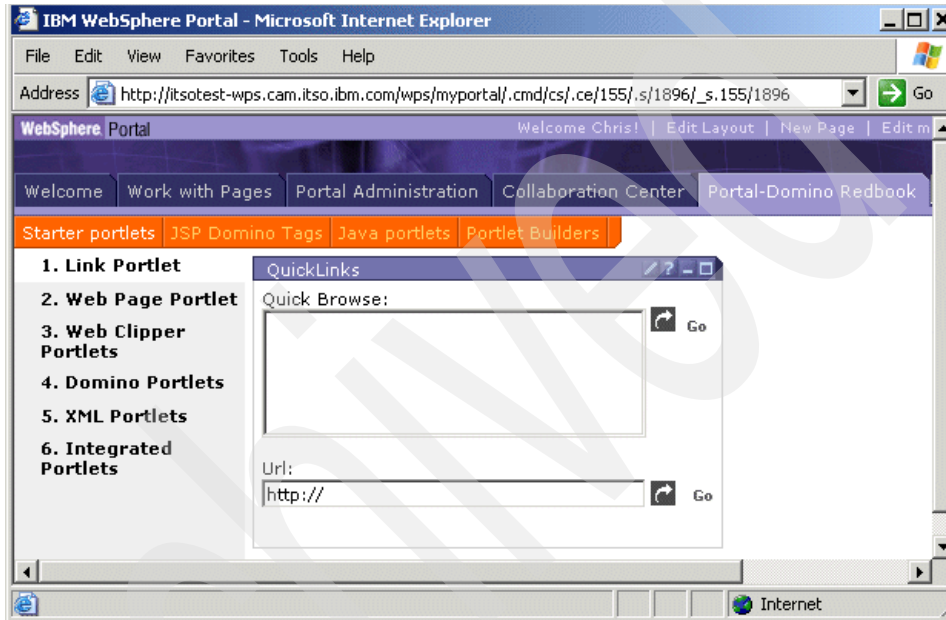


Figure 3-2 QuickLinks portlet

Configuration options

In order to show Quick Browse URLs you must complete the configuration. Enter the portlet's edit mode and complete the configuration screen. The configuration screen is shown in Figure 3-3.

Figure 3-3 QuickLinks portlet configuration for Quick Browse URLs

Click OK to exit edit mode and return to the portlet display mode.

Use the following edit options and naming suggestions to configure the QuickLinks portlet:

► Edit options (Figure 3-3):

Add When a name and URL are placed in the corresponding fields, clicking the Add icon places the name into the Quick Browse panel, while the URL is stored to be executed when a user highlights the name and clicks Go while in portal display mode.

Delete When you click on a Quick Browse entry, the entry's Name and URL are placed into the corresponding fields. Clicking the Delete icon will delete the entry from the Quick Browse list.

Import This feature allows the importation of Netscape's bookmarks.htm file or Internet Explorer's favorites listings. Click the Import icon to access a file input dialog that allows you to browse your workstation for the appropriate file.

► Name and URL considerations:

Name Choose a short name to describe the URL being entered in the URL box. Limit your entry to one line. Wrapping of the name to another line is not allowed.

URL Any valid URL can be entered. Non-valid URLs will generate an error.

Note: The Domino database Access Control List must allow for the access, or a login screen may be presented.

To link to Web-enabled Domino databases, standard Domino URLs must be used. See Table 3-3 for our examples.

To link to Domino databases using a Notes client, the Notes client must be installed on your workstation and be listed in the workstation's registry system. The format of the URL is notes://server/filepath/database/view/document. See Table 3-3 for our examples.

Table 3-3 displays the Name and URL values in our example.

Table 3-3 Configuration values for the QuickLinks portlet example

Name	URL value
Web browser Open: Customer view in customer database	http://itsotest-dom.cam.itso.ibm.com/apps/Customer.nsf/CustomersByName
Web browser Open: Sales view in the Sales database	http://itsotest-dom.cam.itso.ibm.com/apps/Sales.nsf/saByCustomer
Notes client Open: Customer view in customer database	notes://itsotest-dom.cam.itso.ibm.com/apps/Customer.nsf/CustomersByName
Notes client Open: Sales view in the Sales database	notes://itsotest-dom.cam.itso.ibm.com/apps/Sales.nsf/saByCustomer

Results

Once the Quick Browse links are configured, click OK and return to the QuickLinks display mode. A close-up of our configured QuickLinks portlet is shown in Figure 3-4.

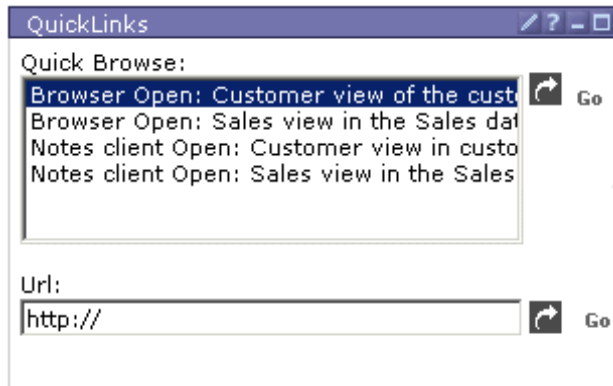


Figure 3-4 QuickLinks portlet with example links

To visit the first link, highlight the first listing and click the Go icon next to the Quick Browse area. Figure 3-5 shows the result.

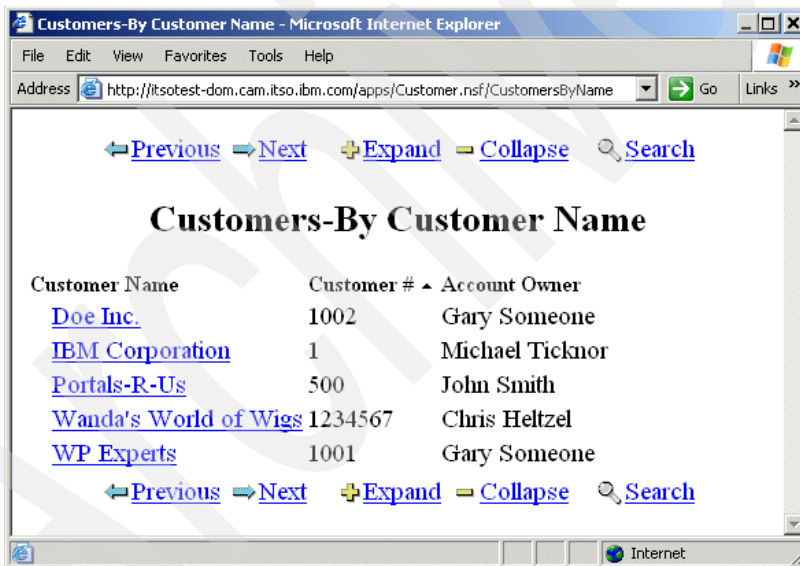


Figure 3-5 Result of clicking the first link in Quick Browse

Next, highlight the second listing and click the same Go icon. The result is shown in Figure 3-6.



Figure 3-6 Result of clicking the second link in Quick Browse area

The previous results show the appropriate Web views in new Web browser sessions. Highlighting the third listing and clicking the Go icon, a Notes client will be started if it has been installed on the workstation and configured properly. Figure 3-7 on page 73 exhibits this result.

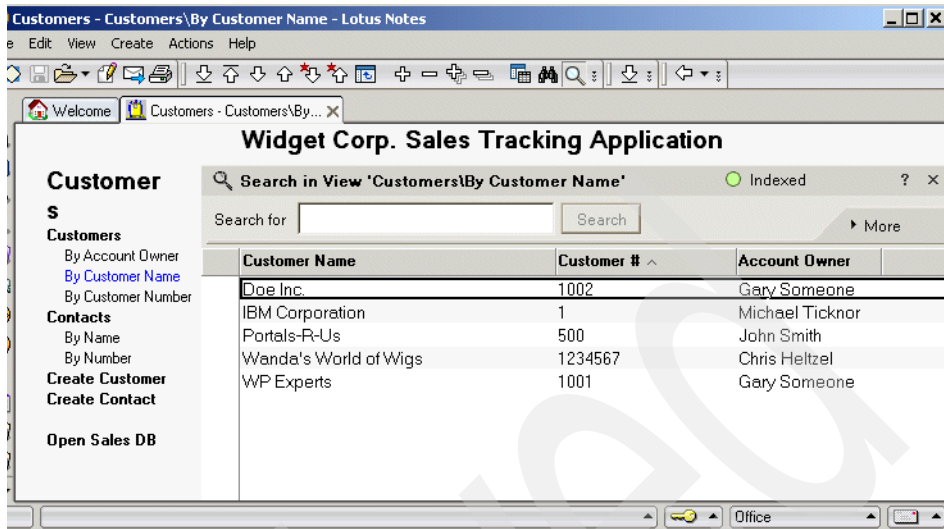


Figure 3-7 Result of clicking the third link in Quick Browse area

Finally, highlighting the fourth link opens the Notes client with the appropriate Notes view showing, as shown in Figure 3-8.

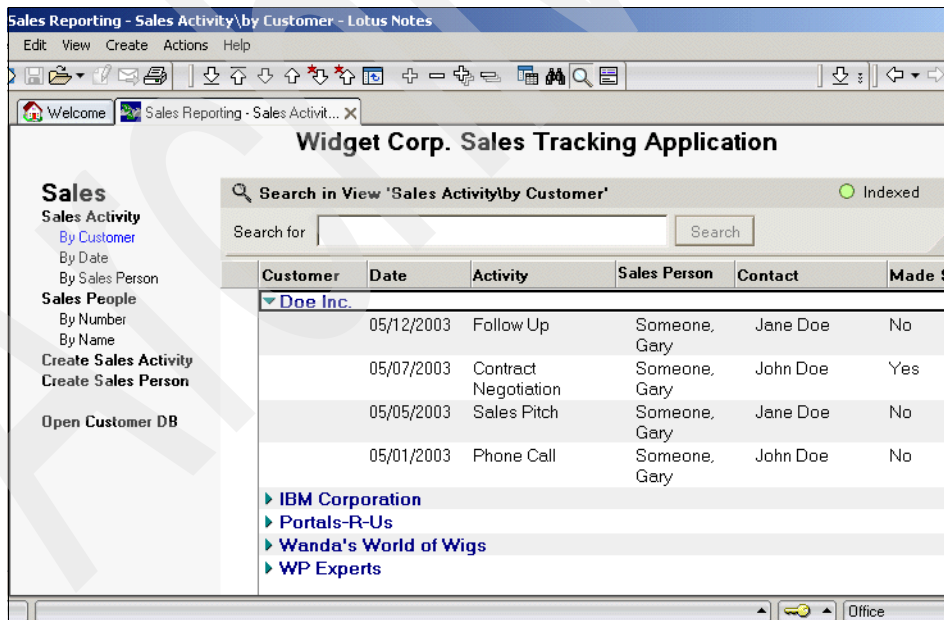


Figure 3-8 Result of clicking the fourth link in Quick Browse area

3.3 Integrate using the Web Page portlet

The Web Page portlet is a quick and easy technique to bring a Web-enabled Domino application to the portal environment. Similar to the QuickLinks portlet, the Web Page portlet can be used to configure a link to a Web-enabled view in a Domino database. The link will open the element into your portlet context. No external Web browser session will open.

Portlet listing

The Web Page portlet is listed in WebSphere Portal Extend using the nomenclature in Table 3-4.

Table 3-4 Portlet listing

Web Page Portlet	The name used in the documentation (for example, InfoCenter)
Web Page Portlet	Title in portlet selector (Edit Layout and Content)
WebPagePortlet	Name in portlet selector (installation)

3.3.1 Considerations

The Web Page portlet allows you to embed almost any HTML content, including Web-enabled Domino applications, within the portlet context. It is a quick and easy way to provide custom content within a portal page.

The Web Page portlet uses an “iFrame.” This portlet type is basically a frameset within the portal page. Scroll bars or maximizing the portlet window are often required to make the portlet usable because it is difficult to fit the existing application within the portlet confines. The navigation within the iFrame can add an additional layer of complexity to overall portal navigation.

The overall look and feel of the embedded application may not be consistent with the rest of the portal, meaning the portlet will not use portal themes. Multiple iFrame portlets do not work well on the same portal page because of different content styles and colors. Also, iFrames containing applets will not work correctly in a portal. For every request, the portal server sends down a new portal page, the old applet instance is stopped, and a new instance is built.

The Web Page portlet should not be used if access to portal resources or inter-portlet communication is required.

Applicable portlet patterns	Link, Display, Hybrid patterns
Development time	Insignificant
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills would be useful but not required.
Range of applications	Moderate
Handle rich text	No
Performance	Limited
Session Management	No
Clustering	No
Scalability	No
Requires single sign-on	Depending on the application, the Web Page portlet can be configured to use single sign-on.
Required software versions	Standard HTML browsers with JavaScript support are required. Netscape 5 and above and Internet Explorer 4 and above are supported

3.3.2 Implementation details

This section describes how to construct a link to be placed into the Web Page portlet located on a page in your portal. This link will display a Web-enabled view from one of your Domino databases. The view will be displayed within the portal context. No external Web browser session will open. To fully explain this implementation, we provide complete details concerning:

- ▶ Initial setup
- ▶ Configuration options
- ▶ Results

Initial setup

Our example begins with accessing the Web Page portlet. Install the Web Page portlet and place it onto a page. Figure 3-9 shows a page with the Web Page portlet at its initial setup. The portlet in the page has not been configured.

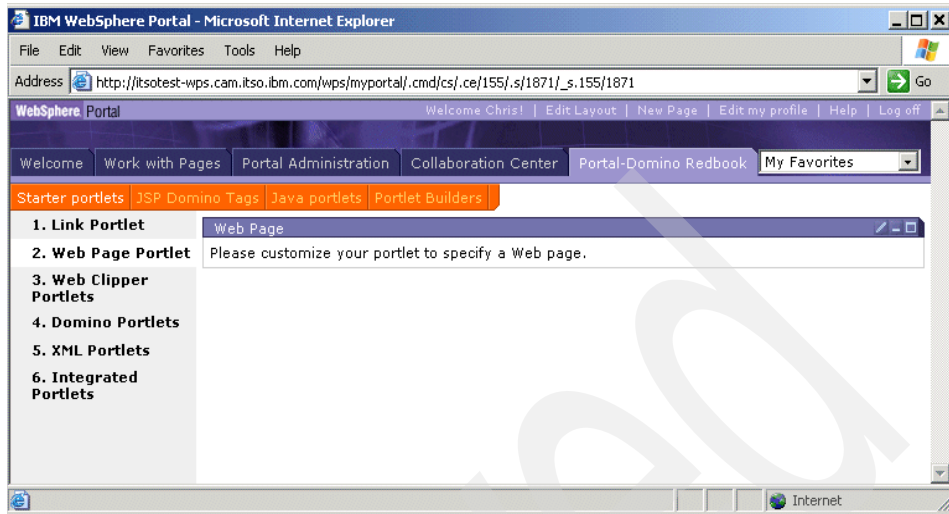


Figure 3-9 Web Page portlet before configuration

Configuration options

As in the QuickLinks portlet, the Web Page portlet has an edit mode with a configuration screen that must be completed in order to display properly. The configuration page is shown in Figure 3-10.

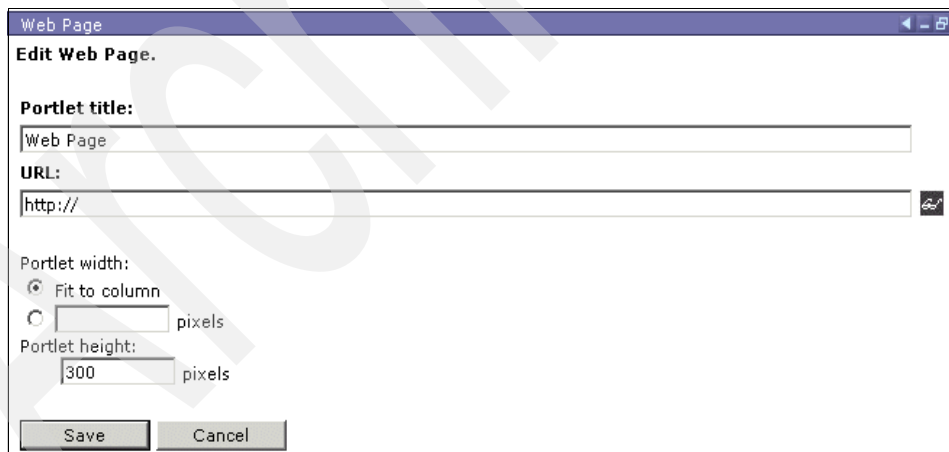


Figure 3-10 Web Page portlet configuration

Fill in the appropriate values and click Save to exit the configuration screen and return to the portlet display mode. Click Cancel to return to the portlet's display mode without saving your changes.

Use the following edit options and URL considerations to configure the Web Page portlet:

- ▶ Edit options (Figure 3-10 on page 76):

- Portlet title** The title that will be displayed in the portlet title area.
- URL** The location of the Web page that you want to display. Clicking the eyeglass icon at the end of the URL line opens a browser window to assist you in entering the URL for the Web page.
- Portlet width** This feature allows you to set the width of the browser window. The choices are Fit to column or a numerical value for the width in pixels. The default is Fit to column.
- Portlet height** This feature allows you to set the height of the browser window in pixels. The default is 300 pixels.

- ▶ URL considerations:

- URL** Any valid URL can be entered. Non-valid URLs will generate an error.

Note: The Domino database access control list must allow for the access; otherwise, a login screen may be presented.

Our example provides a link to a Web-enabled view in one of our databases. The configuration values are listed in Table 3-5.

Table 3-5 Configuration values for the Web Page portlet

Field	Inserted value
Portlet title	Web Page portlet showing the By Customer Name view of the customer database in a portlet
URL	http://itsotest-dom.cam.itso.ibm.com/apps/customer.NSF/Customers ByName
Portlet width	Fit to column
Portlet height	300

Results

The result of our configuration of the Web Page portlet is shown in Figure 3-11. The view shows inside the Web Page portlet within our portal context. The vertical scroll bar is visible because the portal height was not set high enough. Modifying this parameter higher would remove the scroll bar.

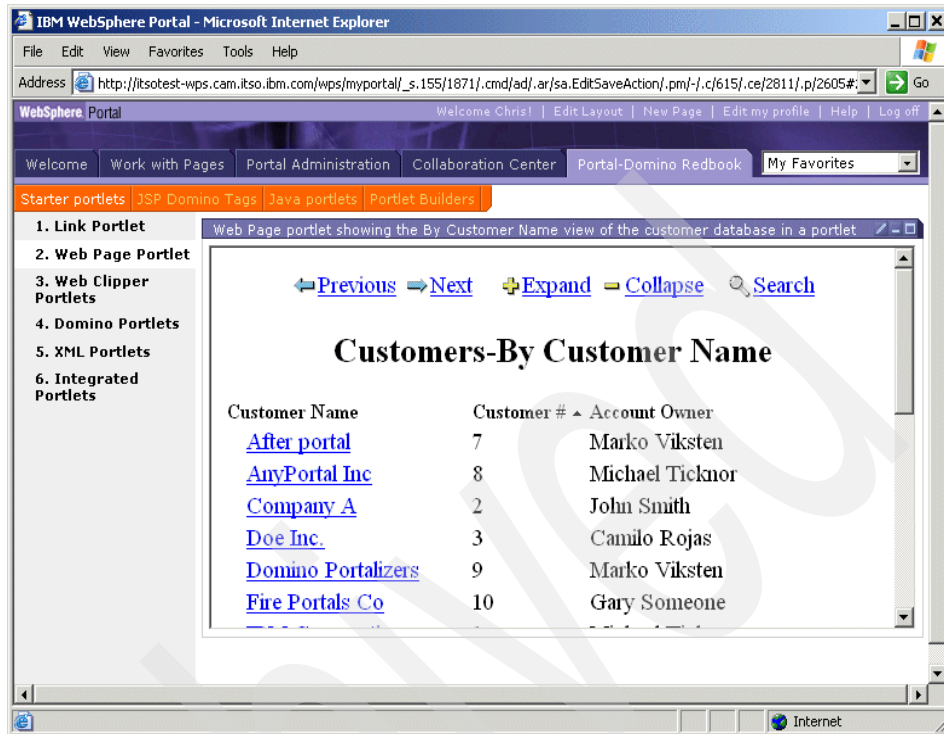


Figure 3-11 Configured Web Page portlet

Clicking any highlighted link within the portlet would perform that action inside of the portlet if that element was able to be displayed in the Web. For example, if you click the first link under Customer Name, the database record shown in Figure 3-12 is displayed inside the portal context.

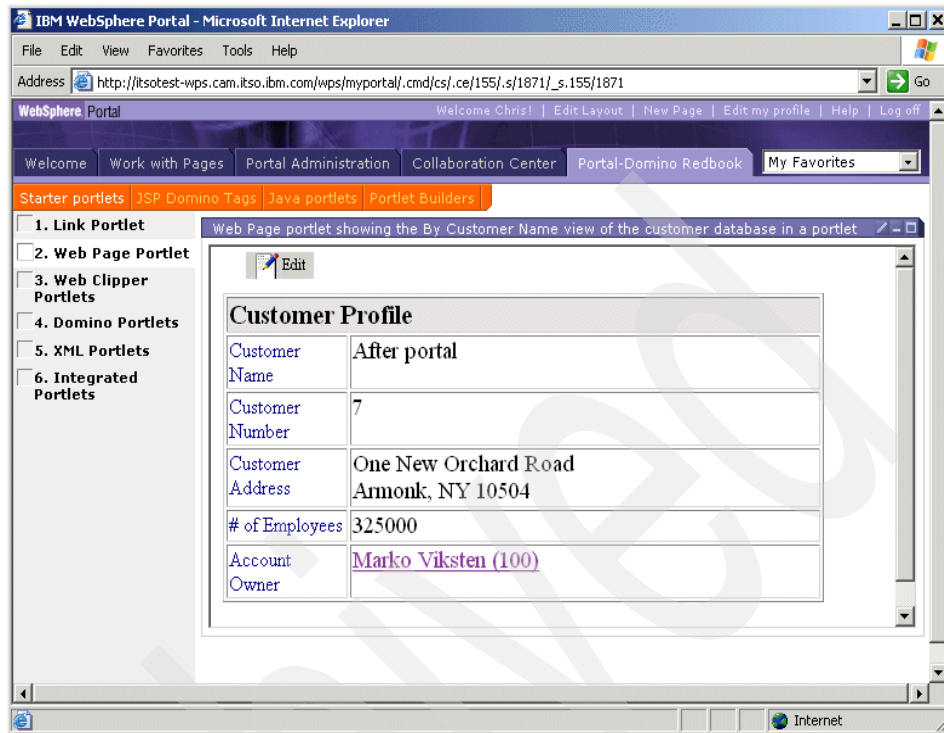


Figure 3-12 Result of clicking the first highlighted link inside of the portlet

3.4 Integrate using the Web Clipping portlet

The Web Clipping portlet displays, on a portal page, Web content that was “clipped” from another Web site. Web Clipping actually uses two portlets:

- ▶ Administration portlet - The administration portlet builds the resultant “runtime” portlet by obtaining the specifications of the external HTML page to be “clipped” and then clipping the regions of those pages that should be displayed in the portlet.
- ▶ Runtime portlet - The resultant “clipped” portlet is an instance of the Web Clipping portlet. This portlet is added to your page like any other portlet.

The Web Clipping portlet provides a quick and effective means of presenting portions of, or even whole Web pages in WebSphere Portal. We used the Web Clipping portlet to “clip” a Web-enabled view from one of our sample Domino databases.

Portlet listing

Table 3-6 contains information on the Web Clipping portlets.

Table 3-6 Portlet listing

Web Clipping	The name used in the documentation (for example, InfoCenter)
Web Clipping	Title in portlet selector (Edit Layout and Content")
Web Clipping Portlet	Name in portlet selector (installation)

When a Web page has been clipped, a new entry into the portal catalog will be shown that includes the new clipped portlet. For our example, the new listing is shown in Table 3-7.

Table 3-7 Clipped Web page portlet listing

Web Clipping Customer View	No information in the documentation for this portlet
Web Clipping Customer View	Title in portlet selector (Edit Layout and Content)
Web Clipping Customer View	Name in portlet selector (installation)

3.4.1 Considerations

The Web Clipping portlet takes content from a specified URL, extracts a selected subset of this Web page, and embeds the content into a portlet. Unlike the Web Page portlet, the content is rendered within the portlet context. However, this portlet does not always work as intended with existing HTML. For example, JavaScript does not always function as intended. Header information is mostly lost.

Display cache

The *display cache* is a feature that uses the WebSphere Application Server dynamic cache feature to store portlet output for a specified time before it expires. If a user requests that portlet during that timeframe, the content is drawn from the cache instead of from the portlet. This may not be desirable since you may want caching enabled for some, but not all, clipping portlets. An alternative is to specify a CacheTimeout configuration parameter for a specific portlet. To specify a caching timeout value, set the following configuration parameter in the clipping portlet under Portal Administration → Portlets → Manage Portlets. Highlight your portlet and click Modify Parameters. Specify:

```
CacheTimeout = ##
```

where ## is some number of seconds. Note that this parameter has no effect if the display cache is not enabled for the portal.

Applicable portlet patterns	Display
Development time	Insignificant
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills would be useful, but are not required.
Range of applications	Narrow
Handle rich text	No
Performance	Limited
Session Management	No
Clustering	No
Scalability	No
Requires single sign-on	Depending on the application, the Web Clipping portlet has extensive firewall, security, and authentication configurations.
Required software versions	Standard HTML browsers with JavaScript support are required. Netscape 5 and above and Internet Explorer 4 and above are supported.

3.4.2 Implementation details

In this section we describe how to use this technique to “clip” a Web-enabled view from one of your sample Domino databases and generate a runtime portlet. We have already “clipped” the example for presentation purposes. We have placed the administration Web Clipping portlet on a page along with our completed portlet for this demonstration. Although we have included the Web Clipping (administration portlet) on the page, it can also be accessed (and most likely would be) by clicking Portal Administration -> Web Clipping. This approach would generally be used since the administration portlet would not be shown to users. We used the first method in our example just to show both portlets in the same page. To fully explain this implementation, we provide complete details concerning:

- ▶ Initial setup
- ▶ Administration portlet options
- ▶ Configuration options
- ▶ Results

Initial setup

Our example begins with accessing the Web Clipper portlet page. Deploy the portlet and add it to a page; Figure 3-13 shows the result.

The screenshot shows the IBM WebSphere Portal interface in Microsoft Internet Explorer. The browser address bar displays the URL: http://itsotest-wps.cam.its.ibm.com/wps/myportal/_s.155/1901/.cmd/ad/.ar/1713089082/.c/621/.ce/1923/.p/1716#1923. The portal navigation bar includes links for Welcome, Work with Pages, Portal Administration, Collaboration Center, Portal-Domino Redbook, and My Favorites. Below this, there are tabs for Starter portlets, JSP Domino Tags, Java portlets, and Portlet Builders.

The main content area is divided into two sections. The top section, titled "Web Clipping", contains the text "Add, edit, and delete Web clippers" and a list of "Web clippers":

- Customers
- John Doe3
- Web Clipping Customer View

Each item has associated actions: Add, Edit (checked), and Delete.

The bottom section, titled "Web Clipping Customer View", displays a table of customer information. The table has three columns: Customer Name, Customer #, and Account Owner. The data is as follows:

Customer Name	Customer #	Account Owner
After portal	7	Marko Viksten
AnyPortal Inc	8	Michael Ticknor
Company A	2	John Smith
Doe Inc.	3	Camilo Rojas
Domino Portalizers	9	Marko Viksten
Fire Portals Co	10	Gary Someone
IBM Corporation	1	Michael Ticknor
ITSO	5	John Smith
Joe's Pizza	13	Camilo Rojas
Keyboards Inc	17	Gary Someone
My Bank.com	19	Marko Viksten
MyCompanyPortal	4	Chris Heltzel
Neo Portals	18	Michael Ticknor
Portal-makers	3	Camilo Rojas
Portal People eaters	15	Marko Viksten
Portals-R-Us	11	Chris Heltzel
The company	6	Marko Viksten
Wanda's World of Wigs	14	Chris Heltzel
Winging Portals	12	Chris Heltzel
WP Experts	16	Gary Someone

Figure 3-13 Web Clipper Portlets page

A close-up of the administration portlet is shown in Figure 3-14.

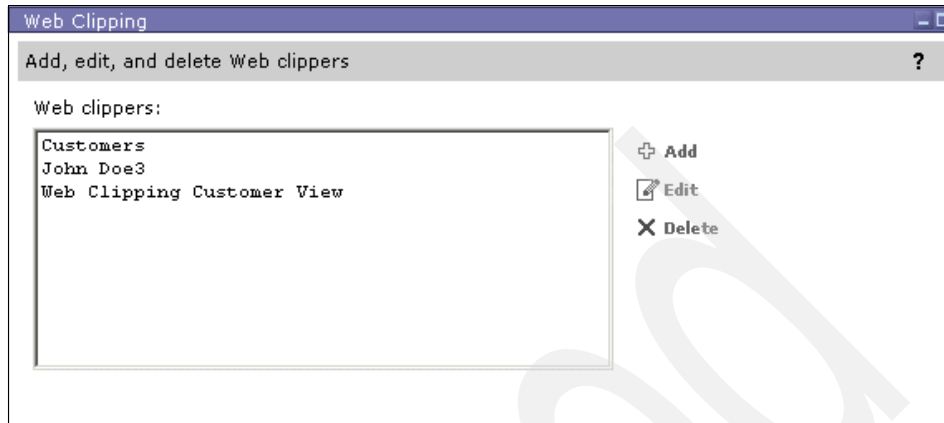


Figure 3-14 The Web Clipping (administration) portlet

Administration portlet options

- Add Web clipper** To add a Web clipper, click Add. Follow the Web clipping configuration options and then add the generated portlet to a page and activate.
- Edit Web clipper** To modify the settings for a Web clipper, select it from the Web clippers list, and click Edit. Make your desired changes to the Web clipper configuration information, and click Done.
- Delete Web clipper** To delete a Web clipper, select it from the Web clippers list, and click Delete. A confirmation dialog appears. Click OK to delete the Web clipper. Click Cancel to keep the Web clipper.

Note: When deleting a Web clipper, be sure to delete both the portlet itself, and its associated portlet application, through Portal Administration. Otherwise, you will not be able to create another Web clipper using the same name.

Configuration options

The Web Clipping portlet has a number of configuration options. This portlet actually has three basic edit screens that must be completed. The first edit screen is shown in Figure 3-15 on page 84. Be sure to click Next on the bottom of the first edit screen to go on to the second edit screen. If you click Cancel on this screen, you will exit the edit view and no changes will be saved.

Web Clipping

Add a Web clipper

Next | Cancel

Name and default locale title:

Description:

Set locale specific titles and descriptions

URL to clip:

Connection timeout (seconds):

Modify clipping type Select how you will choose the content to clip.

Modify firewall options If you use a proxy server to access the con

Modify authentication options If you need a user ID and password

Modify rules for URL rewriting If URLs in the clipped content need s

Modify security options Choose whether to include or remove Java

Next | Cancel

Figure 3-15 Adding a Web clipper: Edit screen 1 options

The first Web Clipping edit screen options are the following:

- ▶ Name and default locale title:
Specify the name for the Web clipper. This is the default locale title. Select the Set locale-specific titles and descriptions option to set locale-specific titles.
- ▶ Set locale-specific titles and descriptions:
Select this option to specify locale-specific titles and descriptions for the Web clipper. This option only appears if the portal is configured to support more than one language. A list of locales appears. Select a locale, then click Set title for selected locale. A prompt for the title appears. Type in the Web clipper title for the selected locale, then click OK. Repeat this procedure for each locale for which you want a locale-specific title. Click Done when you have finished setting locale values. Figure 3-16 shows the locale options.

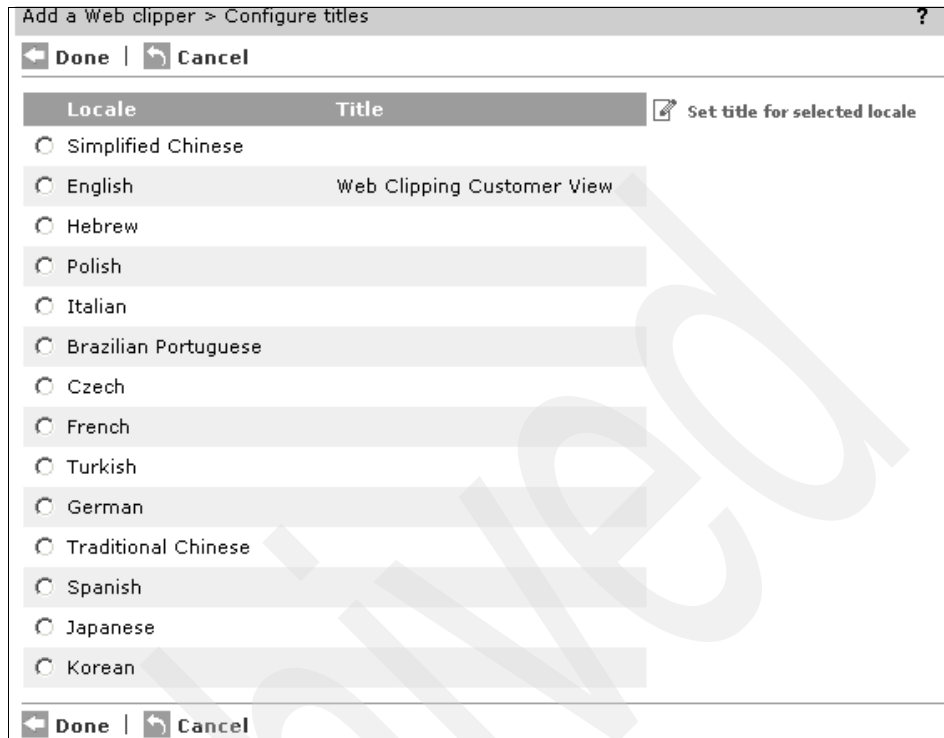


Figure 3-16 Set title for selected locale page from edit screen 1 options

- ▶ **Description:**
Specify a description to provide more detailed information about the Web clipper.
- ▶ **URL to clip:**
Enter the URL of the Web page you want to clip.
- ▶ **Connection timeout:**
Enter the amount of time, in seconds, that you want to keep the connection to the original server open when idle (maximum time: 100).
- ▶ **Modify clipping type:**
To change the type of clipping that the Web clipper performs, click Modify clipping type when creating or editing a Web clipper. Specify the clipping method, and click Done. Click Cancel to return to the edit page without modifying the clipping type. The Modify clipping type screen is shown in Figure 3-17.

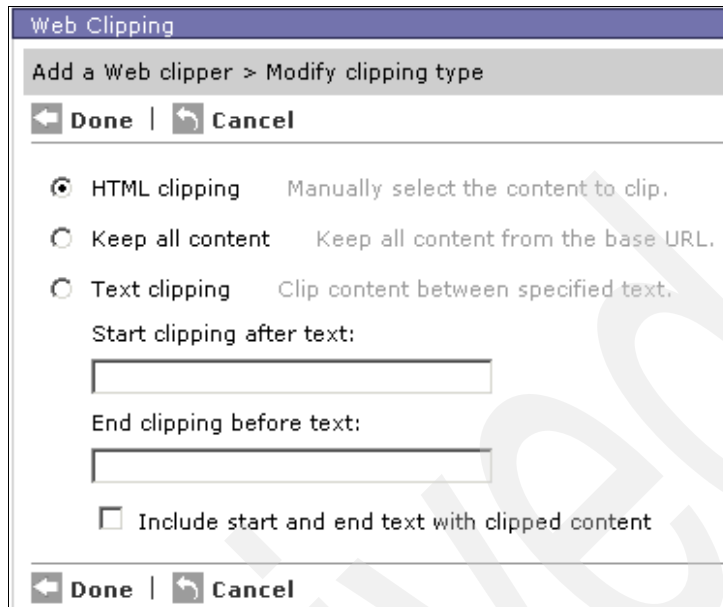


Figure 3-17 Modify clipping type page from edit screen 1 options

The following are options on the modify clipping type page:

- HTML clipping: This choice enables you to manually select elements of the document by clicking them with the mouse. Selected elements are highlighted in yellow to indicate that they will be kept after clipping.

Note: When performing HTML clipping, consider using the following:

To select more than one element, hold the Control key when you click on additional elements. By default, a single click selects an element, and if any other elements were previously selected, they are deselected.

You might need to experiment on the best place to click to highlight the particular element you are interested in, whether it is the contents of a table cell or the entire table itself. You can also click mouse button 2 and step up through the element nesting order to whatever level you wish. To ensure that you get the expected content, pay attention to the highlighting.

- Text clipping: This choice enables you to select the content between specific text strings that are in the HTML document. Content between these strings is kept, and all other content is discarded.

- Start clipping after text: Begin clipping (or keeping) content after this text string.
 - End clipping before text: Stop clipping content after this text string.
 - Include start and end text with clipped content: Select this option to include the text strings in the resulting output.
 - Keep all content: This choice brings the entire Web page into your portlet without discarding any content.
- **Modify firewall options**
 If you are clipping an HTML document accessed through a firewall, specify the proxy server information. Click **Modify firewall options**. Click **Use a proxy server**. Enter the hostname and port of the proxy server, along with a user ID and password, if necessary. Click **Done**. Click **Cancel** to return to the edit screen without modifying the firewall options. The **Modify firewall options** are displayed in Figure 3-18.

The screenshot shows a dialog box titled "Web Clipping" with a breadcrumb "Add a Web clipper > Modify firewall options". At the top, there are "Done" and "Cancel" buttons. Below this is a checkbox labeled "Use a proxy server" which is currently unchecked. Underneath the checkbox are four text input fields: "Proxy hostname:", "Proxy port:", "User ID:", and "Password:". At the bottom of the dialog, there are again "Done" and "Cancel" buttons.

Figure 3-18 Modify firewall options page from edit screen 1 options

The following are options on the **Modify firewall options** page:

- Do not use a proxy server: This choice indicates that you do not require a proxy server to access the HTML document you want to clip.
- Use a proxy server: This choice indicates that you require a proxy server to access the HTML document.

Note: If you specify proxy settings, the proxy *must* have access to the target content. If the proxy does not, local system access will not be used, and you will receive an error in the browser indicating that the content could not be retrieved.

- Proxy host: Enter the hostname of the proxy server.
 - Proxy port: Enter the port number used to communicate with the proxy server.
 - User ID: If required, enter the user ID used to access the proxy server.
 - Password: If required, enter the password used to access the proxy server.
 - Confirm password: Enter the password again for verification.
- **Modify authentication options:**
If you are clipping an HTML document which requires authentication for access, provide the authentication information.
To specify the proxy server information, click **Modify authentication options**. Click **Authentication required**. Select the type of authentication to be used and enter any appropriate information, such as a user ID and password. Click **Done**. Click **Cancel** to return to the edit screen for modifying the authentication options. The authentication options are shown in Figure 3-19.

Web Clipping

Add a Web clipper > Modify authentication options

← Done | ↻ Cancel

No authentication required
 Authentication required
 Set credentials

HTTP Basic Authentication
 Realm:

Form-based authentication
 Log-in URL:

 User parameter name:

 Password parameter name:

 Additional key value pairs:

← Done | ↻ Cancel

Figure 3-19 Modify authentication options page from edit screen 1 options

The following are options on the Modify authentication options page:

- No authentication required: Select this choice if no authentication is needed to access the HTML document.
- Authentication required: Select this choice to define an authentication method for accessing the HTML document.
- Set credentials: Clicking this takes you to the credential vault. Figure 3-20 on page 91 shows the credential vault options screen.

Note: The Web Clipping Portlet now stores authentication information within the WebSphere Portal's credential vault. The authentication settings page has a link to a credential settings page.

Select to either use a shared credential slot or a non-shared slot. In either case, the slots must already have been configured on the Credential Vault administration page. The shared credential slot will already have the user ID and password credential, which cannot be modified on this page. The non-shared slot is a slot identifier only. If a non-shared slot is selected, a user ID and password is requested in order to create a credential for the administrator building this clipping portlet.

The editor will pull the user ID and password from the credential in order to log in to the backend system. From then on, the credential vault is required to store all credential information.

When the clipping portlet is later edited, the Authentication Settings page will pull up the credential of the user editing the clipping portlet and will show user ID and password if the credential is non-shared.

If the Web clipper is using a shared credential, the portlet will access the backend content using that credential without requiring the user to log in first. If the Web clipper specifies a non-shared slot, and an instance of that slot does not yet exist for this portlet and the user accessing it, then the user will have to edit the portlet and provide a user ID and password for logging into the backend system. This user ID and password are stored in a new slot instance for the defined non-shared slot. The next time that user accesses this portlet, the credential will be reused and the user will not have to log in again.

Editing of the portlet at runtime is still available to users with existing credentials to allow them to modify the user ID and password.

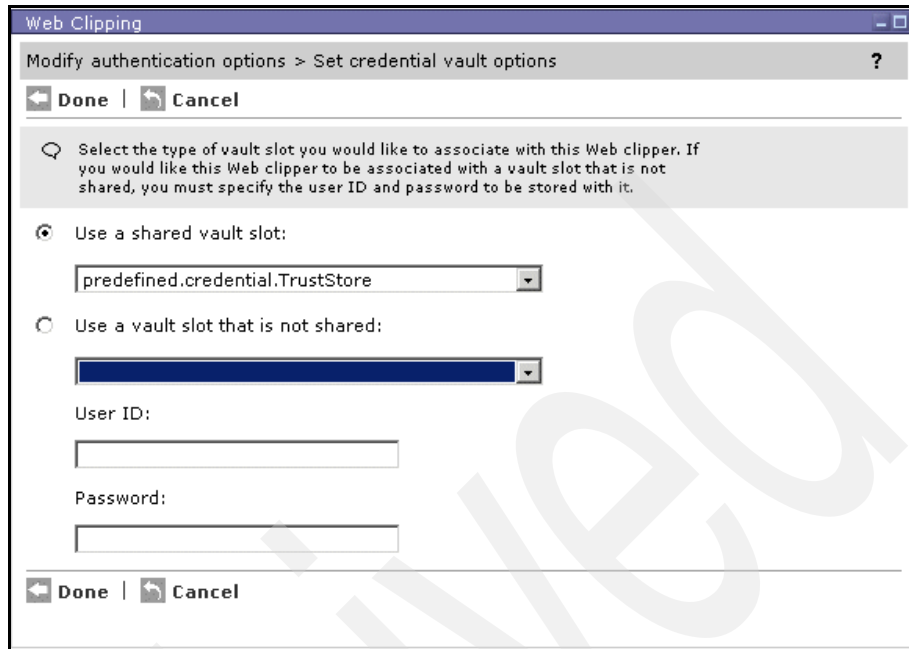


Figure 3-20 Credential vault options

- HTTP Basic Authentication: If HTTP basic authentication is used to access the HTML document, specify the realm to which the document belongs.
- Form-based Authentication: If the HTML document you are clipping is a form, specify the URL and parameters used by the form.
- Log-in URL: Enter the URL of the form. To locate the target URL of the form submission, look for the FORM tag on the login page (browse the source of the page) and locate the ACTION attribute. The URL in the ACTION is the URL that you need to specify.
- User parameter name: To determine this value, log in to the page, right click on View Source, and use the name attribute of the INPUT tag.
- Password parameter name: To determine this value, log in to the page, right click on View Source, and use the password attribute of the INPUT tag.
- Additional key value pairs: Enter any additional key value pairs corresponding to other parameters required by the form. Use an ampersand (&) to separate each pair.
- ▶ Modify rules for URL rewriting: To change the way URLs are handled by the Web clipper, you can specify rules for URL rewriting. The regular expression

syntax for specifying URL rewriting rules is based on the Perl standard. Enter the appropriate entries and click Done. Click Cancel to return without modifying the rules. Figure 3-21 shows this option.

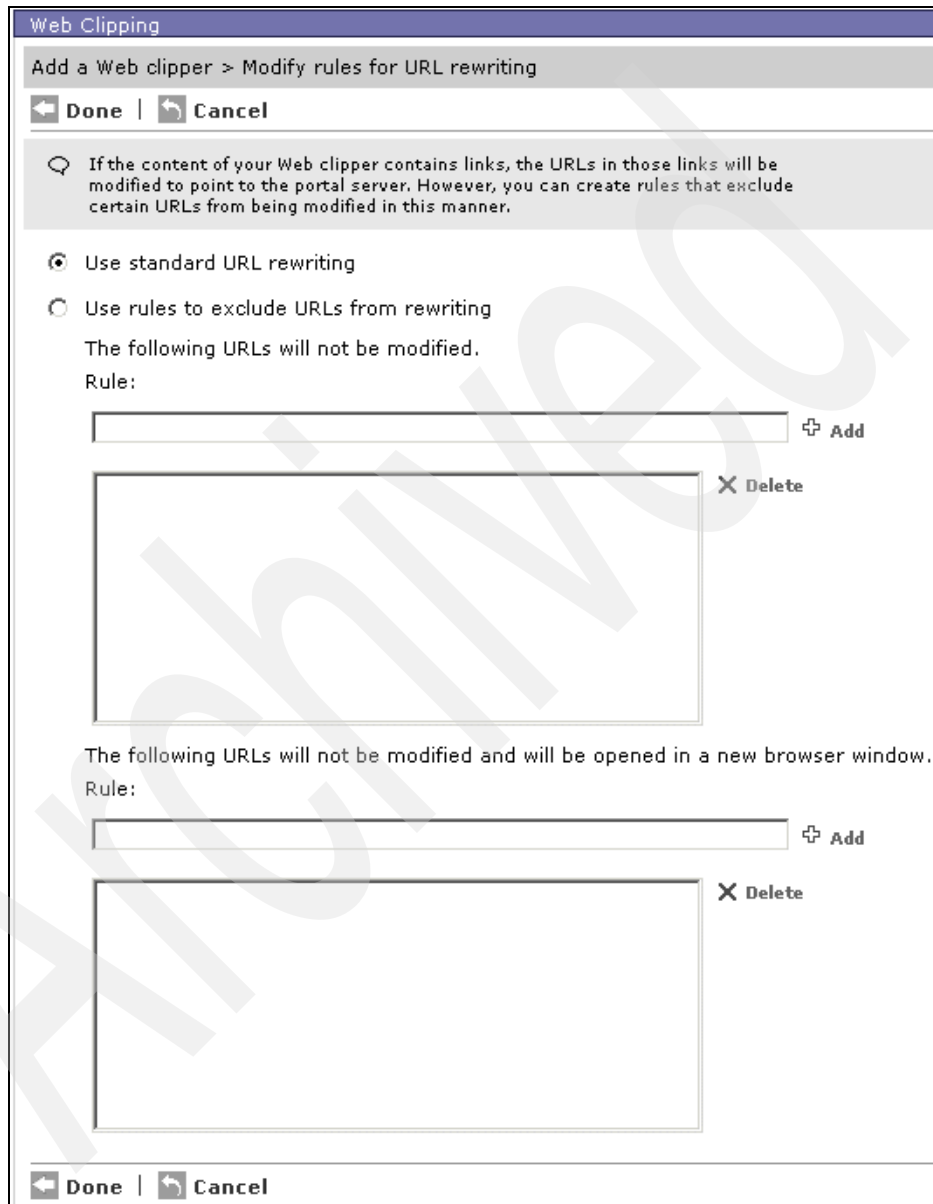


Figure 3-21 Modify rules for URL writing page from edit screen 1 options

The following are options on the Modify rules for URL rewriting page:

- Use standard URL rewriting: This choice ensures that URLs in links within the Web clipper are modified to point to the portal, rather than the host originally targeted by the link.
 - Use rules to exclude URLs from rewriting: This choice enables you to specify rules that are used to identify which URLs will not be altered. These links will not go through the portal. For example, a rule of `*.gif$` will match any URL that ends in `.gif`, while a rule of `*ibm.com.*` will match any URL containing `ibm.com`. In the field “The following URLs will not be modified,” enter one or more rules to identify URLs whose content will be retrieved from the original host. These links are opened in the same window. Click Add to add listings or highlight a link and click Delete to delete. In the field “The following URLs will not be modified and will be opened in a new window,” enter one or more rules to identify URLs whose content will be retrieved from the original host. These links are opened in a new window. Click Add to add listings or highlight a link and click Delete to delete.
- **Modify security options:**
Choose whether to include or remove JavaScript in the clipped content. Check Remove JavaScripts from the clipped content to remove JavaScript from the clipped contents. Click Done or Cancel. This option is shown in Figure 3-22.

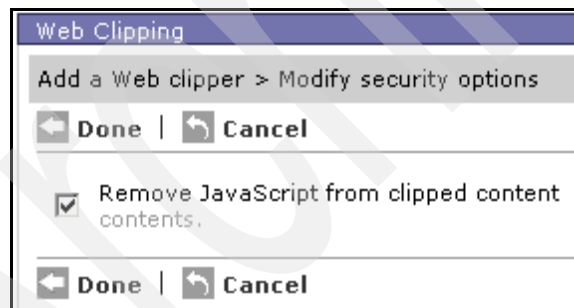


Figure 3-22 Modify security options page from edit screen 1 options

Web Clipping edit screen 2 (clip contents) is shown in Figure 3-23. Choose the content you want to keep by pointing and clicking.

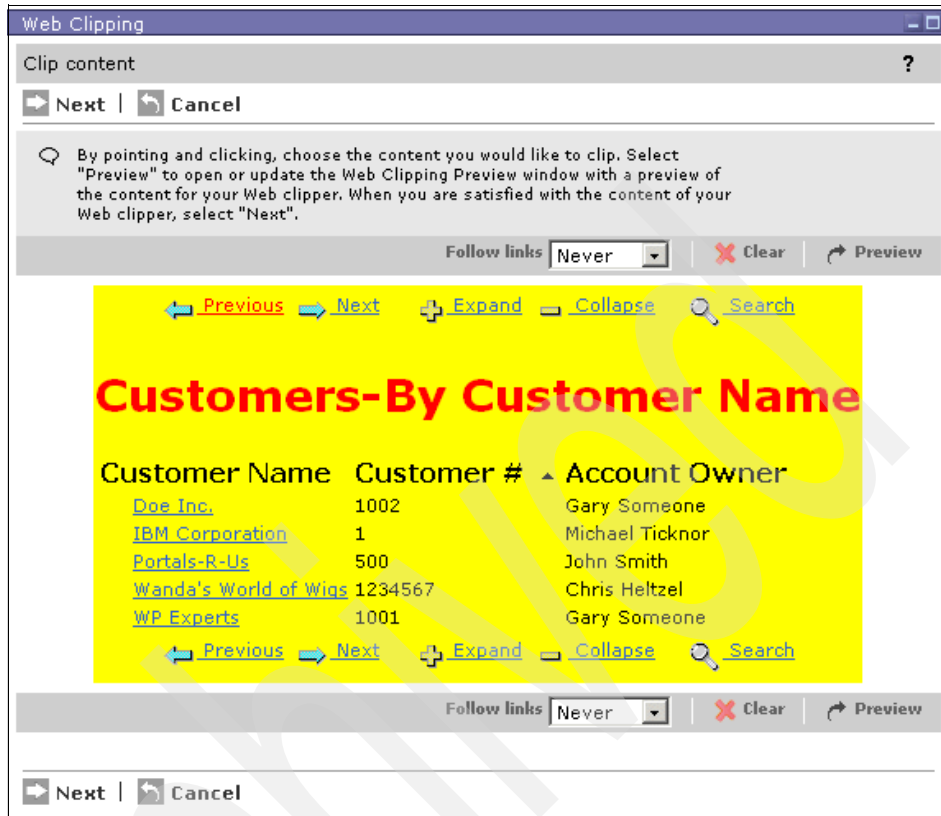


Figure 3-23 Web Clipping portlet edit screen 2 options

The selected contents will be yellow. The options on this edit screen are the following:

- Next: Click Next to progress to the next screen after you have made your selection.
- Cancel: Click Cancel if you would like to go back to the previous screen.
- Follow Links: Specify whether links should be followed. The choices are Never and Ask. The default is Never.
- Clear: Clicking Clear resets the Specify a description to provide more detailed information about the Web clipper.
- Preview: Clicking Preview pops up a browser to show your highlighted selection.

Web Clipping Edit screen 3 (content preview) options are shown in (Figure 3-24 on page 96). This screen contains the preview of the clipped Web page. Click

Done to keep your new content; click Cancel if you would like to go back to the previous screen.

Table 3-8 provides the configuration values that we used in our example.

Table 3-8 Configuration values for the Web Clipping portlet example

Field	Inserted value
Name and default local title	Web Clipping Customer View
Description	This is an example of the Web clipper clipping the customer view
URL to clip	http://itsotest-dom.cam.itso.ibm.com/apps/CUSTOMER.NSF/CustomersByName?OpenView
Connection timeout (seconds)	5
All other fields	default

Results

The result of setting up the Web Clipper, “clipping” the Web page, and placing the portlet on the page is shown in the original figure, Figure 3-13 on page 82. A close-up of our Web Clipped Web page is shown in Figure 3-24.

The screenshot shows a web application window titled "Web Clipping Customer View". At the top right, there is a "Back" button. Below the title bar, there are navigation links: "Previous", "Next", "Expand", "Collapse", and "Search". The main content area features a large heading "Customers-By Customer Name". Below this heading is a table with three columns: "Customer Name", "Customer #", and "Account Owner". The table lists 20 entries, each with a blue underlined link for the customer name, a numerical customer ID, and the name of the account owner. At the bottom of the table, there are the same navigation links as at the top: "Previous", "Next", "Expand", "Collapse", and "Search". A "Back" button is also present at the bottom right of the window.

Customer Name	Customer #	Account Owner
After portal	7	Marko Viksten
AnyPortal Inc	8	Michael Ticknor
Company A	2	John Smith
Doe Inc.	3	Camilo Rojas
Domino Portalizers	9	Marko Viksten
Fire Portals Co	10	Gary Someone
IBM Corporation	1	Michael Ticknor
ITSO	5	John Smith
Joe's Pizza	13	Camilo Rojas
Keyboards Inc	17	Gary Someone
My Bank.com	19	Marko Viksten
MyCompanyPortal	4	Chris Heltzel
Neo Portals	18	Michael Ticknor
Portal-makers	3	Camilo Rojas
Portal People eaters	15	Marko Viksten
Portals-R-Us	11	Chris Heltzel
The company	6	Marko Viksten
Wanda's World of Wigs	14	Chris Heltzel
Winging Portals	12	Chris Heltzel
WP Experts	16	Gary Someone

Figure 3-24 Close-up of “clipping” using the Web Clipping portlet

Clicking a highlighted document will cause it to be displayed within the portlet context, as shown in Figure 3-25.

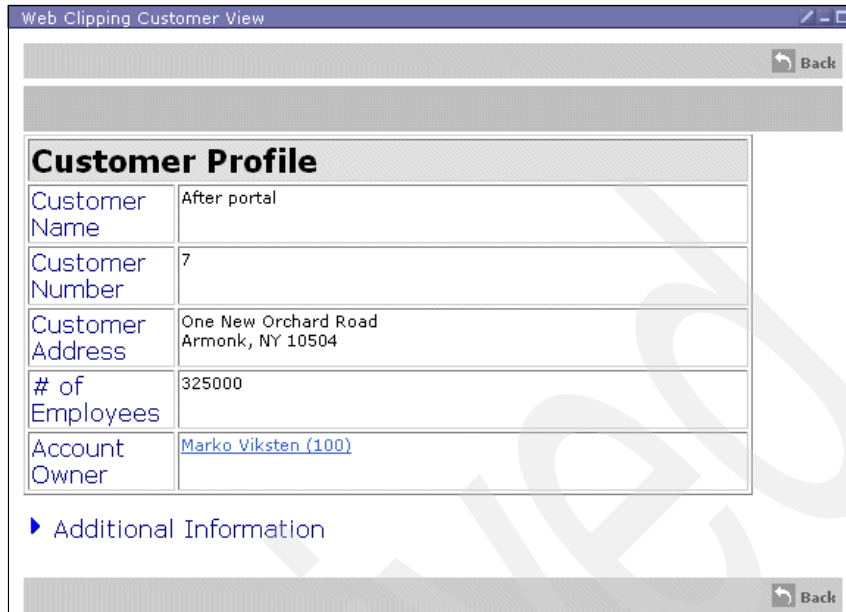


Figure 3-25 Result of clicking a highlighted link in the Web Clipped portlet

The navigation row of the “clipped” view also functions. If you click the Next navigation icon, the screen will show the remaining items in the view, as shown in Figure 3-26.

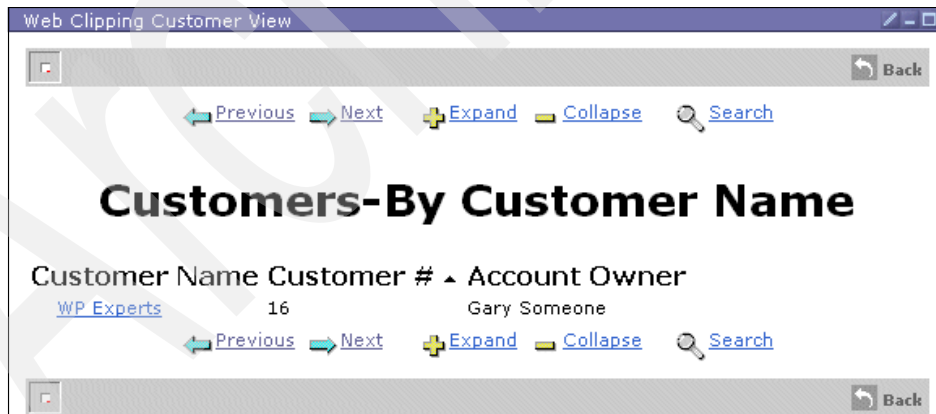


Figure 3-26 Result of clicking the navigation Next

3.5 Integrate using Lotus Notes portlets

Using the Lotus Notes portlets available in WebSphere Portal Extend is another example of the simplicity of using prepackaged portlets. This technique configures a Web-enabled view in your Domino application to the portal environment through direct and efficient configuration screens. The view is displayed in your portal context and no external Web browser session will open.

3.5.1 Lotus Notes portlets

There are a number of prepackaged Lotus Notes portlets available in WebSphere Portal Extend. In the beta and soon to be released version 5 of WebSphere Portal Extend, the number of portlets drops to two. This is because the two portlets will include the configurations of all Lotus Notes portlets. The collaborative feature of Lotus Notes portlets is of particular use to quickly integrate existing Domino applications into the WebSphere Portal environment. Table 3-9 lists a number of useful portlets, and although this is not an exhaustive list, it provides a good understanding of the use of these portlets. The Redpaper, “WebSphere Portal Collaboration Services,” REDP0319, provides a specialized look into these collaboration portlets.

In this section we describe the use of NotesViewPortlet as it can be used to show any view from any Domino database.

Table 3-9 Collaborative portlets provided in WebSphere Portal Extend

Portlet Name	Portlet Description
iNotesMailPortlet iNotesCalendarPortlet iNotesContactsPortlet iNotesNotebookPortlet iNotesToDoPortlet	Provides iframe-based access to iNotes™-enabled Domino servers and mail files. It includes access to the Welcome, Mail, Calendar, To Do List, Contacts, and Notebook iNotes functions.
MyNotesMailPortlet MyNotesCalendarPortlet MyNotesToDoPortlet	Displays users' mail, calendar, and to-dos from their traditional Lotus Domino mail file. The “my” name designates the fact that these portlets autodetect a user's mail file settings—and thus do not need to be configured individually for each user.
NotesMailPortlet	Displays any view in a traditional Domino mail database.
NotesDiscussionPortlet	Displays a Notes database built with the Domino Discussion database template.
NotesTeamroomPortlet	Displays a Notes database built with the Domino Teamroom database template.

Portlet Name	Portlet Description
NotesViewPortlet	Displays any view in any Domino database.
In-line Quickplace Portlet	Displays a QuickPlace® inside an iframe.
QuickplacePortlet	Launches a designated QuickPlace in a separate browser window. This portlet is also included in WebSphere Portal Enable.
SametimePortlet	Launches the Sametime Connect Java client. This portlet is also included in WebSphere Portal Enable.

Portlet listing

We used the portlet setting in our example technique. This portlet is designed to allow you to display any Notes database view in a portlet window. It is the most flexible of all the Notes portlets, as it is not limited to databases with specific templates, such as the NotesDiscussion portlet.

Lotus Notes View Portlet	Name used in the documentation (for example, InfoCenter)
Lotus Notes View Portlet	Title in portlet selector (Edit Layout and Content™)
NotesViewPortlet	Name in portlet selector (installation)

3.5.2 Considerations

The Lotus Notes View portlet can be used to implement the basic Display portal pattern. However, when a highly customized interface or a requirement with additional functionality is needed, the Lotus Notes View portlet may not be a good choice.

Applicable portlet patterns	Display
Development time	Insignificant
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills would be useful but not required.
Range of applications	Moderate
Handle rich text	Display
Performance	Limited
Session Management	No
Clustering	No

Scalability	No
Requires single sign-on	Depending on the application, the Lotus Notes View portlet can be configured to use single sign-on.
Required software versions	<p>Each portlet may have its own software requirements. The Lotus Notes portlets generally require:</p> <p>Client requirements:</p> <p>Notes client versions supported - 4.67, 5.01, 5.05 (for Notes Calendar portlets only), 5.09, 6.0</p> <p>Web browser versions supported: Netscape Navigator 4.7 and later releases, including Netscape Navigator 6.2, Microsoft Internet Explorer 4.01 and later releases, excluding Microsoft Internet Explorer 6.x. JavaScript and Java applets must be enabled.</p> <p>Server requirements:</p> <p>Lotus Notes portlets require network access to the Domino servers that host the Notes databases that are the sources of portlet information. The HTTP service must be running on the Domino server.</p> <p>Lotus Notes portlets work on the following versions of Domino: 4.67, 5.01, 5.05 (for Notes Calendar portlets only), 5.09, 6.0</p>

3.5.3 Implementation details

This section describes how to use the Lotus Notes View portlet to display within the portal context view information from one of the example databases. To fully explain this implementation, we provide complete details concerning:

- ▶ Initial setup
- ▶ Configuration options
- ▶ Results

Initial setup

Begin by accessing the Lotus Notes View portlet. Deploy the Notes View portlet and add it to a page. Figure 3-27 shows the Domino portlets page before configuration.

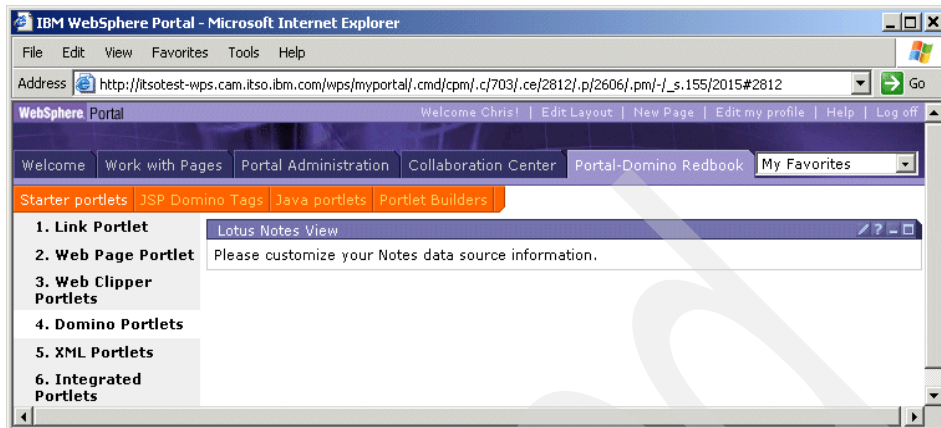


Figure 3-27 Domino portlets page before configuration

Configuration options

This portlet has two configuration screens that must be completed. Figure 3-28 on page 102 shows the first configuration screen.

Lotus Notes View: Customer view of the Customer database

Portlet title:

Source
 Server:

Database filename:

Database:

- bookmark.nsf
- Catalog (R5)
- Domino LDAP Schema
- Domino Server.Planner Sample DB
- Domino Web Administrator (R5)
- Domino Web Server Configuration (R5.0)
- Domino Web Server Log

View:

- ContactsRSSbyName
- Customer Contacts\By Customer Name
- Customer Contacts\By Customer Number
- Customer Contacts\By Name
- CustomerContactsPortletIntegrated
- CustomerPortletIntegrated
- Customers\By Account Owner
- Customers\By Customer Name

View category:

Protocol:
 HTTP
 HTTPS (SSL)
 Detect protocol automatically

Figure 3-28 Lotus Notes View portlet edit mode: Page one

Click Next on the bottom of the first edit screen to go to the second screen. If you click OK on the first edit screen instead, you will exit the edit view and be left with

the default settings for much of the portlet's configuration. Clicking Cancel will exit the edit screens without saving any configuration changes. Details for the edit screens follow.

► Edit screen 1 options (Figure 3-28):

- The first configuration setting to make is to name the portlet. This name will show at the top of the portlet window in the portal.
- The next few options are basic options selecting the server, database, and the view for the portlet. After selecting a server, select the check mark, and the portlet populates the database list. The check box next to the database list then populates the view list once you have chosen a database, and so forth.

Note: There is some support for spelling in these configuration fields. For example, the database name does not have to end in “.nsf”.

If you don't see your servers in the server list, or the databases/views to choose, then it is possible that the access rights are not properly set in your environment. Verify that you have enabled all the correct Domino settings (LDAP, HTTP, DIIOP, and so on).

- The protocol that is used for the portlet can also be selected on this screen. If Detect protocol automatically is selected, the portal will first try HTTP. If that is unavailable, it will try HTTPS.

After completion of the first edit screen, go on to Edit screen 2, which is shown in Figure 3-29.

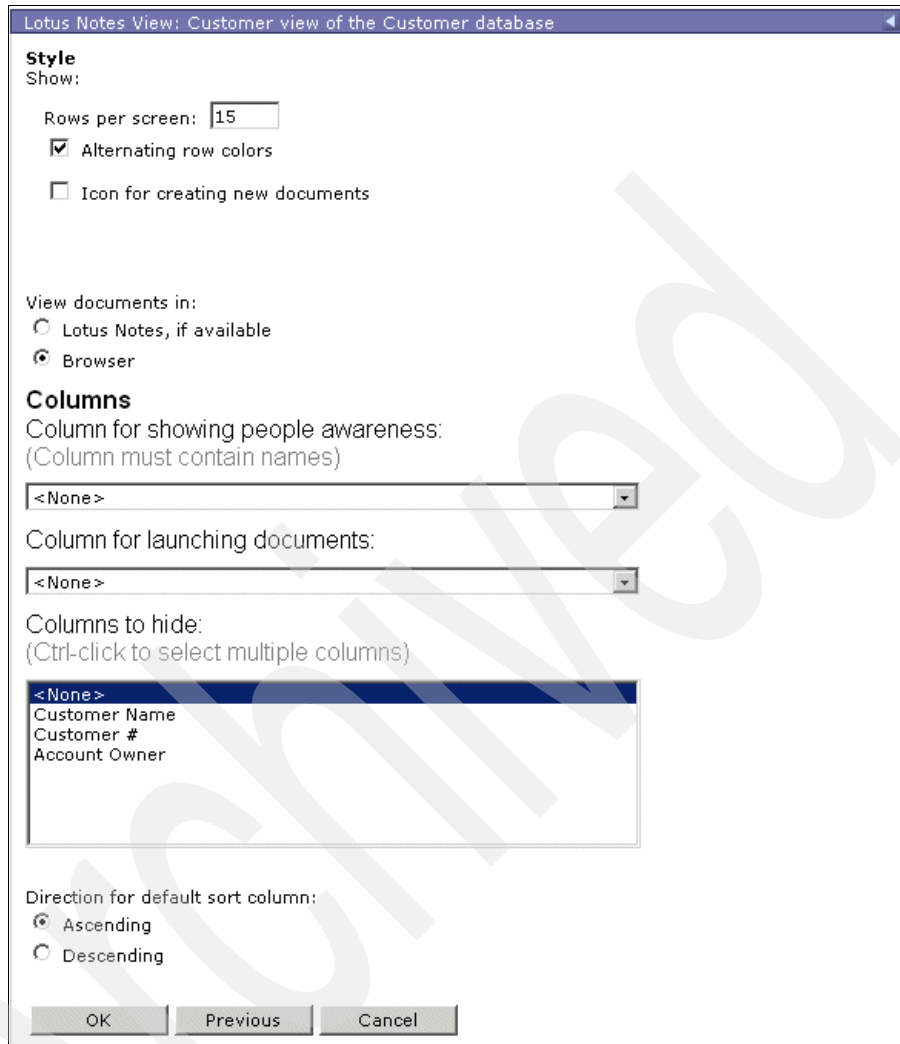


Figure 3-29 Lotus Notes View portlet edit mode: Page two

Click OK on the bottom of this edit screen to complete the configuration and return to display mode. If you click Previous on this edit screen instead, you will return to the first edit screen. Clicking Cancel will exit the edit screens without saving any configuration changes. Details for the second edit screen follow.

- ▶ Edit screen 2 options (Figure 3-29):
 - Choose the number of rows to appear in your portlet. The default setting is 15.

- Choose whether or not to alternate row colors to better highlight the row divisions. The default setting is Yes.
- You can choose to display the icons to create new documents. If you check this option, you then have to enter which Notes form is used for creating new documents. The default setting is No.
- Choose how documents the user wishes to view are launched—either in the Lotus Notes client, or in another Web browser window. The default setting is to use the browser.

Restriction: The option to view documents in the Notes client only works with Internet Explorer browsers, as it functions via a small ActiveX control which launches the Notes client and passes the database and document details to it. Non-IE browser users will view documents within their browser via Domino HTTP.

- Choose whether categorized views are expanded or collapsed when opening. The default is expanded. If a view category is not selected this item is not displayed.
- Choose several options for the view columns:
 - Select which column will be enabled for people awareness, so that names will be highlighted if the people are online, and contact option menus will be presented. Collaboration portlets need to be installed on the server for this option to work.
 - Specify which column you want to use to launch the document selected via document action menus.
 - Select which columns (if any) you want to hide from the view as it is presented by the portlet.
- Finally, you must choose the direction for default sorting: either Ascending or Descending. The default is Ascending.

Configuration values used in our example are listed in Table 3-10.

Table 3-10 Configuration values for the Lotus Notes View portlet example

Field	Inserted value
Portlet title	Lotus Notes View: Customers view of the Customers database
Server (location of the Domino server)	itsotest-dom.cam.itso.ibm.com
Database filename (including filepath if not in the Domino Data directory)	apps/Customer.nsf

Field	Inserted value
View	Customers\By Customer Name
View Category	We left this blank for our example
Protocol (HTTP/HTTPS/detect protocol automatically)	Detect protocol automatically
Which column you want to use to launch the document selected via document action menus	Customer Name

Results

After configuring the Lotus Notes View portlet, the result of the Domino portlets page is shown in Figure 3-30.

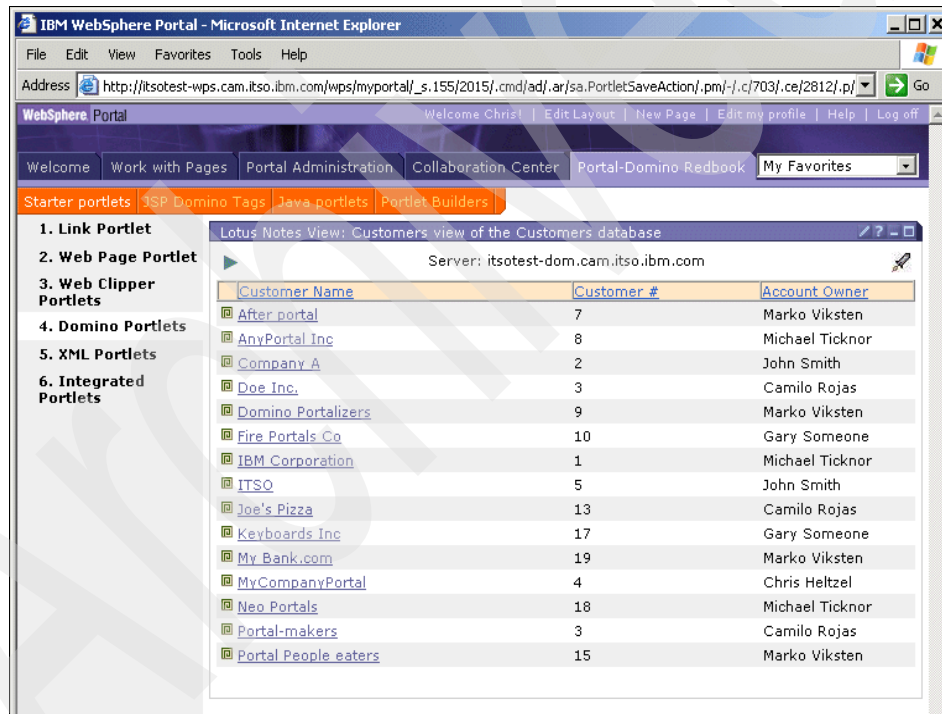


Figure 3-30 Result of modifying the Lotus Notes View portlet

If you click a highlighted link inside the portlet, you are able to open a document (if access allows) into a new Web browser session and out of the portal context, as shown in Figure 3-31.



Figure 3-31 Result of clicking the first highlighted link in the Lotus Notes View portlet

For column titles with a highlighted link, you can order the listing based on that column. The configured server always shows up below the portlet window's title region (itsotest-dom.cam.itso.ibm.com). If the portlet allows for launching the Notes client or a Web browser, the launch rocket icon is present below the title region. The arrows to the left of the server name are for navigation if more than one screen is available. If you have rights to create documents in the respective Domino database, which is "author" at minimum, then a "Create new document" icon also shows up next to the launch rocket icon.

3.6 Integrate using XML helper and RSS portlets

One way to better integrate Domino with the prepackaged portlets from WebSphere Portal is to modify your existing Domino data to use the RSS and the XML/XSL helper portlets.

eXtensible Markup Language

Examples in this option utilize eXtensible Markup Language (XML). XML is a set of rules for defining tags that break documents into parts. XML is useful because you can create tags. While these tags follow certain general principles, the meaning can be very flexible. Domino has supported XML since 1999. Document Type Definitions (DTDs) are repositories of these tags that make up a document. Domino uses a DTD to document the XML that Domino produces. It is also

relatively easy to get XML data from Domino. You might already have a view that can be used, or you may have to create one. You can pass the tags that XML uses through view columns.

eXtensible Stylesheet Language: Transformations

eXtensible Stylesheet Language: Transformations (XSLT) is a language designed to transform XML into another format. It also can be used to transform XML into HTML.

RSS portlet

The Rich Site Summary (RSS) news format is a simple, common format for delivering news to portals and other Web sites. The RSS Portlet renders XML data that conforms to the RSS DTD. However, it can be easily configured to render any XML data using any XSLT stylesheet.

XML/XSL helper portlet

An alternative to configuring the RSS portlet is to go get the XML from Domino and format it for presentation. This set of methods produces portlets by simply modifying your Domino application to serve up XML to the XML/XSL helper portlet. The style sheet used to render the view creates an HTML table out of the Domino XML (DXL) resulting from a ReadViewEntries request to the Domino database. Each <viewentry> template creates a new row in the table. Each <entrydata> template creates a new cell in the row. For simplicity, the entrydata template assumes that it has a text child element. This can be accomplished by passing the PreFormat option to ReadViewEntries.

3.6.1 Considerations

The RSS portlet only allows a single, static URL to be specified when the portlet is configured. The XML/XSL helper portlet utilizes the same technology. As an effective integration technique, these portlets are considerably reduced because of this specified static URL.

Applicable portlet patterns	Link, Display - Display portal patterns using a very small data set can be implemented using this technique.
Development time	Moderate
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills with experience with JavaScript, XML, and XSLT are needed.
Range of applications	Moderate

Handle rich text	If a connection is made to the Domino HTTP server directly (out of the portlet context) then Rich Text is shown, otherwise no.
Performance	Limited
Session Management	No
Clustering	No
Scalability	No
Requires single sign-on	Depending on the application, the RSS portlet or copies can be configured to use single sign-on.
Required software versions	Standard HTML browsers with JavaScript support are required. Netscape 5 and above and Internet Explorer 4 and above are supported.

3.6.2 Implementation details

This section describes how we configured a portlet page with Domino view data using copies of the RSS portlet, added elements to a Domino design to enhance the presentation, and configured our modified RSS portlets to display this data. To fully explain this implementation, we provide complete details concerning:

- ▶ Domino enhancements
- ▶ Portal configuration options
- ▶ Results

Domino enhancements

To see the design elements in detail, access the accompanying Domino databases. For the first RSS portlet, Customer Contacts RSS Portlet, we generated a new view, `ContactsRSSbyName` in the customer database. We had XML with a hardcoded URL link to a view column, which is presented in our portlet. This column code formula is shown in Figure 3-32.

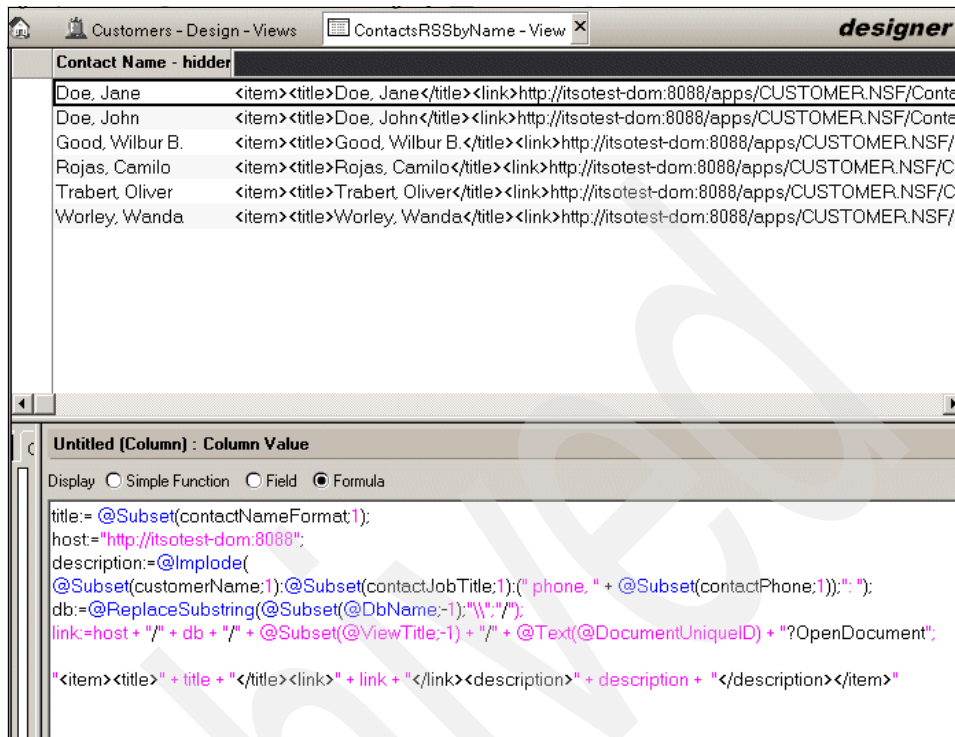


Figure 3-32 ContactsRSSbyName view

We added a view template, `$$ViewTemplate` for `ContactsRSSbyName` form in the database. This form incorporates XML as well as an embedded view containing the `ContactsRSSbyName` view. This is shown in Figure 3-33.



Figure 3-33 Creation of a `$$ViewTemplate` for the `ContactsRSSbyName` view

For the Customer Contacts RSS Portlet portlet, we styled the sheet with the generic XSLT that is included with WebSphere Portal.

For the Customers XML-XSLT portlet, we used the CustomersByName view shown in Figure 3-35 on page 112.

Customer Name	Customer #	Account Owner	hidden 4
After portal	7	Marko Viksten	After portal (7) 7
AnyPortal Inc	8	Michael Ticknor	AnyPortal Inc (8) 8
Company A	2	John Smith	Company A (2) 2
Doe Inc.	3	Camilo Rojas	Doe Inc. (3) 3
Domino Portalizers	9	Marko Viksten	Domino Portalizers (9) 9
Fire Portals Co	10	Gary Someone	Fire Portals Co (10) 10
IBM Corporation	1	Michael Ticknor	IBM Corporation (1) 1
ITSO	5	John Smith	ITSO (5) 5
Joe's Pizza	13	Camilo Rojas	Joe's Pizza (13) 13
Keyboards Inc	17	Gary Someone	Keyboards Inc (17) 17
My Bank.com	19	Marko Viksten	My Bank.com (19) 19
MyCompanyPortal	4	Chris Heltzel	MyCompanyPortal (4) 4
Neo Portals	18	Michael Ticknor	Neo Portals (18) 18
Portal-makers	3	Camilo Rojas	Portal-makers (3) 3
Portal People eaters	15	Marko Viksten	Portal People eaters (15) 15
Portals-R-U.s	11	Chris Heltzel	Portals-R-U.s (11) 11
The company	6	Marko Viksten	The company (6) 6
Wanda's World of Wigs	14	Chris Heltzel	Wanda's World of Wigs (14) 14
Winging Portals	12	Chris Heltzel	Winging Portals (12) 12
WP Experts	16	Gary Someone	WP Experts (16) 16

Figure 3-34 CustomersByName view in the Customers database

We wanted to present the view in our own style. We generated an XSLT stylesheet, naming the stylesheet Customers.XSL and placing it as a form in the Customers database. Figure 3-35 on page 112 shows this stylesheet.

```

Customers - Design - Forms Customers.XSL-Form X
CGI Variables: Server_Name T Server_Port T Server_Protocol T Query_String T
URLGENERATE: DIRECTORYURL T DBURL T
<?xml version='1.0'?>
<xsl:stylesheet version='1.0'
xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
<xsl:template match='viewentries'>
<table border="0"><tr><td>This is
CustomersByName?ReadViewEntries</td>
</tr></table>
<table cellspacing="1" cellpadding="4" border="0" width="100%">
  <xsl:apply-templates/>
</table>
</xsl:template>
<!-- Template: viewentry
Description: Creates a new row in the table for the entry.
Adds a link to the entry based on UNID. -->
<xsl:template match='viewentry'>
  <xsl:variable name="pos" select="@position"/>
  <xsl:element name="tr">
    <!-- Alternate bgcolor of the rows -->
    <xsl:if test="$pos mod 2"><xsl:attribute
name="class">wpsTableShdRow</xsl:attribute></xsl:if>
    <xsl:apply-templates/>
  </xsl:element>
</xsl:template>
<!-- Template: entrydata
Description: Reads the value out of a text child element. -->
<xsl:template match="entrydata">
  <xsl:variable name="col" select="@columnnumber"/>
  <td>
    <xsl:choose>
      <xsl:when test="$col = 0">
        <a href="<Computed Value>
CustomersbyName/(../@unid)?OpenDocument"
target="_blank"><xsl:value-of select="./text"/></a>
      </xsl:when>
      <xsl:when test="$col > 0">
        <xsl:value-of select="./text"/>
      </xsl:when>
    </xsl:choose>
  </td>
</xsl:template>
<!-- Default template does nothing -->
<xsl:template match='@*|node()' />
</xsl:stylesheet>

```

Figure 3-35 Customers.XSL stylesheet form in the Customers database

For more information on using XSLT and XML inside Domino, see the Lotus Domino Designer® 6 Help database or the IBM Redbook *Lotus Domino Designer 6: A Developer's Handbook*, SG24-6854.

Portal configuration options

To construct portlets, in the portal environment click Portal Administration → Manage Portlets. This page is shown in Figure 3-36.

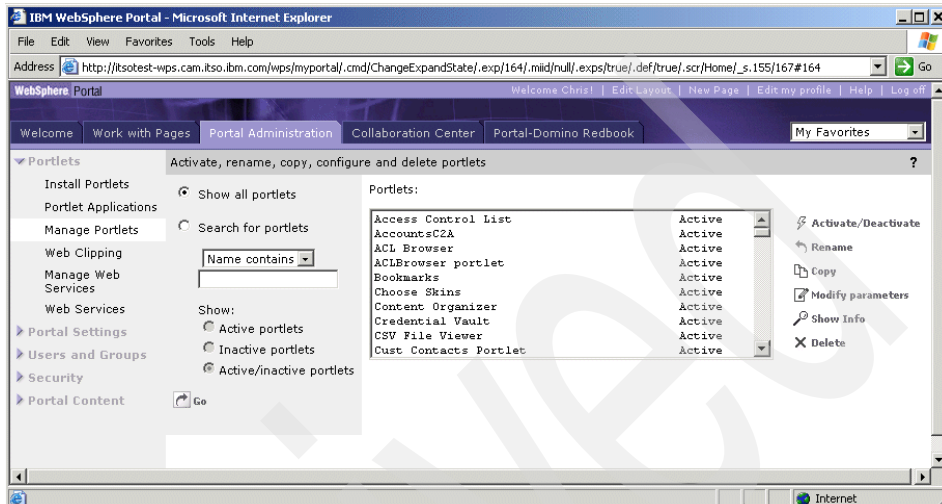


Figure 3-36 The Manage Portlets page under Portal Administration

Navigating down the Portlets scroll bar you find the RSS Portlet entry shown in Figure 3-37.

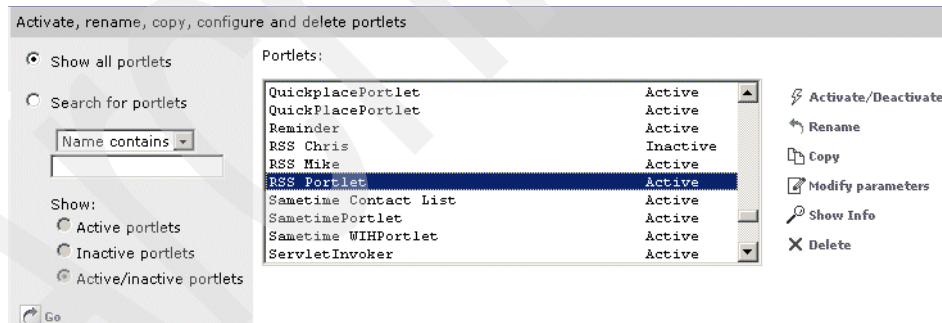


Figure 3-37 Copying the RSS Portlet

Highlight this entry and click the Copy icon from the actions on the right side of the page. The script prompt dialog box pops up and requests the name of the new "copied" portlet. Figure 3-38 shows this prompt.

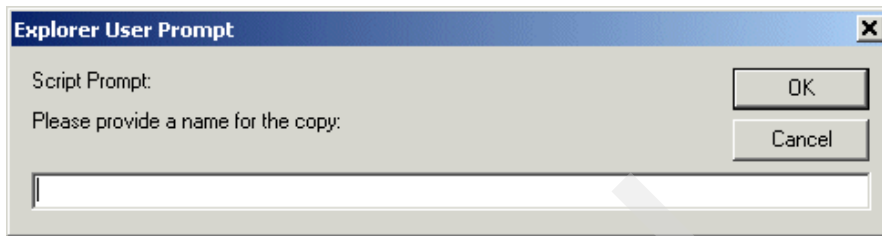


Figure 3-38 Prompt for name of the copied portlet

Enter a name for the copy and click OK; we entered Customer Contacts RSS Portlet. Clicking the OK button adds the portlet to the Portlets area. Scroll through the available portlets and find your newly copied portlet. You will notice that it is Inactive. Highlight this entry and click the Modify Parameters icon from the actions on the right side of the page. The configuration page for this new portlet is displayed, as shown in Figure 3-39.

Configure parameters and titles ?

Save | **Cancel**

Portlet Name: Customer Contacts RSS Portlet

Edit Parameters:

Parameter	Value	
<input type="checkbox"/> stylesheet.wml	/WEB-INF/xml/wml/rss.xml	X Delete
<input type="checkbox"/> stylesheet.html	/WEB-INF/xml/html/rss.xml	
<input type="checkbox"/> url	http://itsotest-dom:8088/apps/CL	
<input type="checkbox"/> itemDisplayed	10000	
<input type="text"/>	<input type="text"/>	+ Add

Edit Locale Specific Titles:

Locale	Title	<input checked="" type="checkbox"/> Set title for selected locale
<input type="radio"/> Simplified Chinese	RSS portlet	
<input type="radio"/> English	Customer Contacts - Using RSS	
<input type="radio"/> Hebrew	טלפון RSS	
<input type="radio"/> Polish	Portlet RSS	
<input type="radio"/> Italian	Portlet per RSS	
<input type="radio"/> Brazilian Portuguese	Portlet RSS	
<input type="radio"/> Czech	Portlet RSS	
<input type="radio"/> French	Portlet RSS	
<input type="radio"/> Turkish	RSS Portal Uygulamacıđı	
<input type="radio"/> German	Portlet für RSS	
<input type="radio"/> Traditional Chinese	RSS Portlet	
<input type="radio"/> Spanish	Portlet RSS	
<input type="radio"/> Japanese	RSS ポートレット	
<input type="radio"/> Korean	RSS Portlet	

Save | **Cancel**

Figure 3-39 Configuration screen for the Customer Contact RSS portlet

Click Save to save the settings and exit the configuration screen. Click Cancel to exit the configuration page without saving any information. There are two sections on this configuration page, Edit Parameters and Edit Locale Specific Titles. The Edit Parameters section is shown in Figure 3-40.

Portlet Name: Customer Contacts RSS Portlet
 Edit Parameters:

Parameter	Value	
<input type="checkbox"/> stylesheet.wml	<input type="text" value="/WEB-INF/xsl/wml/rss.xsl"/>	<input type="checkbox"/> X Delete
<input type="checkbox"/> stylesheet.html	<input type="text" value="/WEB-INF/xsl/html/rss.xsl"/>	
<input type="checkbox"/> url	<input type="text" value="http://itsotest-dom:8088/apps/Cl"/>	
<input type="checkbox"/> itemDisplayed	<input type="text" value="10000"/>	
<input type="text"/>	<input type="text"/>	<input type="button" value="+ Add"/>

Figure 3-40 Edit Parameters section

Details for the parameters shown are:

- ▶ Edit Parameters:
 - Adding a parameter: To add a parameter, enter the parameter name in the text box below Parameter. To add the corresponding value, enter the value in the text box below Value and then click Add.
 - Deleting a parameter: To delete parameters, mark the checkboxes to the left of each parameter you would like to delete. When finished, click Delete. A confirmation dialog appears. Click OK to delete the parameter. Click Cancel to keep the parameter.
 - Our configuration values are listed in Table 3-11.

Table 3-11 Configuration values for the Lotus Notes View portlet example

Field	Inserted value
stylesheet wml (using the WP built-in default)	/WEB-INF/xsl/wml/rss.xsl (default)
stylesheet html (using the WP built-in default)	/WEB-INF/xsl/html/rss.xsl (default)
URL (this is the view that we generated in Domino)	http://itsotest-dom:8088/apps/CUSTOMER.NSF/ContactsRSSbyName?openview&count=9999
itemDisplayed	10000 (records to show)

- ▶ Edit Locale Specific Titles:
 - To set local titles, click a locale in the Edit Locale Specific Titles area. Click the Set title for selected locale icon to display the title page as shown in Figure 3-41 on page 117. After entering your title, click OK. Click Close to exit this page without saving any information.

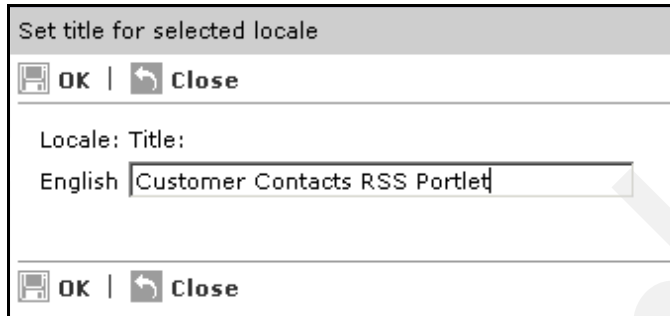


Figure 3-41 Setting the title for selected locale

- Click OK on the bottom of this edit screen to complete the configuration and return to display mode. If you click Previous on this edit screen instead, you will return to the first edit screen. Clicking Cancel will exit the edit screens without saving any configuration changes.

After saving the modifications to the Customer Contacts RSS Portlet, activate the portlet. Highlight the portlet and click the Active/Deactivate icon from the actions on the right side of the page. This activates the inactive portlet.

Follow similar instructions for the second portlet in this example, the Customers XML-XSLT portlet. Again highlight the RSS portlet in the Portlets area and click Copy. Type in the new name and clicking OK. Highlight this inactive portlet and click Modify Parameters. In the lab we entered the values shown in Table 3-12 into the parameter list, and our title in the English locale.

Table 3-12 Configuration values for the Lotus Notes View portlet example

Field	Inserted value
stylesheet wml (this is the customer.xsl form we built in Domino)	http://itsotest-dom:8088/apps/CUSTOMER.NSF/customers.xsl?readform
stylesheet html (this is the customer.xsl form we built in Domino)	http://itsotest-dom:8088/apps/CUSTOMER.NSF/customers.xsl?readform
URL (this is generating XML from the customersbyname view from Domino)	http://itsotest-dom:8088/apps/CUSTOMER.NSF/customersbyname?readviewentries&count=1000
itemDisplayed	8 (records to show)

The result of our actions is shown in Figure 3-42.

Configure parameters and titles ?

Save | Cancel

Portlet Name: Customers XML-XSLT

Edit Parameters:

Parameter	Value	
<input type="checkbox"/> stylesheet.wml	MER.NSF/customers.xml?readform	✕ Delete
<input type="checkbox"/> stylesheet.html	MER.NSF/customers.xml?readform	
<input type="checkbox"/> url	me?readviewentries&count=1000	
<input type="checkbox"/> itemDisplayed	8	
<input type="text"/>	<input type="text"/>	+ Add

Edit Locale Specific Titles:

Locale	Title	<input checked="" type="checkbox"/> Set title for selected locale
<input type="radio"/> Simplified Chinese	RSS portlet	
<input type="radio"/> English	Customers XML-XSLT	
<input type="radio"/> Hebrew	RSS פורטלט	
<input type="radio"/> Polish	Portlet RSS	
<input type="radio"/> Italian	Portlet per RSS	
<input type="radio"/> Brazilian Portuguese	Portlet RSS	
<input type="radio"/> Czech	Portlet RSS	
<input type="radio"/> French	Portlet RSS	
<input type="radio"/> Turkish	RSS Portal Uygulamacıđı	
<input type="radio"/> German	Portlet für RSS	
<input type="radio"/> Traditional Chinese	RSS Portlet	
<input type="radio"/> Spanish	Portlet RSS	
<input type="radio"/> Japanese	RSS ポートレット	
<input type="radio"/> Korean	RSS Portlet	

Figure 3-42 Configuration screen for Customers XML-XSLT portlet

After saving the modifications to the Customers XML-XSLT portlet, activate the portlet. Highlight the portlet and click the Active/Deactivate icon from the actions on the right side of the page. This will activate the inactive portlet.

Results

We have created a page and placed the Customer Contacts RSS portlet and the Customers XML-XSLT portlet on this page and have activated the page. Figure 3-43 on page 119 shows the XML portlets page.

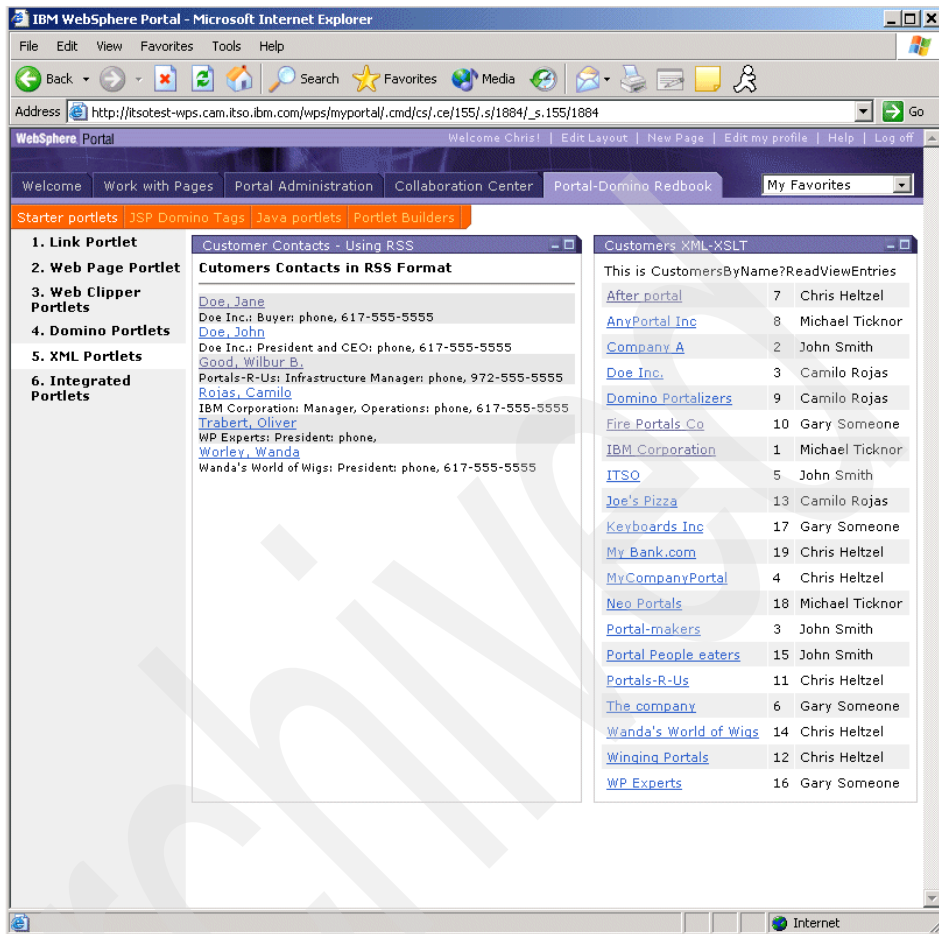


Figure 3-43 XML portlet page

Clicking the first entry in the Customer Contacts - Using RSS portlet results in the screen shown in Figure 3-44.

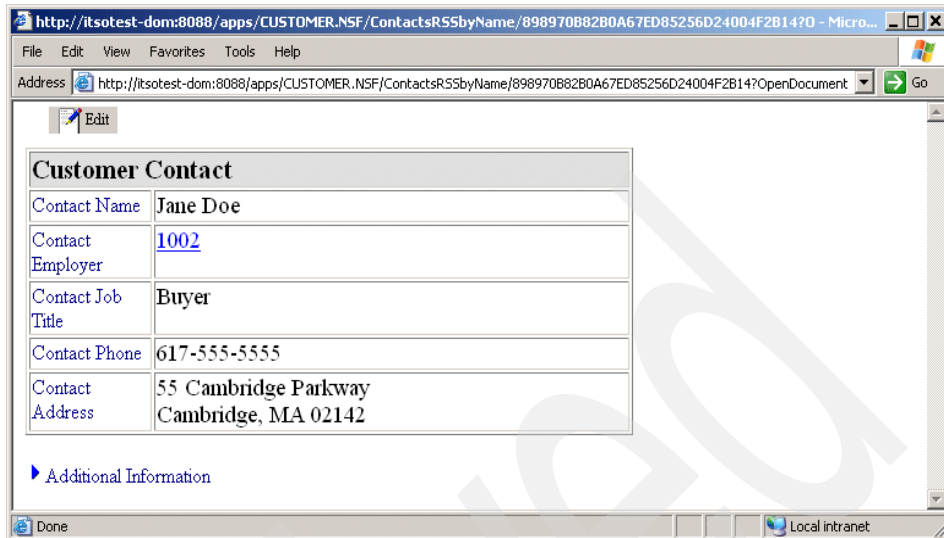


Figure 3-44 Result of clicking a highlighted link in the RSS portlet

Notice that clicking on the document link brought up a new browser window. If you close this window and click the first entry in the Customers XML-XSLT portlet, again, a new browser window is opened. The result is shown in Figure 3-45.

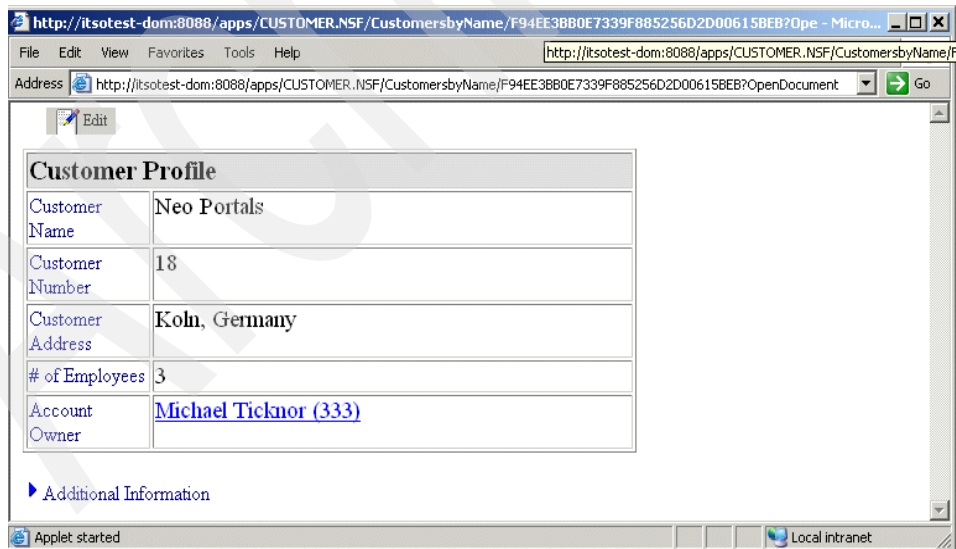


Figure 3-45 Clicking on a highlighted link in the XML/XSL portlet

3.7 Integrate using multiple portlets

One of the advantages of using WebSphere Portal is the single point of access of the portal page. Activities in portlets on the page may or may not be associated with one another. This section deals with using two types of portlets to access and display Domino data on the same portal page. Choosing an item on one portlet triggers events on other portlets.

By combining the Web Clipping and Web Page portlets with some custom JavaScript and Domino development, it is possible to create a basic portal application which implements the integrated portlet pattern without custom Java or JSP development.

3.7.1 Considerations

The main advantage of this integration technique is that it is relatively quick and easy to implement. Its main limitation is that the developer is tied to the features provided by the Web Clipping and Web Page portlets being used.

Applicable portlet patterns	Integrated
Development time	Moderate
Developer skill set	WebSphere Portal administration skills are needed. Domino developer skills, with experience with JavaScript, XML, and XSLT are needed.
Range of applications	Moderate
Handle rich text	No
Performance	Limited
Session Management	No
Clustering	No
Scalability	No
Requires single sign-on	The Web Clipping portlet has extensive firewall, security, and authentication configurations. Web Page portlets can be configured to use single sign-on.
Required software versions	Standard HTML browsers with JavaScript support are required. Netscape 5 and above and Internet Explorer 4 and above are supported.

Implementation details

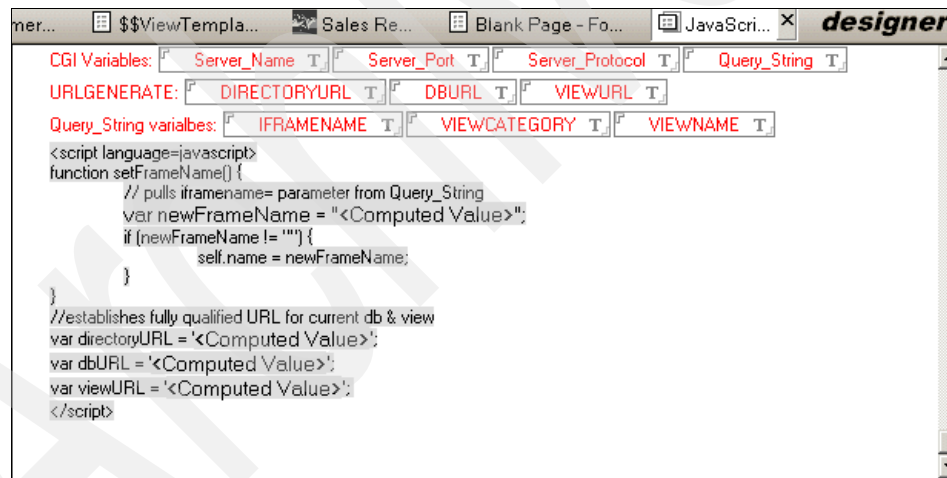
We describe how to use this technique to generate a portlet page with a “Web clipped” Domino view and display corresponding related information in three Web page portlets located on the same page. We discuss the Domino enhancements that are made, followed by the WebSphere Portal configurations that have to be generated. Ultimately, we show you the results of our integration. To fully explain this implementation, we provide complete details concerning:

- ▶ Domino enhancements
- ▶ Portal configuration options
- ▶ Results

Domino enhancements

We didn’t have the necessary elements in our databases, so we had to create them. In this section, we do not discuss in detail how to create these elements in Domino, just their implementation.

First we create a Subform to generate the URL for us. This Subform, shown in Figure 3-46, will be added to other forms.



```
mer...  $$$ViewTempla...  Sales Re...  Blank Page - Fo...  JavaScri...  designer

CGI Variables: [Server_Name T] [Server_Port T] [Server_Protocol T] [Query_String T]
URLGENERATE: [DIRECTORYURL T] [DBURL T] [VIEWURL T]
Query_String variables: [IFRAMEFRAME T] [VIEWCATEGORY T] [VIEWNAME T]

<script language=javascript>
function setFrameName() {
    // pulls iframeName= parameter from Query_String
    var newFrameName = "<Computed Value>";
    if (newFrameName != "") {
        self.name = newFrameName;
    }
}
//establishes fully qualified URL for current db & view
var directoryURL = '<Computed Value>';
var dbURL = '<Computed Value>';
var viewURL = '<Computed Value>';
</script>
```

Figure 3-46 JavaScript Subform in the customer database

This Subform uses the iframeName, viewcategory, and viewname to generate the URL. We will be clipping a view, so we will generate this view element and the accompanying view template form. Figure 3-47 shows the view.

Customer Name	Customer Name
After portal	[Aft
AnyPortal Inc	[Anyl
Company A	[Com
Doe Inc.	[Do
Domino Portalizers	[Do
Fire Portals Co	[Fire
IBM Corporation	[IB
ITSO	[ITS
Joe's Pizza	[Jo
Keyboards Inc	[Key
My Bank.com	[M
MyCompanyPortal	[MyC
Neo Portals	[N
Portal-makers	[Port
Portal People eaters	[P
Portals-R-Us	[P
The company	[T
Wanda's World of Wig	[W
Winging Portals	[W
WP Experts	[W

Figure 3-47 CustomerPortletIntegrated view in the Customers database

The Customer Name view column has a JavaScript link included.

Next we generate the Web view template form for this view. Figure 3-48 shows this view template.

```

ms CustomerPortletIntegrated - View $$$ViewTemplate for CustomerPortlet... designe
|
CGI Variables: Server_Name T Server_Port T Server_Protocol T Query_String T
URLGENERATE: DIRECTORYURL T DBURL T VIEWURL T
Query_String variables: IFRAME_NAME T VIEWCATEGORY T VIEWNAME T
<script language=javascript>
function setFrameName() {
    // pulls iframeName= parameter from Query_String
    var newFrameName = "<Computed Value>";
    if (newFrameName != "") {
        self.name = newFrameName;
    }
}
//establishes fully qualified URL for current db & view
var directoryURL = '<Computed Value>';
var dbURL = '<Computed Value>';
var viewURL = '<Computed Value>';
</script>

<script language=javascript>
function viewLinkClicked(documentUNID, customerNum) {
    var linkStr;
    var targetFrameStr;
    linkStr = viewURL + documentUNID + "?openDocument";
    targetFrameStr = "profiledisplay";
    window.open(linkStr, targetFrameStr);
    linkStr = dbURL +
"DISPLAYVIEWCATEGORY?OpenForm&viewname=customercontactsportletintegrated&vie
wcategory=" + customerNum;
    targetFrameStr = "contactdisplay";
    window.open(linkStr, targetFrameStr);
    linkStr = directoryURL +
"sales.nsf/DISPLAYVIEWCATEGORY?OpenForm&viewname=sabycustomernum&viewcate
gory=" + customerNum;
    targetFrameStr = "salesactivities";
    window.open(linkStr, targetFrameStr);
}
</script>

```

Customer Name	Cust
[After 7	
[AnyPc 8	
[Comp: 2	
[Doe 1 3	

Figure 3-48 \$\$\$ViewTemplate for CustomerPortletIntegrated view

This view template embeds the view CustomerPortletIntegrated, and also incorporates the JavaScript Subform. The viewLinkClicked JavaScript function performs the placement and opening of the Web page portlets. We will display a blank form when the Web page portlets on the portlet page are assessed initially. We create the blank form in the Customer database. Figure 3-49 shows the blank form.

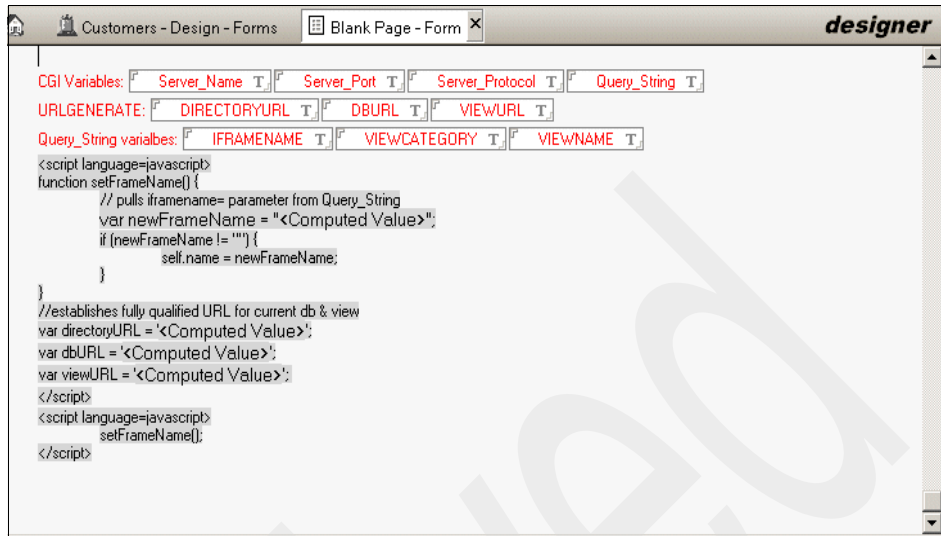


Figure 3-49 Initial blank form

This form includes the JavaScript Subform. This form will be shown in each of the three Web Page portlets when we access the portal page initially. We need to generate the views and forms to provide content for these portlets. The portlets will show the customer profile, the customer contact, and any relevant sales activities associated with this customer. We generate the view CustomerContactsPortletIntegrated. This view is shown in Figure 3-50.

Employer	Contact Name	Phone #	hidden 4
1	Rojas, Camilo	617-555-5555	Camilo Rojas
1001	Trabert, Oliver		Oliver Trabert
1002	Doe, Jane	617-555-5555	Jane Doe
	Doe, John	617-555-5555	John Doe
1234567	Worley, Wanda	617-555-5555	Wanda Worley
500	Good, Wilbur B.	972-555-5555	Wilbur B. Good

Figure 3-50 Customer Contacts portlet integrated view

Next, we generate the view template form for this view. We build a form to display the view called DisplayViewCategory. Figure 3-51 on page 126 shows this form.

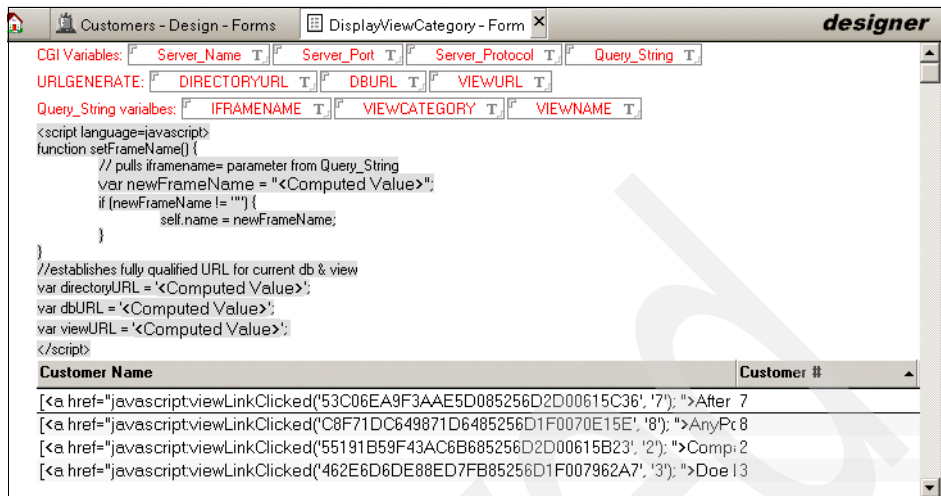


Figure 3-51 DisplayViewCategory form for the Customers database

The JavaScript Subform has been placed in this form, along with an embedded view that is passed with the viewname. For the Sales database, we generate a similar DisplayViewCategory form. This is shown in Figure 3-52.

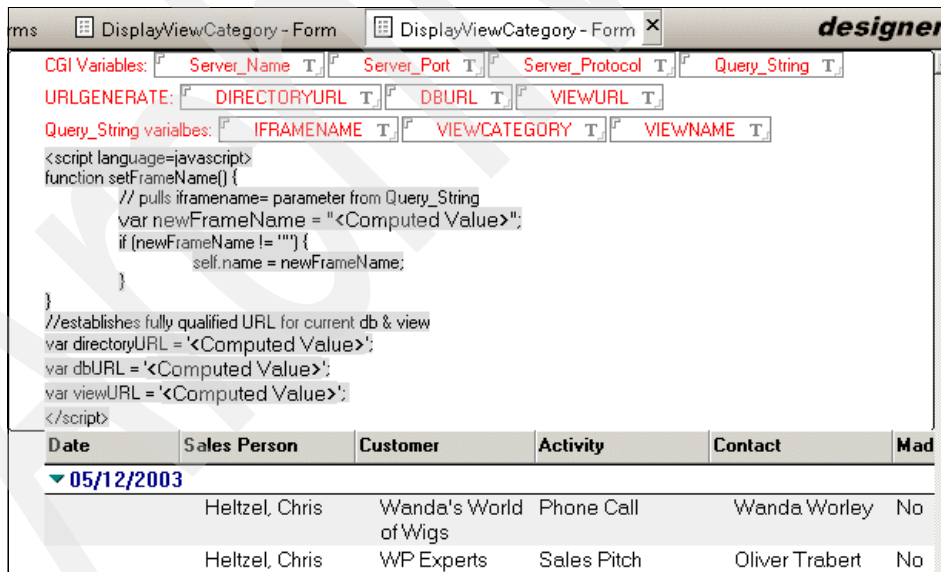


Figure 3-52 DisplayViewCategory form for the Sales database

The JavaScript Subform has been placed in this database as well as the form. We do not have to generate any special views in this database.

Configuration options

We now have to construct our portlets. We start with the Web Clipper portlet. In the portal environment, we click Portal Administration → Web Clipping. This page is shown in Figure 3-53.

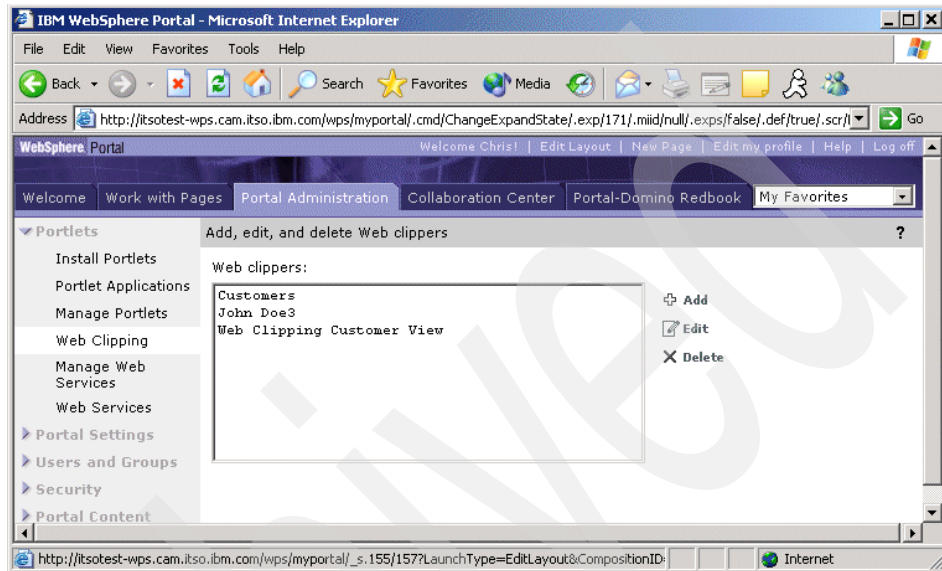


Figure 3-53 Web clipping through Portal Administration

Following the same format as in 3.4, “Integrate using the Web Clipping portlet” on page 79, we click the Add icon and proceed with clipping the customerportletintegrated view in the customer database. Table 3-13 shows the configuration values for this example.

Table 3-13 Configuration values for the Customers “clipped” view

Field	Inserted value
Name and default local title	Customers
Description	(none)
URL to clip	http://itsotest-dom.cam.itso.ibm.com:8088/apps/CUSTOMER.NSF/CustomerPortletIntegrated?OpenView
Connection timeout (seconds)	5
All other fields	default

Since our view has JavaScript embedded in a column, we click the Modify security options icon and uncheck the checkbox next to Remove JavaScript from

clipped content. We click Done on this page and Next on the main configuration page. We now have to select the elements from the URL that we want to keep, as shown in Figure 3-54.



Figure 3-54 "Clipping" from the URL

We highlight our "clip" and click Next. A preview is shown, and we click Done to save it. We have created the portlet. We put this portlet on a page and also add three Web Page portlets. We will place the "clipped" portlet and one Web page portlet in one row and the other two portlets on the second row. Figure 3-55 on page 129 shows a close-up of this portlet page.

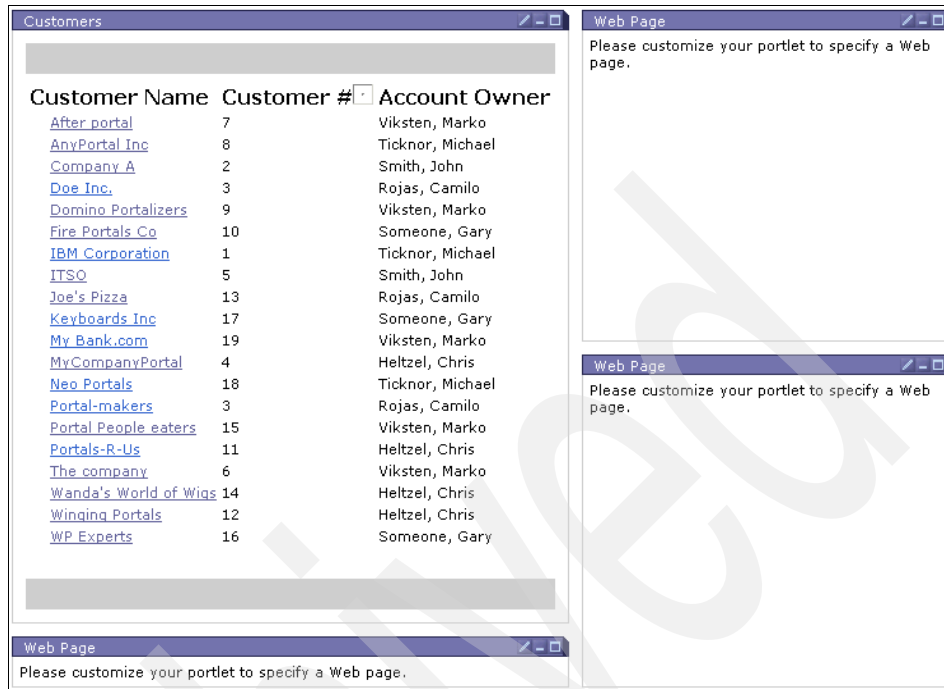


Figure 3-55 Web Page portlets need to be configured

We now configure the three Web Page portlets. We followed the steps described in 3.3, “Integrate using the Web Page portlet” on page 74 to do this. The configuration values for the Web Page portlet in row one, Customer Profile, are listed in Table 3-14.

Table 3-14 Configuration values for the “Customer Profile” Web Page portlet

Field	Inserted value
Portlet title	Customer Profile
URL (This URL opens the blank form in the customer database. It also names this iframe as profiledisplay.)	http://itsotest-dom:8088/apps/customer.nsf/Blank%20Page?OpenForm&iframeaname=profiledisplay
Portlet width	Fit to column
Portlet height	200

The configuration values for the first Web Page portlet in row two, Customer Contacts, are listed in Table 3-15.

Table 3-15 Configuration values for the “Customer Contacts” Web Page portlet

Field	Inserted value
Portlet title	<i>Customer Contacts</i>
URL (This URL opens the blank form in the customer database. It also names this iframe as contactdisplay.)	http://itsotest-dom:8088/apps/customer.nsf/Blank%20Page?OpenForm&iframeName=contactdisplay
Portlet width	Fit to column
Portlet height	200

And finally, the configuration values for the second Web Page portlet in row two, Customer Sales Activities, are listed in Table 3-16.

Table 3-16 Configuration values for the Customer Sales Activities Web Page portlet

Field	Inserted value
Portlet title	<i>Customer Sales Activities</i>
URL (This URL opens the blank form in the customer database. It also names this iframe as salesactivities.)	http://itsotest-dom:8088/apps/customer.nsf/Blank%20Page?OpenForm&iframeName=salesactivities
Portlet width	Fit to column
Portlet height	200

Results

After configuring the portlets, we created a page and placed and configured the four portlets, and we have activated them. Figure 3-56 on page 131 shows the resulting portal page.

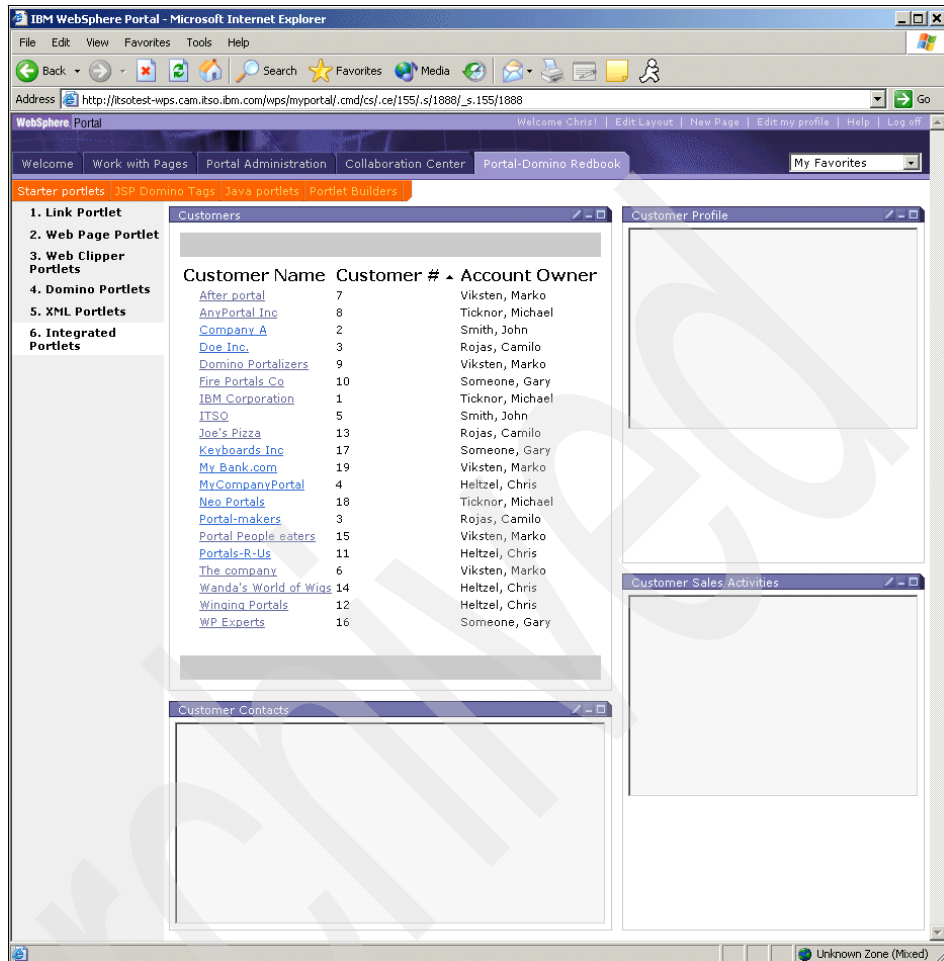


Figure 3-56 The integrated portlets page

The portal page has three portlets that are blank because there is not any information to show yet. In reality, the three portlets are displaying what we want, a blank view from the Domino database. If we click the first highlighted link in the Customers portlet (upper left portlet in Figure 3-56), the resulting screen is shown in Figure 3-57.

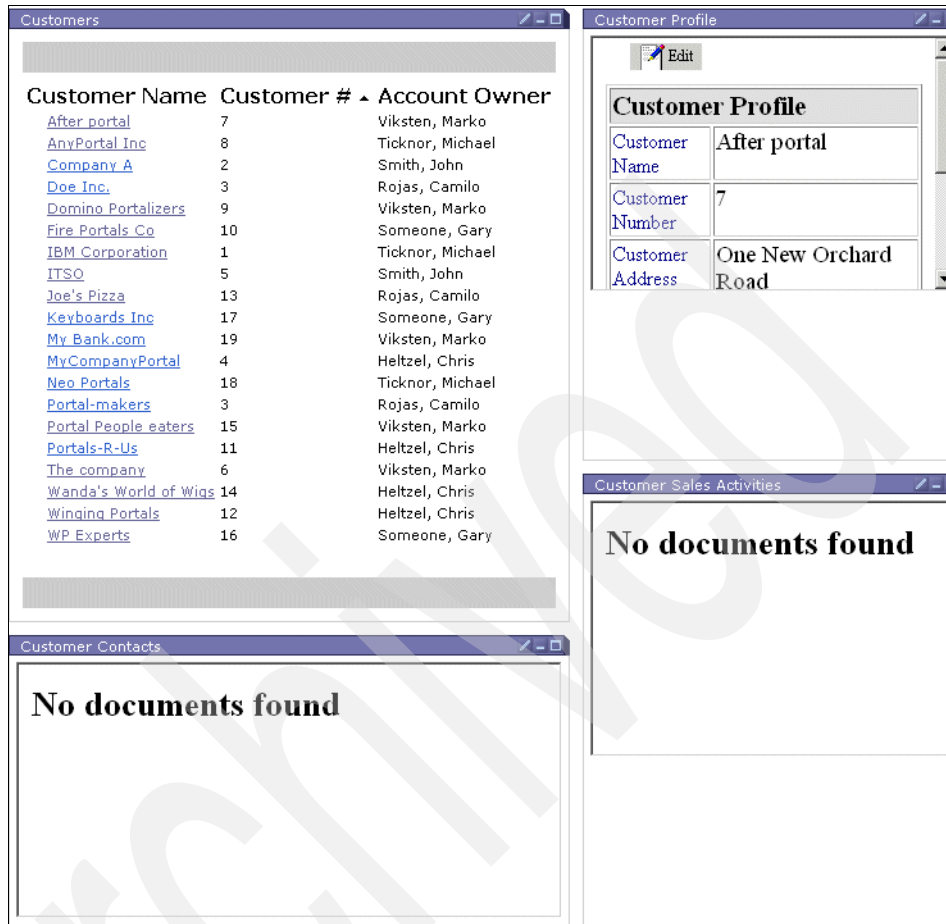


Figure 3-57 Clicking on After Portal link

Notice that two of the remaining three portlets show “No documents found.” This is because there are, in fact, no documents for this entry. A customer profile does exist, as noted in the Customer Profile portlet (upper right portlet in Figure 3-57). Also note that there is a scroll bar even though we have room in that portlet to display more of the document. This is due to our setting of the length in the configuration settings for that portlet. If we click the Fire Portals Co name in the Customers portlet, the screen that results is shown in Figure 3-58. Now the remaining three portlets show relative information. These three remaining portlets are iFrame portlets. All other portlets remain the same even though the document in the Customer Profile portlet is now editable.

Note: You must have the appropriate access to edit the document (at least Author access).



Figure 3-58 Clicking [Fire Portals Co](#) in Customers portlet launches content in other portlets.

If we click the first highlighted document link in the Customer Contacts portlet (lower left portlet), with the portal context and all other portlet contexts remaining the same, the resulting screen displays the contact information, as shown in Figure 3-59.

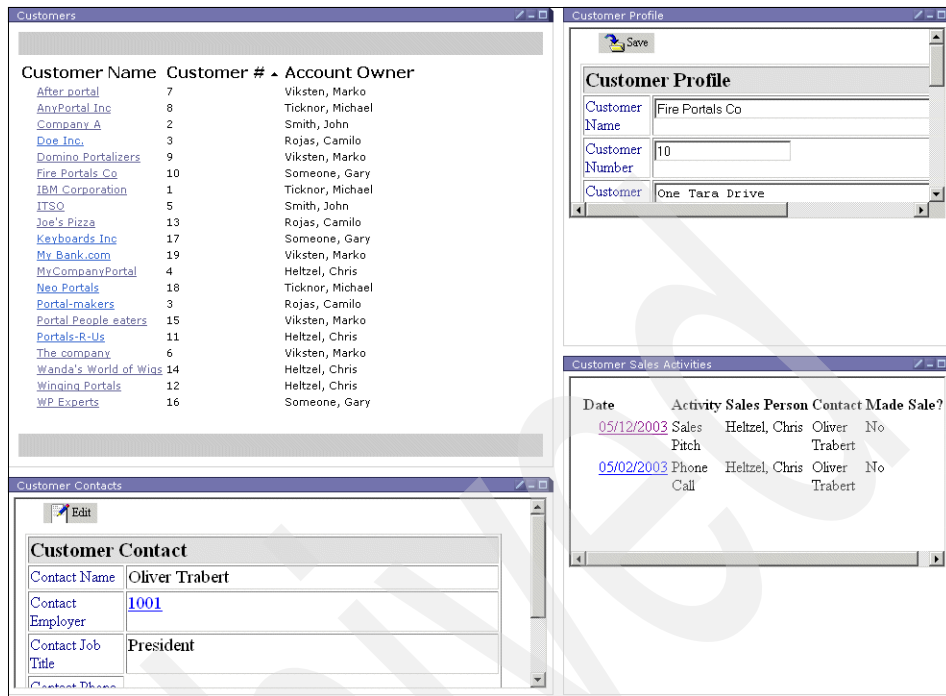


Figure 3-59 Result of clicking on link in the Customer Contact portlet

Finally, if we click the first highlighted document link in the Customer Sales Activities portlet (lower right portlet), the resulting screen is shown in Figure 3-60.

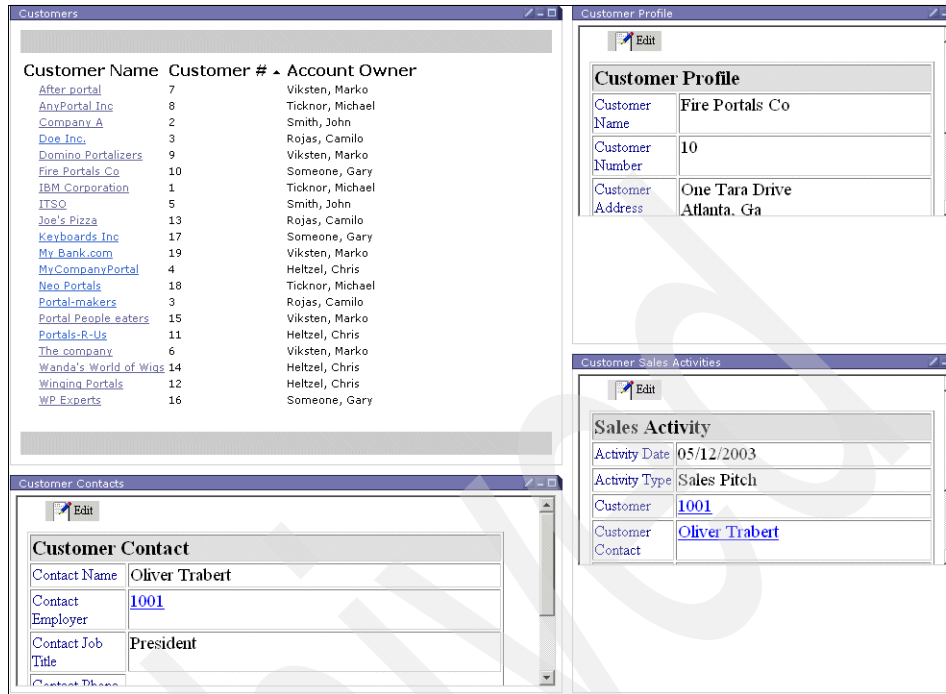


Figure 3-60 Result of clicking on the first highlighted document link in the Customer Sales Activity portlet

Again, this demonstrates that all portlet contexts stay intact and the document is displayed in the Customer Sales Activities portlet context. Finally, we can click the Edit button in the three remaining portlets. This is shown in Figure 3-61 on page 136.

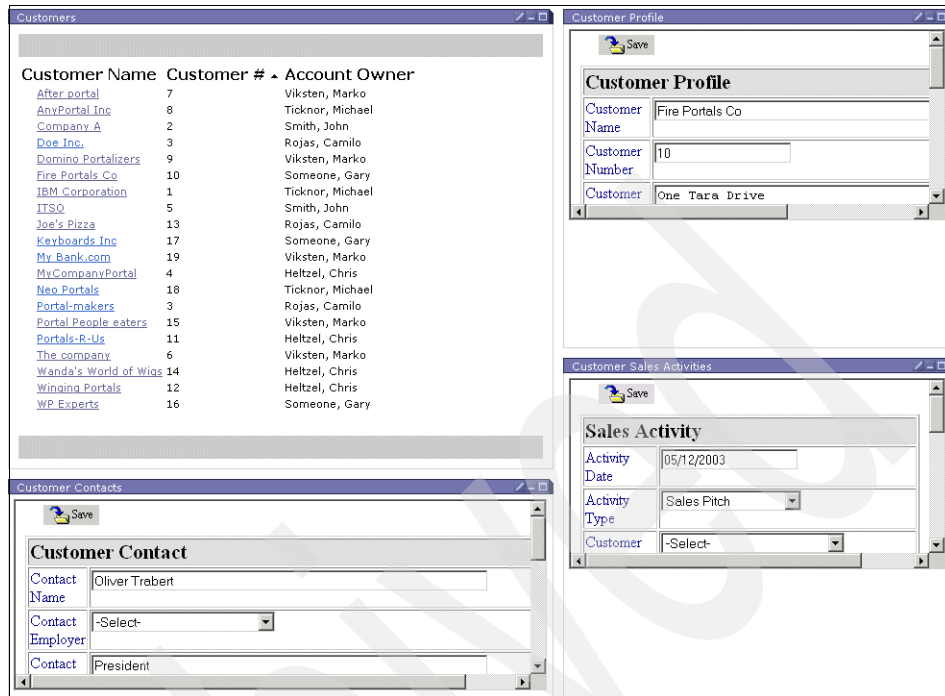


Figure 3-61 Three documents in “edit” mode on the same portlet page

The main advantage of using this technique is that all three documents have been put into “Edit” mode inside their respective portlet contexts.

3.8 Reference material

The following Web sites and redbooks are helpful when exploring the integration of Lotus Domino and WebSphere:

- ▶ *Domino and WebSphere Together, Second Edition*, SG24-5955
- ▶ IBM Lotus Domino Developer Domain
<http://www.lotus.com/1dd>
- ▶ IBM WebSphere Developer Domain - Portal Zone
<http://wilson.boulder.ibm.com/wsdd/zones/portal/>
- ▶ IBM WebSphere Portlet Catalog
<http://www.ibm.com/software/Webservers/portal/portlet/catalog/>

Using custom Domino JSP tag libraries

This chapter describes how to integrate Lotus Domino applications into a WebSphere Portal using the custom Domino JSP Tag libraries, which were introduced in release 6 of the product. We tried our examples with both R5 and Domino 6 databases, and both of them worked well. Be aware that IBM does not support the tag libraries on R5.

We begin the chapter by reviewing several pertinent technologies: J2EE, JSP, and Tag libraries. Next, we introduce the tools needed to accomplish a successful integration. As we describe how we integrated our sample application, we point out some common pitfalls along the way, and how they can be overcome.

Attention: The Lotus Domino JSP Tag libraries in the WebSphere Portal are not formally supported by IBM at this time. This is because there are some tags that require HTML headers on the JSP files. These tags include *nocache* and others that insert JavaScript libraries that are dynamically generated; because portlets serve fragments of HTML code, you should first test these tags in a controlled WebSphere Application Server environment to assess the viability of including them in your portal applications.

4.1 Overview of the Domino custom JSP Tag option

In the previous chapter we reviewed some simple methods of integrating Lotus Domino applications with WebSphere Portal, and found limitations in those methods that need to be lifted for real world implementations. Some of the limitations are addressed in this chapter as we explore the benefits available from technologies like J2EE and tools like WebSphere Studio Application Developer.

One of the most common situations when integrating Lotus Domino applications is that you are faced with a growing number of J2EE applications that should interact with Lotus Domino applications. A basic knowledge of the capabilities J2EE brings to a Lotus Domino developer is important in the current marketplace.

Combining J2EE technology with Lotus Domino in a portal environment helps overcome several challenges, like the ones outlined in the first chapter. For example, with J2EE there is access to transactional services and personalization services that will extend your current Lotus Domino application.

Exposing your current Lotus Domino applications through WebSphere Portal will lift some of the weight that Lotus Domino has to handle as a Web application server.

Integrating Lotus Domino applications with JSP Tag libraries yields shorter development time than that for a custom Java development option. The technology comes with some limitations, especially regarding performance. These limitations are discussed as we explore each integration technique.

You will probably choose this integration option if the functionality of prepackaged portlets isn't enough to fulfill your integration needs, and if there is an underlying motivation to understand what Java technologies bring to Lotus Domino when developing WebSphere Portal applications.

4.2 Technologies involved

The integration techniques described in this chapter rely on several technologies; this section is a brief introduction to them. It begins with a look at J2EE and a description of some of its components, then provides a closer look at JavaServer Pages (JSP) technology and custom JSP Tag libraries.

4.2.1 J2EE overview

The benefits of Java 2 Enterprise Edition (J2EE) and its basic architecture and components are described in this section.

J2EE is a set of synchronized standards and practices that when used together enable solutions for developing, deploying, and managing multi-tier distributed server-centric applications. It is based on the Java 2 Standard Edition (J2SE, formerly called Java Runtime Environment or JRE), which has been complemented with enterprise standards that provide a scalable, robust, secure, and stable platform to build enterprise solutions. It enables developers to extend existing solutions by creating applications quickly, with reduced complexity, cost, and facilities.

A platform for enterprise solutions

Java 2 Enterprise Edition defines a set of standard technologies for developing multi-tier, distributed enterprise applications. J2EE lowers the risk for developers and customers since it's based on a standard and modular architecture. It also can simplify enterprise applications by taking advantage of a complete set of services to help avoid the most common pitfalls of application development, and automatically handling many details of your application without complex programming.

J2EE takes advantage of characteristics such as the “write once, run anywhere” paradigm, which enables Java programs to run on different platforms without requiring rewriting of the entire application. Also, it takes advantage of other well known technologies, like JDBC and CORBA, for interaction with existing enterprise information systems. It is based on a robust security implementation that will protect the data as it is exposed to public networks. Furthermore, J2EE includes a complete set of standards, compliance tests, and documented best practices that assure the stability and portability of applications across different platforms.

Benefits of developing with J2EE

J2EE provides the Lotus Domino developer with the following benefits:

- ▶ J2EE will build enterprise solutions quickly and get robust solutions to market faster.

J2EE is based on containers which provide a clear separation of business components and enterprise services, which focuses developers on writing business logic rather than spending precious time on developing infrastructure code. For example, the Enterprise JavaBeans (EJB) components (implemented by J2EE) handle distributed communication, threading, scaling, transaction management, messaging infrastructure, and lots of others requirements of modern applications. Similarly, Java Servlets

and JavaServer Pages (JSP) simplify Web development by providing a flexible infrastructure for component communication, and session management in Web applications that is integrated with the Web server.

► Standards.

J2EE is a set of technologies that many vendors can implement and extend. Vendors like IBM with its WebSphere Application Server, are free to extend on implementations but not on standards or APIs. Sun Microsystems provides a complete J2EE Compatibility Test Suite (CTS) to grant J2EE interoperability. The J2EE CTS helps ensure compatibility among the application vendors, which helps in ensuring portability of the applications and components written on J2EE. J2EE brings the “write once, run anywhere” paradigm to the server.

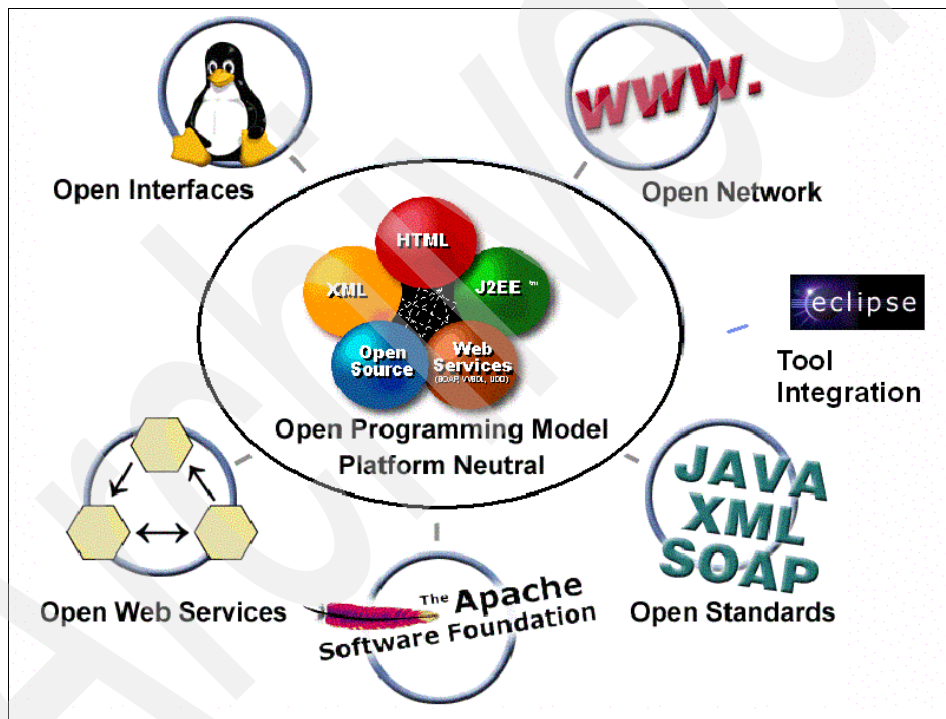


Figure 4-1 IBM WebSphere commitments on Open Standards

► Separation of tiers.

J2EE brings to the Lotus Domino developer the capacity to separate the business logic from the presentation layer. By doing this the developer builds components that are loosely coupled, providing openness to your development environment and reusability to your software.

- ▶ Extended connectivity.

J2EE will enable your applications to connect to external applications through technologies and standards (like the Java Connector Architecture and Java Messaging Service) that let you easily expose current business logic to Web applications, and that can serve different clients (Web browsers, cell phones, pervasive devices, voice systems), offering the basis of a multichannel e-business infrastructure. J2EE also offers CORBA support for external systems through remote method calls.

J2EE offers a platform with faster solution development time, freedom of choice, code reuse and extended connectivity. J2EE helps developers and customers to reduce Total Cost of Ownership (TCO), time to market, and take advantage of the evolving standards communities, while avoiding locking into a single provider for the infrastructure needed to build an e-business platform.

Technologies included in J2EE

Some of the technologies in the J2EE platform are:

- ▶ JavaServer Pages (JSPs)
- ▶ Java Servlets
- ▶ Enterprise JavaBeans (EJBs)
- ▶ J2EE Connector Architecture
- ▶ Java Management Extensions (JMX)
- ▶ Java API for XML Registries (JAXR)
- ▶ Java API for XML-Based RPC (JAX-RPC)
- ▶ Java Message Service (JMS)
- ▶ Java Naming and Directory Interface (JNDI)
- ▶ Java Transaction API (JTA)
- ▶ CORBA
- ▶ JDBC data access API

For updated standards and levels for each standard, check with the J2EE official Web site.

J2EE focus on portlet development for Domino developers

One of the main advantages and strengths that J2EE brings from a design point of view is the separation of the tiers or layers that compose an e-business application. Unlike a Lotus Domino application where all the data, business logic, and presentation objects are packed on a single .NSF file, in the J2EE design

pattern there are several separate containers which represent different logical tiers, as depicted Figure 4-2. A brief explanation of these tiers follows.

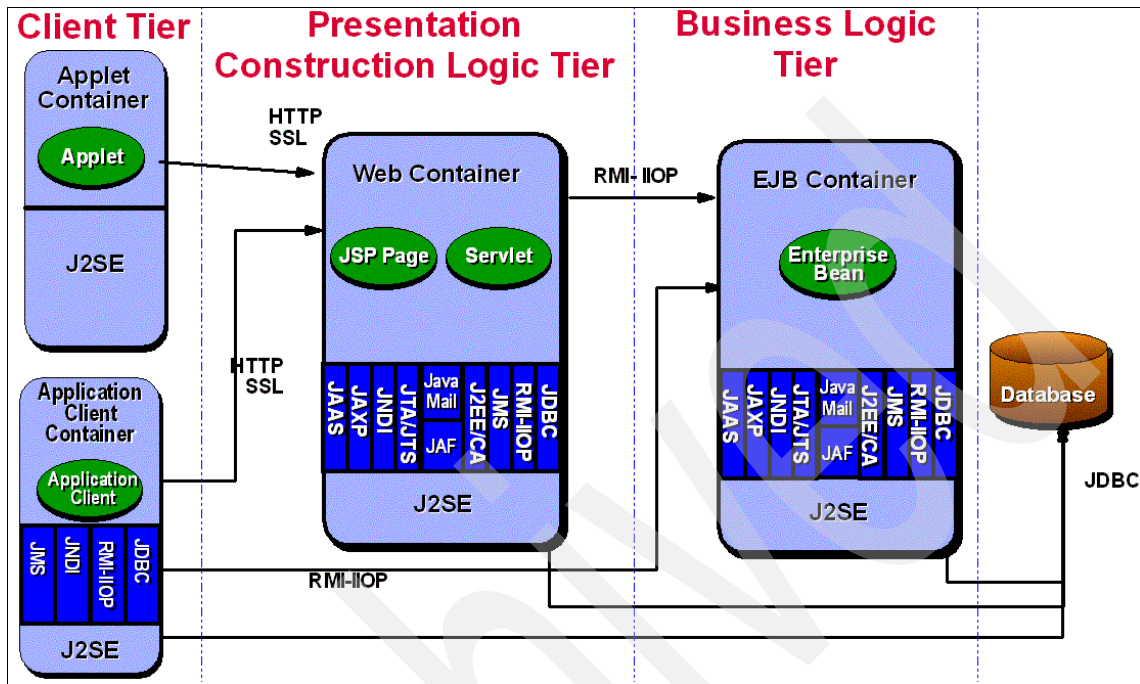


Figure 4-2 J2EE design

Database tier

Viewing the figure from right to left, the first tier is the Database tier. This tier is not included in J2EE and is accessed directly by a technology called JDBC, which provides access to the application data stored on relational databases. From a J2EE developer point of view this tier is basically a provider of data access and storage.

EJB container tier

Next is the EJB container tier, which is a J2EE runtime component. You would place business logic here by using Java components called Enterprise Java Beans (EJB), which actually are designed to provide seamless access to relational databases and enterprise information systems (EIS) with transactional integrity. Along with the data storage and access facilities, you can embed the business logic of your application in these components since they offer high performance and availability mechanisms that can be accessed from different channels of communication, not just through the Web.

Web container tier

The next tier is the Web container (also a J2EE runtime component), which is designed to construct presentation components and control presentation logic. It is based on technologies that offer a high performance, extendible presentation framework to build upon, like Servlets and JavaServer Pages. The Web container is also the host of other technologies, such as Web Services and XML-XSLT applications, that enable your applications to extend to other channels of interaction.

Client tier

The final layer is the client tier, which is represented by two types of client. The most commonly used is the thin client Web browser, which basically understands and renders the HTML or XHTML broadcast from the Web container into Web pages. Inside the browser you can have a Java component called Applets, which as shown in the diagram can only access the J2SE basic services. Also there is another type of client, called the J2EE Client container, because J2EE allows a client-server configuration. This fat client has access to several J2EE services; it is used when the use of external devices and usability issues arise that the Web Browser is unable to attend.

Additional thoughts on J2EE

The Web container and the EJB container are the two main blocks that construct a J2EE application server. The WebSphere Portal allows the construction of Portlet applications that will reside on the Web container, so our focus remains on this container as we explore the JSP - Domino custom JSP tag libraries option of portalizing Lotus Domino applications.

The Portlet components extend the Servlet component, and can present its information using JSP pages. And since JSP development is part of the Lotus Domino development environment, it seems natural to understand and unleash the power of JSPs in the portal environment.

First, let's take a closer look to the JavaServer Pages (JSP) component.

4.2.2 JavaServer Pages

A JavaServer Page, or JSP, is a simple but very powerful component of J2EE. It allows you to easily create content for Web applications while maintaining the ability to include information, generated on the fly, that can be computed by Java code or extracted from other third party systems like, in our case, Lotus Domino applications.

The JSP allows the developer to mix the robust advantages of the Java language and the visual facilities of HTML editors like WebSphere Studio.

Definition of a JSP

A JSP is a standard file that is written basically in HTML, which can include special tags to let developers include dynamic content for Web applications. From a Java point of view it extends the Servlet component used in most J2EE Web applications. The JSP files end with a .JSP extension.

The following code snippet contains the code of a simple JSP file.

Example 4-1 JSP sample snippet

```
<html>
<body>
<% out.println("Hello World !!"); %>
</body>
</html>
```

As you can see, this snippet looks much like a standard HTML file, with some special tags that contain Java code inside. The output of this JSP will be a Hello World !! message on the Web browser.

The JSP file is stored on the file system in a text format and it is compiled when the first request is made through a Web browser. Figure 4-3 illustrates the steps a JSP goes through during that first request.

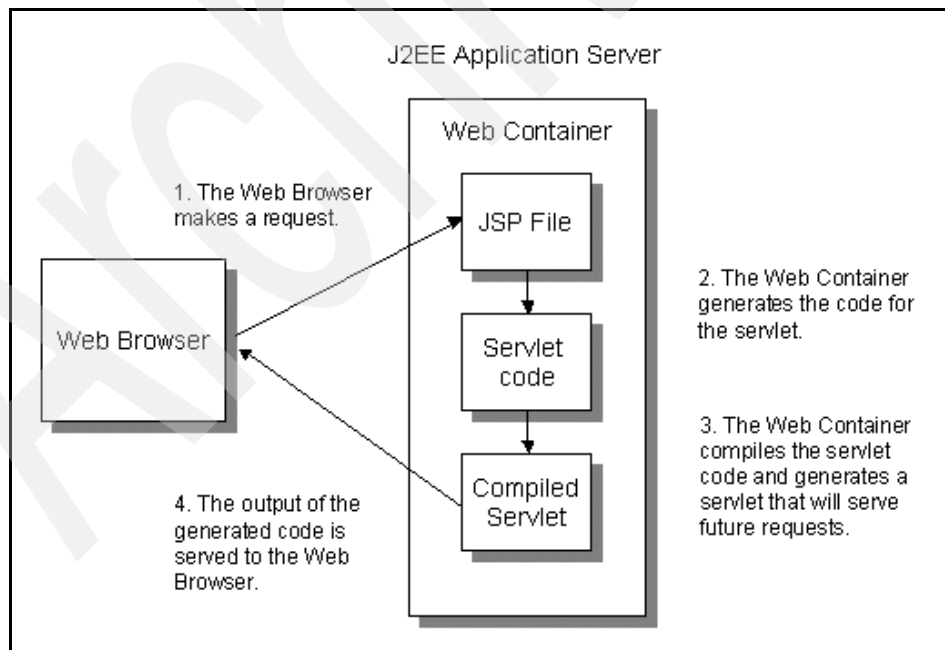


Figure 4-3 Process of serving the first JSP request

As you can see, the JSP is compiled at runtime into a Servlet, and any subsequent request is served by the compiled Servlet. Most J2EE application servers, like WebSphere Application Server, can check on a preset schedule whether the original JSP has changed; if there is a change, the JSP file is recompiled.

There are several elements in a JSP file, one of which will help us include JSP Tag libraries. The elements included in a JSP file are briefly described in the following section.

Elements of a JSP

The JSP files can embed Java code through the use of special tags. These tags can be grouped into four main categories. In this section we describe the importance and use of each.

Directives

The JSP *Directives* are elements that give the J2EE Application Server global information on the JSP file. The syntax of the JSP Directive is:

```
<%@ directive attribute="value" %>
```

The attribute and value pair are optional. Table 4-1 identifies some of the directives.

Table 4-1 JSP Directives

Directive	Description	Available attributes
page	Defines information that affects the global definition of the JSP file. The attributes are not required since they are set by default.	autoFlush buffer contentType errorPage extends info import isErrorPage isThreadSafe language session
include	This directive let's the developer include files in the JSP at compile time.	file
taglib	Lets the JSP include custom tag libraries on the JSP.	uri prefix

We used the taglib directive in our JSP examples to include the custom Domino JSP tags.

For further information on the JSP directives and attributes refer to the references at the end of this chapter.

Declarations

Declarations are used when a JSP developer wants to include methods or define variables on the Servlet class generated by the J2EE Web Container.

Example 4-2 is a sample of a String variable declaration inside a JSP file.

Example 4-2 Sample variable declaration inside a JSP file

```
<html>
<body>
<%! String myString=new String("My String");%>
</body>
</html>
```

Example 4-3 shows the declaration of a method inside a JSP file.

Example 4-3 Sample method declaration inside a JSP file

```
<html>
<body>
<%! public getMyString() { return myString; }%>
</body>
</html>
```

Expressions

Expressions are used to display dynamic Java content on the Web browser. These are some of the most commonly used elements in JSP files. The expression tags are replaced by the evaluation of the Java code; this evaluation is performed at runtime.

The syntax of an expression is:

```
<%= expression%>
```

The expression evaluated must successfully return a String class or a class that can be casted into a String.

Example 4-4 is a sample of a JSP expression.

Example 4-4 Sample JSP expression

```
<html>
<body>
<%=this.getMyString()%>
```

```
</body>
</html>
```

Scriptlets

Scriptlets are general purpose Java tags within which a developer can embed normal Java code.

The syntax of the scriptlets is:

```
<% javaCode %>
```

Example 4-5 is an example of a JSP Scriptlet.

Example 4-5 Sample JSP scriptlet

```
<html>
<body>
<% out.println("Hello World !!"); %>
</body>
</html>
```

Implicit objects in JSP files

Working with JSP files, you will be able to reference—without declaring—several Java objects. These objects have exactly the same characteristics as if they were being called from a Servlet component. Table 4-2 outlines the most important objects.

Table 4-2 Some important implicit objects in JSP files

Object name	Description
session	This object represents the <code>javax.servlet.http.HttpSession</code> object and is most commonly used to store data that's related to the user, through the user interaction with the application. One of the most important uses of this object is to replace the stateless nature of HTTP by providing a temporary stateful storage of the information the user generates through his interaction with the Web application.
request	This object represents the <code>javax.servlet.http.HttpServletRequest</code> interface and is primarily used to allow the user to access the request parameters through the <code>getParameter(String)</code> method. Also this object is used frequently to pass attributes from one request to the target JSP file. It also exposes through different methods the requesting user and device information that is trying to access the JSP file.

Object name	Description
response	This object represents the <code>javax.servlet.http.HttpServletResponse</code> and is responsible for passing information back to the requesting client. It is commonly used to write information to the output stream, although the <code>out</code> object takes care of that.
out	This object represents a <code>JspWriter</code> which is derived from the <code>java.io.Writer</code> and provides the client the ability to stream information back to the requesting client. One of the most common methods used by this object is the <code>out.println(String)</code> method, which prints the <code>String</code> parameter directly on the output generated by the JSP file.

For further information on JSP implicit objects, check the references at the end of this chapter.

Providing a stateful interaction through a stateless protocol

The nature of the HTTP protocol is *stateless*, meaning it basically accepts a request through a TCP/IP port and responds with a bytestream of characters that are rendered on the Web browser. To compensate for this shortcoming, the J2EE technology includes objects that can be used by our JSPs which are implicit on their use and solve the stateless characteristics of the HTTP protocol. In the previous section we discussed an implicit object called *session*. This object represents the `javax.servlet.HttpSession` class; its purpose is to maintain a temporary storage area that can be used by our applications to keep the context of who is interacting with our application and with what characteristics this interaction is taking place.

As shown in Figure 4-4 on page 149, there are two techniques used to enable this interaction. The first is to maintain a Session ID on a cookie inside the browser. This cookie contains the ID code that the Web container can use to look up the session data in which the current user is working. The second approach is rewriting the URL, which appends the Session ID to the URL to maintain the session-browser relationship. This approach is not supported by our portal infrastructure since we could have a session for each portlet.

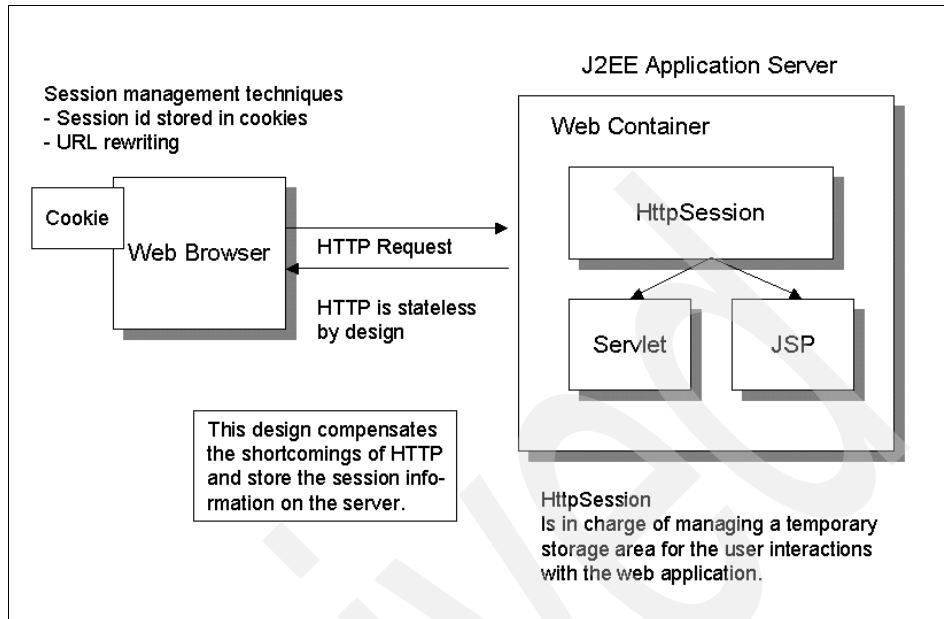


Figure 4-4 Stateful interactions using Java components

The use of the session object inside our JSPs will enable us to maintain portlet-specific information through the user interaction with the portal. To illustrate this, imagine you have a portlet that extracts user information, but then you change to another WebSphere Portal place or page to interact with other portlets. When you come back to the original portlet it would be useful to still get the same information as you had before. This can be accomplished using the session object.

JSP limitations

The JSP technology has one main limitation that should be taken into account: the size of the Java bytestream is limited to 64k.

4.3 Software and tools used

As we discuss the examples, we refer to several tools available to developers. Some of them are:

- ▶ Lotus Domino Designer 6
- ▶ WebSphere Studio Application Developer 5
- ▶ WebSphere Portal Toolkit for WebSphere Studio
- ▶ Lotus Domino Toolkit for WebSphere Studio

Since we discussed the Lotus Domino Designer in previous chapters, we will focus on reviewing the other tools available in this chapter.

4.3.1 WebSphere Studio Application Developer 5

WebSphere Studio Application Developer is one of the WebSphere Studio family of products that has been developed based on the Eclipse Workbench.

The Eclipse Workbench is an open source development platform, designed by IBM and released to the open source community. It is an open, portable, universal tooling platform that provides frameworks, services and tools for building tools.

In essence the workbench provides the tool infrastructure. With this infrastructure in place, the tool builders are able to focus on the actual building of their tools. The workbench has been designed for maximum flexibility to support the development of tools for new technologies that may emerge in the future.

Development tools written for the workbench should support a role-based development model, in which the outcomes of the developers' work will be consistent. The developers should not have to be concerned with how different individual tools may be treating their files.

The WebSphere Studio family of products is an integrated platform (IDE) for developing, testing, debugging, and deploying J2EE applications. It provides support for each phase of the application development life cycle.

WebSphere Studio Application Developer 5 includes the following tools:

- ▶ Web development tools
- ▶ Jakarta Struts development tools
- ▶ Enterprise Java Beans development tools
- ▶ Relational database tools
- ▶ XML tools
- ▶ Java development tools
- ▶ Web services development tools
- ▶ Team collaboration tools
- ▶ Integrated debugger for JSPs, Servlets, EJBs and Java code
- ▶ Server tools for testing and development
- ▶ Performance profiling tools
- ▶ Plug-in development tools

- ▶ Integration with automated build tools like ANT
- ▶ Unit test tools like JUnit

The WebSphere Studio Application Developer has many features to assist the developer; some are illustrated in Figure 4-5. Within the framework there are development perspectives that let you invoke the tools without losing the reference to the working object. In addition, the framework is extendable, allowing you to plug in other tools like WebSphere Portal Toolkit for WebSphere Studio and Lotus Domino Toolkit for WebSphere Studio.

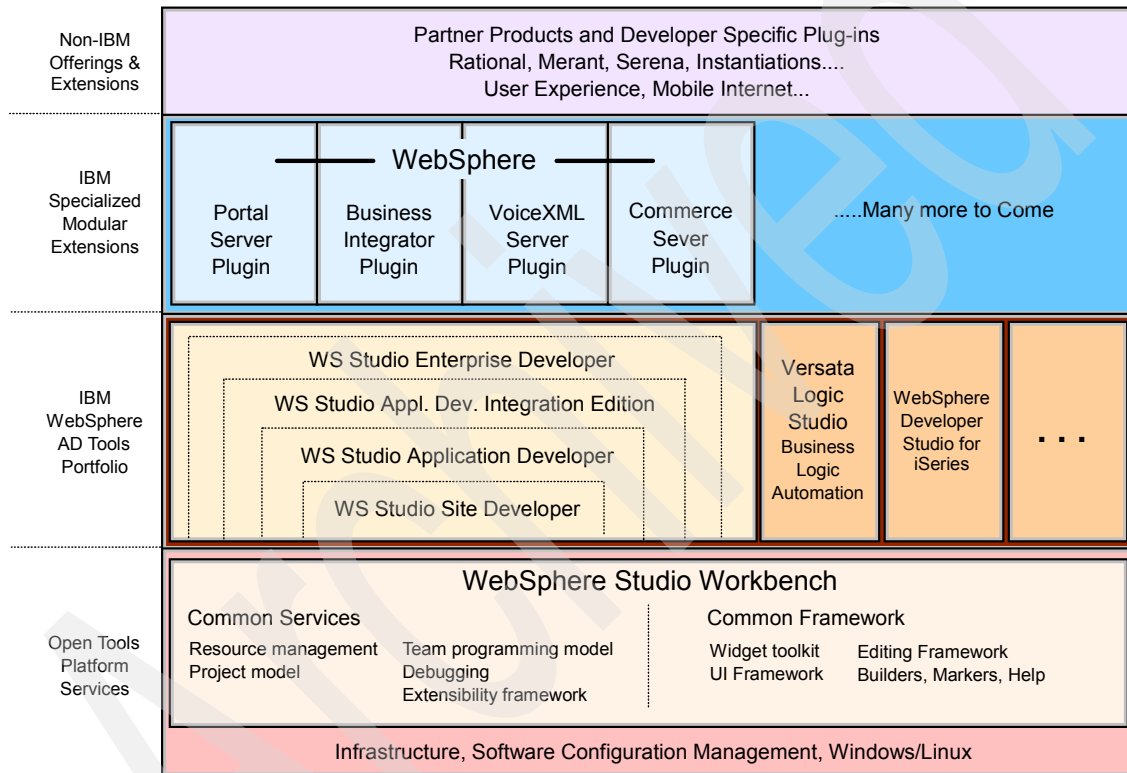


Figure 4-5 WebSphere Studio internal structure

As the figure shows, the WebSphere Studio tool is constructed of building blocks. The cornerstone of the structure is the Eclipse Framework, and on top of it IBM WebSphere Studio Workbench is the basic framework on which several flavors of the WebSphere Studio products are supported.

The first item in the WebSphere Studio portfolio is WebSphere Studio Site Developer, which provides support for creating Web applications and Web

services projects and utilizes the broad number of services provided by the WebSphere Studio Workbench.

Next is WebSphere Studio Application Developer, which gives the developer a full J2EE environment for developing enterprise applications.

On top of it, the WebSphere Studio Application Developer Integration Edition delivers a full environment for developing JCA adapters and enterprise services provided by the WebSphere Application Server Enterprise Edition.

Finally, the top of the line tool is the WebSphere Studio Enterprise Developer, which gives the traditional Cobol, CICS® developer an environment for developing legacy applications, along with the new EGL language, which helps develop enterprise applications on a 4GL environment.

The WebSphere Studio family comes with excellent tools to visually build JSP interfaces and construct visually the logic of Web applications using the Jakarta Struts framework. The Struts framework and how it fits in the Portal environment is discussed in the next chapter.

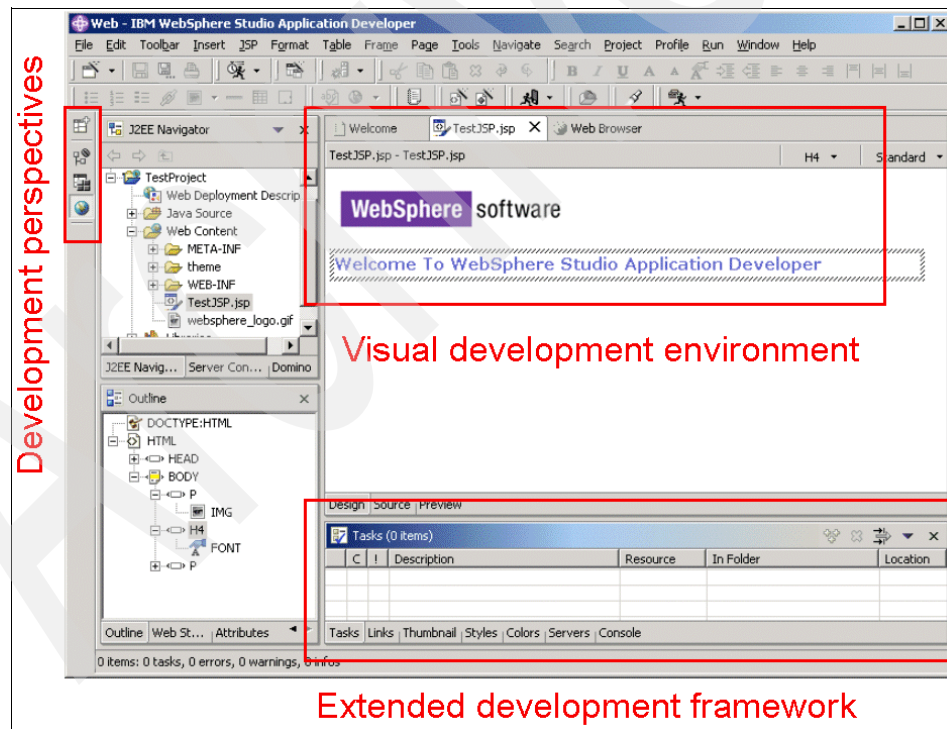


Figure 4-6 Snapshot of WebSphere Studio Application Developer

For more information on WebSphere Studio Application Developer 5 refer to the references at the end of this chapter.

4.3.2 WebSphere Portal Toolkit for WebSphere Studio 4.2.5

The WebSphere Portal Toolkit is designed to help you develop portlet applications for WebSphere Portal Version 4.2 and later versions. Everyplace® Toolkit is designed to help you develop mobile e-business applications for WebSphere Everyplace Access Version 4.2 and later versions, in addition to aiding portlet application development. Portal Toolkit or Everyplace Toolkit provides the following tools for use in developing your portlet applications:

- ▶ Portlet project
- ▶ Portlet application examples
- ▶ Portlet perspective
- ▶ New portlet application wizard
- ▶ portlet.xml editor
- ▶ Portal configuration
- ▶ Exporting to the Web Archive (WAR) file

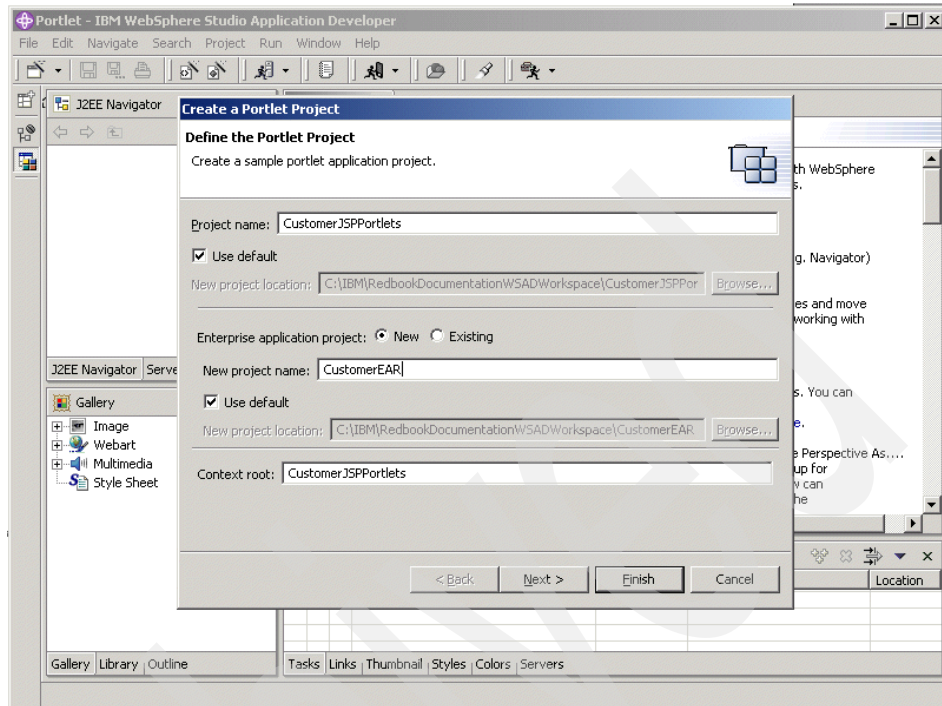


Figure 4-7 WebSphere Portal Toolkit for WebSphere Studio

For the latest information on the toolkit, and for downloadable upgrades and fixes, check the following Web site:

<http://www.ibm.com/websphere/portal/toolkit>

4.3.3 Lotus Domino Toolkit for WebSphere Studio 1.0

The Lotus Domino Toolkit for WebSphere Studio is a plug-in for WebSphere Studio 5. Currently the Lotus Domino Toolkit for WebSphere Studio ships with Lotus Domino Designer 6.0.2. It allows you to add Domino 6 custom tags to Java server pages, providing a simple way to blend Domino and J2EE applications. JSP tags are XML tags embedded in a JSP providing data access, data input, and process control. The tags abstract the Domino objects for Java (Domino Java API) and provide quick development turnaround for building J2EE applications that use Domino data and services.

The toolkit provides a dedicated pane in WebSphere Studio where you open the Domino database you wish to work with. The pane displays the views, columns, forms, fields, and Domino server agents in the database, and provides two quick ways of putting Domino tags into your page. You can:

- ▶ Right-click an item and choose Add to Web Page from the Context menu
- ▶ Drag and drop an item directly into the page

Either of these operations will paste a tag representation of the item into the editor at the current cursor position. Of course, if you prefer, you can add the tags manually in the editor.

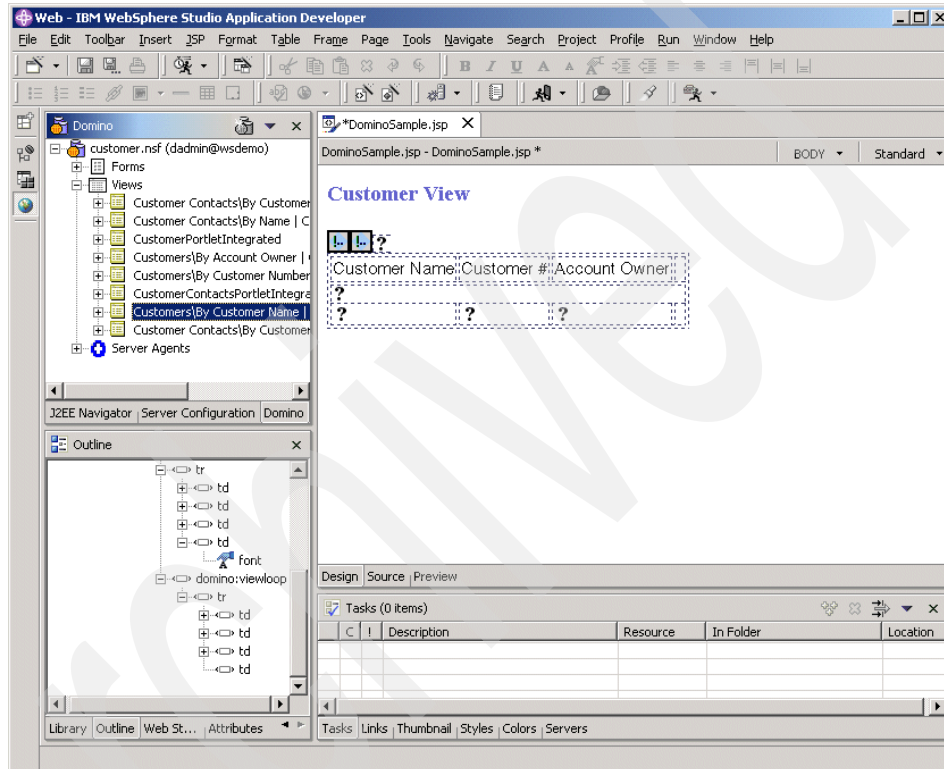


Figure 4-8 Lotus Domino Toolkit for WebSphere Studio 1.0

How it fits in the Lotus technology strategy

To understand the Lotus technology strategy, and how it applies to Domino applications and their development, envision a multi-lane superhighway with a Lotus Domino lane, a WebSphere lane, and a next generation lane. The Lotus Domino lane continues as far as the eye can see and runs parallel to the WebSphere lane, which also continues as far as the eye can see. The next generation lane merges with the other two lanes at a point in the future.

In the Lotus Domino lane, the Domino collaborative application development and deployment environment enables you to develop applications quickly and to take them off-line, bringing people, processes, and data together to facilitate both

productivity in e-business and quick decision-making. Lotus will continue the current Domino application development model and data store (NSF) and in the future, will enhance it to meet customer and developer requirements. As is the IBM tradition, Lotus Notes and Domino customers will benefit from comprehensive support for the foreseeable future. Additionally, Lotus Domino will increase its support of the Java 2 Platform, Enterprise Edition (J2EE) and infrastructure standards, such as Java Server Page (JSP) tags, Java APIs, LDAP, and RDB integration to assist developers interested in working in both the Domino and WebSphere lanes of the superhighway.

In the WebSphere lane, the J2EE specification is leveraged as the application development platform. J2EE provides a specific architecture for building, deploying, and managing applications in multiple tiers, often broken into presentation, logic, and data. This architecture is designed to provide scalability, flexibility, and manageability. While J2EE is a rich application development platform, it has very few features to support collaboration, so it benefits from having Lotus Domino to provide rich collaborative capabilities. Applications designed to use the Lotus Domino and WebSphere lanes blend powerful collaborative features with significant transactional scalability to deliver end-to-end e-business solutions.

4.4 Integration techniques

In this section we explain how to implement the four portlets that comprise the Sales Workplace described in 2.3, “Case study: A simple sales tracking application” on page 47.

We show you how we constructed four portlets from scratch using the Portal Toolkit wizards. These portlets will include JSP pages where we will place the Domino custom JSP tags that allow us to access the Lotus Domino applications. Finally, when all four portlets are working on their own, we show you how we added some extended functionality, first people awareness to be able to collaborate with the people involved on the documents, and next to integrate our portlet at the portal level through Click to Action.

The structure of the portlet page is shown in Figure 4-9.

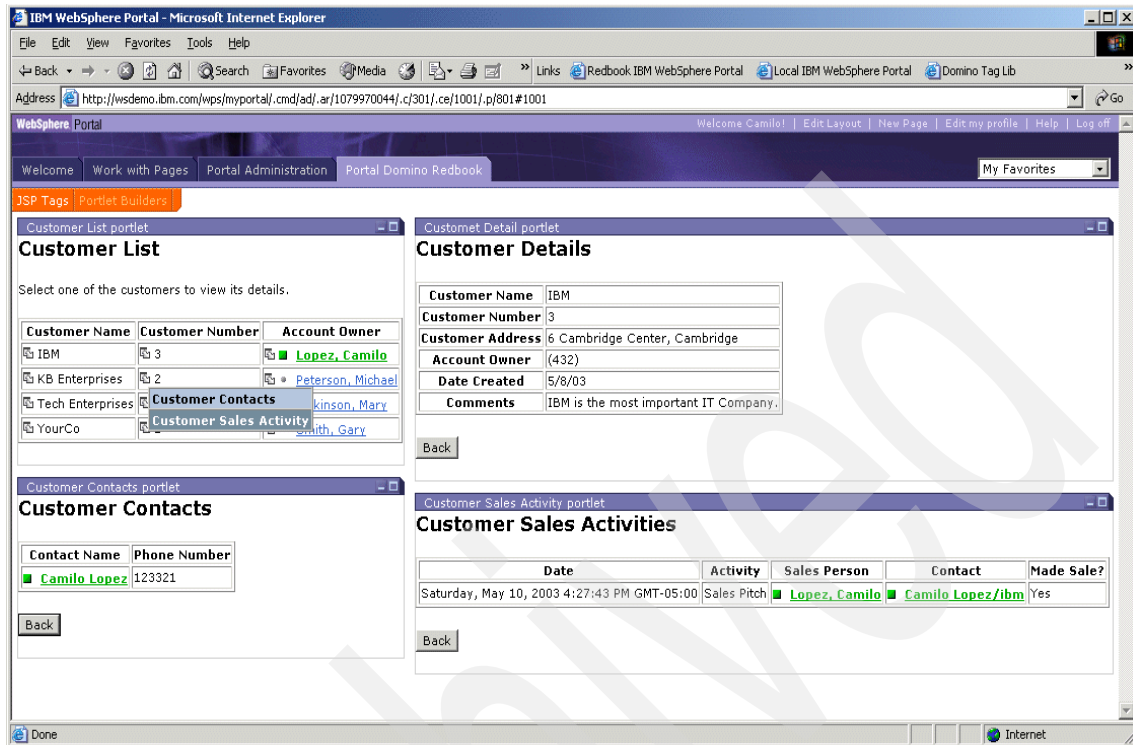


Figure 4-9 Domino custom JSP tags portlets

Now lets take a closer look at what each of the portlets will do.

► Customer List:

The purpose of this portlet is to display the contents of the view Customer By Name from the customer.nsf Domino database. Later on, we will enable this portlet to handle people awareness and click to action functionality. When we finish the enablement of our portlet it will look like the one displayed in Figure 4-10.



Figure 4-10 Customer List portlet

► Customer Detail

The Customer Detail portlet, when first invoked, will display a form with a drop-down list containing the customers from the database. After a customer is selected, the portlet will display the details about the customer. When we finish the enablement of our portlet it will look like the one displayed in the Figure 4-11.

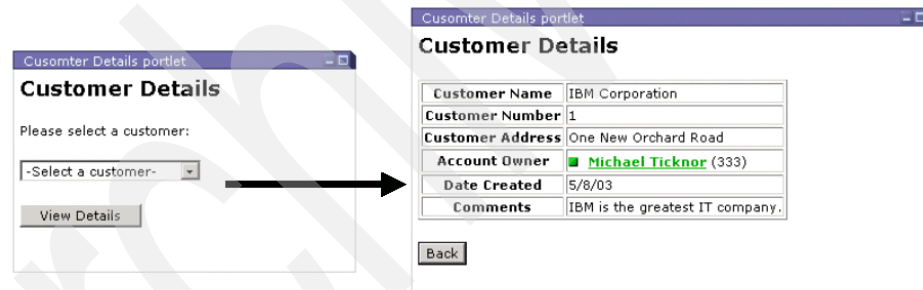


Figure 4-11 Customer Detail portlet

► Customer Contacts

The Customer Contacts portlet behaves similarly to the Customer Detail portlet. When initially viewed it will display a list of customers, and after a selection is made the portlet will display all the contacts for the specified customer. When we finish the enablement of our portlet it will look like the one displayed in Figure 4-12.

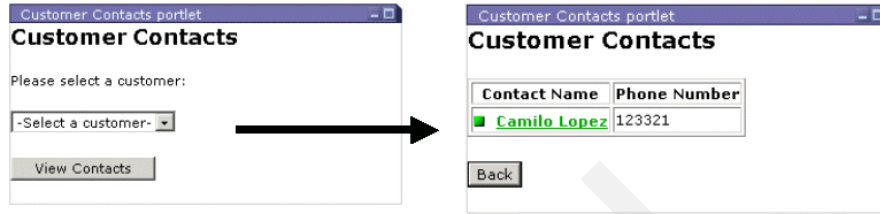


Figure 4-12 Customer Contacts portlet

► Customer Sales Activities

The Customer Sales Activities portlet initially displays a list of customers, and after a customer is selected, a list of past activities will be displayed. This list of activities is a view that is filtered by customer. When we finish the enablement of our portlet it will look like the one displayed in Figure 4-13.

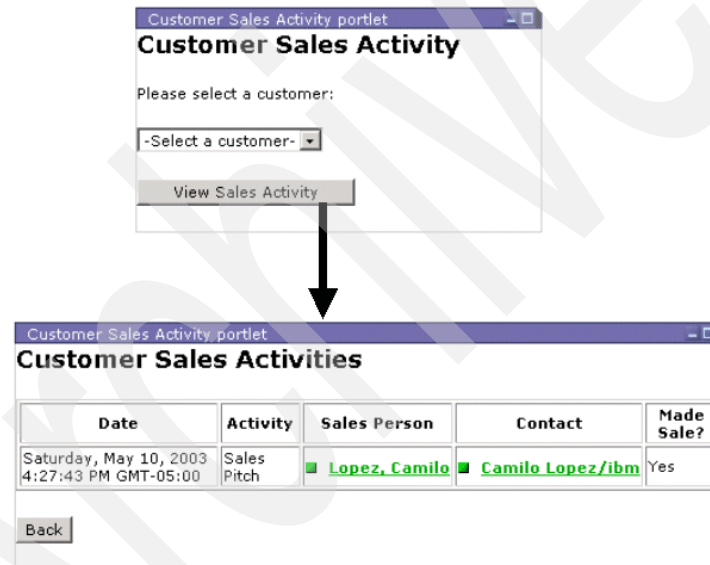


Figure 4-13 Customer Sales Activities portlet

4.5 Integration using Domino custom JSP Tag libraries

We used the Domino custom JSP Tag libraries technology to create the core portlets that will integrate Domino components. The details of the implementation are in this section.

4.5.1 Overview

This integration technique introduces the Portal toolkit's ability to produce portlet projects easily and shows how, in conjunction with the Lotus Domino toolkit for WebSphere Studio, it incorporate the Domino custom JSP tags into standard JSP pages to extract Domino data.

First we review some basic required concepts on the portlet class and the abilities of the portlet to do event-based integration. Further information on the portlet class, APIs, and related objects is discussed on the following chapter.

Domino custom JSP tag libraries

One of the JSP directives mentioned previously was the *taglib* directive. As we explained, the JSP tag library is a collection of custom JSP tags that encapsulate Java code through the simple use of tags. The library defines declarative, modular functionality that can be reused by any JSP page. The tags are defined in an XML format file known as the Tag Library Descriptor file or TLD. This descriptor tells the JSP parser-compiler what Java classes and methods should be interpreted. Grouping these tags in libraries gives the JSP developer a simple but powerful tool to incorporate Java code in the application without getting into the details of Java. These JSP tag libraries can be used extensively on your JSP pages to include Lotus Domino elements into your J2EE/Portal applications.

This option brings to the developer who is not a Java expert, or who is not knowledgeable on handling Lotus Domino Java back-end objects, the ability to enable a Web application to incorporate complex Lotus Domino interactions by simply adding a custom Domino JSP tag to the JSP. And since the WebSphere Portal framework is based on a J2EE environment that allows the inclusion of JSP files, the developer can expose Lotus Domino applications to the portal infrastructure with the use of custom Domino JSP tags.

Note: As explained in the following section, there are some performance considerations to take into account when using this option.

There are two Domino JSP tag libraries. Both comply with the standard specifications of JSP 1.1 and Java Servlet 2.2 developed by Sun Microsystems, which are supported by the WebSphere Portal infrastructure. The Lotus Domino tag libraries are:

- | | |
|--------------------|---|
| domtags.tld | Collaboration tags for accessing standard, back-end objects in the Domino data repository |
| domutil.tld | Utility tags for performing tasks that are common to all J2EE Web containers |

Types of Domino JSP tags

The Domino JSP tags can be grouped into four types:

- ▶ Data access tags
- ▶ Data input tags
- ▶ Process control tags
- ▶ Utility tags

In this section we describe some useful Domino JSP tags. They are the most pertinent for developing portlets that extract information from Lotus Domino applications, but they are just a subset of all the tags available. Refer to the Lotus Domino Designer on-line help for comprehensive information about all the available tags.

Data access tags

The data access tags allow the user to gain access to several of the most important objects in the Lotus Domino object hierarchy. Table 4-3 gives a brief overview of the tags that were useful for our portalizing project.

Table 4-3 *Data access tags*

Tag name	Description
session	Defines the environment that the core collaboration tags run in. Use the session tag when multiple core tags are included on the same JavaServer page. This initializes and tears down the Domino session one time only, providing improved performance. This tag is not required if there is only one core tag on the page; the user and password attributes of the core tag implement the session for you. However, if you are using a CORBA session, this tag must be used, since the host attribute cannot be specified on the top-level tags. If one or more of the core tags reside in the same database, you can further improve the efficiency of the page by using the db tag instead of this tag to wrap them.

Tag name	Description
database	<p>Provides a database context for all enclosed tags. If you are including several core tags that are running off the same database in a page, you can wrap them in this tag to increase scalability. This tag is not required if there is only one core tag on the page; the dbserver and dbname attributes of that core tag implement the db tag for you. Do not use this tag to wrap two core tags that reside in different databases; use the session tag instead.</p> <p>To loop over all the documents in a database, use the docloop tag nested inside the db tag.</p> <p>If you do not supply values for the user and password attributes, the database identifies the user as an Anonymous user. The database's ACL must have an "Anonymous" entry with at least Reader level privileges to the database for the tag to access the database successfully.</p>
document	<p>Enables an author to create or edit a document. The body of this tag is always evaluated. You can use the item, setitem, and formula tags within the body of this tag. To display the document, use the item and formula tags within the body. To edit the document, use the setitem tag within the body. If you set the value of an item using the setitem tag within the body of this tag, you must save your changes also within the body of this tag. You can save your changes by either:</p> <ul style="list-style-type: none"> - Specifying an ID attribute, and calling <id>.save() in a scriptlet - Using the savenow tag
view	<p>Displays the summary information of a subset of documents in a database. You can use the key and ftsearch attributes to further qualify the subset.</p>
ftsearch	<p>Performs a full text search of a database. The result is a list of documents that fit the search criteria. Use the docloop or page tags to sequence over the resulting documents and the item or formula tags to display the summary data.</p>
item	<p>In the context of a form, document, or docloop tag, displays the value of an item in the current document. You can set the tag's readonly format using the format attribute. The body of the item tag is not evaluated. To provide update access to an item, use the input tag in the context of a form tag.</p>
viewitem	<p>Displays the value of an item in the current row of a view. The body of this tag is not evaluated.</p>
unid	<p>Displays the unique ID of the current document. The body of this tag is not evaluated.</p>

Tag name	Description
viewloop	Iterates over the results of a view, evaluating the body once per ViewEntry object in the result. If no results are returned, displays nothing. To create custom text to display when no objects are returned, use the novalues tag. You can access the items of an object using the viewitem tag.
docloop	Iterates through the items returned by a view, ftsearch, db, responses, or page tag as documents. The body of this command is output once per iteration, which means once per document in the result set. When a result contains no objects, the body of this tag is not evaluated. Use the novalues tag to create custom text to display when a result contains no objects. Use the item tag to access the items in a document. Note: If you use this tag within the context of the view tag, there will be a performance hit. The viewloop tag is a good alternative to use for views; it evaluates the body once per entry in the view, instead of loading a document for each entry in the view.

Data input tags

Data input tags allow the input of information from a JSP file to the Lotus Domino application. There are constraints on the use of these tags because of restrictions of the WebSphere Portal on the management of URLs and JavaScript inside a portlet, so they are not covered in detail in this chapter. Let's take a look at why data input tags won't instantly work with our portlet infrastructure like the other tags do.

When you create a JSP and insert the Data Input tags, for example <Domino:form>, <Domino:input> and <Domino:savedoc> tags, you will see a lot of JavaScript code generated on the HTML page that will basically generate action URLs that process the input form. Example 4-6 shows some JavaScript code generated by the Data Input tags.

Example 4-6 JavaScript code generated by Data Input tags

```
function selfNavigate() {
    var actionURL = location.pathname;
    var argName;
    var first = true;
    for (argName in DominoArgs) {
        var argValue = DominoArgs[argName];
        if (argValue != null) {
            if (first) {
                actionURL += '?';
                first = false;
            } else {
                actionURL += '&';
            }
        }
    }
}
```

```

        if (typeof argValue == 'object') {
            for (index = 0; index < argValue.length; index++) {
                if( index > 0)
                    actionURL += '&';
                actionURL += argName + '=' + argValue[index];
            }
        } else {
            actionURL += argName + '=' + argValue;
        }
    }
}
if (DominoForm != null) {
    DominoForm.action = actionURL;
    DominoForm.performSubmit();
} else {
    location.replace(actionURL);
}
}

```

This code appends to the portal URL the command and arguments. That is not the usual way information will be passed on the portal. Also, if there are two portlets accessing these data input tags, then you would have conflicts. Be careful when adding data input tags since they can be tricky on a portlet infrastructure. We did not use any of these tags in our examples.

Process control tags

These tags allow the JSP to query the state or properties of the Domino application, and based on the result, modify the presentation of our portlet.

Table 4-4 Process control tags

Tag name	Description
ifcategoryentry	Conditional tag. In the context of a viewloop tag, if the current ViewEntry object is a category, meaning it displays in the category column of a hierarchical or categorized view, the body of this tag is displayed. This tag can be used to edit the indentation or display properties of category entries to distinguish them from the other entry types, represented by the ifdocumententry, iftotalentry, and ifconflicentry tags.
ifdocumententry	Conditional tag. In the context of a viewloop tag, if the current ViewEntry object represents a document in a hierarchical or categorized view, the body of this tag displays.

Tag name	Description
ifdocauthor	<p>Conditional tag. Restricts generation of the JSP page to only those users who have author access to the current document.</p> <p>Users have author access to a document if they have:</p> <ul style="list-style-type: none"> - Editor, designer, or manager access to a database - Author access to the database and their name is contained in an authors field on the form (or authors item in the document), if the form or document has one. <p>Note: If the user attempting to access the JSP page does not have at least reader access to the database (if they have only depositor access, for instance), an exception is thrown. To avoid this, if a user might have depositor level access, wrap the document or form tag with an ifreader tag.</p>
nodocument	<p>Displays an alternate message if there is no document associated with the unid attribute being passed to the form tag. If the requested document does not exist, displays the body of this tag. To localize the text that displays, use the msg tag in the tag body.</p>
runagent	<p>Runs a specified back-end agent on the server. You can specify agents that run on the server, such as agents that send mail or create a folder. You cannot specify an agent that displays information in the browser to the user. For example, you cannot run an agent that contains a LotusScript print statement using this tag. If you want to capture data using this tag to run an agent, you must write the agent's results to a document and access the document to recover the data.</p> <p>To use this tag to run an agent, set up the basics tab of the Agent Properties box for the agent as follows:</p> <ol style="list-style-type: none"> 1. In the Trigger section, select the On event radio button. 2. Choose Action menu selection in the drop-down box. 3. In the Target drop-down box, choose None.
ifdbrole	<p>Conditional tag. In the context of a db tag, restricts generation of the JSP page to only those users included in a custom ACL role. To allow for programmatic determination of access control, the standard access level names (author, editor, and so forth) can be used as role names.</p>

Utility tags

The utility tags allow you to control the flow of the presentation logic on the portlet based on Domino conditions and expressions.

Table 4-5 Utility tags

Tag name	Description
if	<p>The body of this tag conditionally executes depending on a specified condition. One condition can be specified per tag and can be passed in by including a condition tag in the body of the if tag or as one of the if tag attributes.</p> <p>Note: Only one condition can be specified; if more than one is specified, an exception is thrown.</p> <p>Note: If you do include a condition tag in the body of this tag, the body of this tag <i>must</i> always be evaluated. If the condition fails, the evaluated body is discarded. You should not include code that has side-effects in an if tag that is being used in conjunction with a condition tag.</p>
else	<p>Used after an if or elseif tag; this tag executes only if none of the preceding if or elseif tags were executed.</p>
elseif	<p>Conditionally executes its body if the corresponding if tag did not execute and the condition for the elseif tag is true.</p> <p>Note: If you include a condition tag in the body of this tag, the body of this tag <i>must</i> always be evaluated. If the condition fails, the evaluated body is discarded. You should not include code with side-effects in this tag if it is being used in conjunction with a condition tag.</p>
condition	<p>Specifies the condition of an if or elseif tag.</p> <p>The body of the tag must be a string that represents a Boolean value, for example: true, false, 0, 1, yes, or no. This tag has no attributes.</p> <p>Note: If this tag is used in the context of the if tag, the body of the if tag <i>must</i> always be evaluated. If the condition fails, the evaluated body is discarded. You should not include code with side-effects in an if or elseif tag being used in conjunction with a condition tag.</p>
switch	<p>Enables the JSP author to incorporate control flow. Each case or default tag contained in the body of this tag is processed in turn and the first match that is found is evaluated.</p> <p>You cannot provide more than one value for this tag. If you supply a value attribute, do not include a switchvalue tag in the tag body.</p>
case	<p>Provides processing instructions to a switch tag when a match is made. If the case tag's value attribute matches the switch tag's value attribute, the body of the case tag is evaluated.</p>

Tag name	Description
format	Formats a value. You can pass in a value to format or format the tag body. You can also provide an ID name to save the formatted body or value as a variable.
browsertag	Identifies the capabilities associated with the current (or specified) browser. Either returns the value of the capability or conditionally evaluates the body of the tag based on whether or not the browser supports the passed capability. For capability variables consult the Lotus Domino 6 Designer documentation.

Custom Domino JSP tag issues

There are a number of issues related to the incorporation of the JSP tag libraries from Lotus Domino into WebSphere Portal and WebSphere Application Server.

- ▶ In the WebSphere Application Server, including a form into a page using a `<jsp:include>` tag renders the `validatehref` attribute on the form useless. This attribute can be defined by the following tags:
 - form
 - deletedoc
 - docnavimg
 - savedoc
 - saveclosedoc

4.5.2 Considerations

There are a few issues to consider when using Lotus Domino JSP tag libraries to expose Lotus Domino applications to the WebSphere Portal. They are:

- ▶ Custom Domino JSP tags aren't supported on WebSphere Portal, so you should perform extensive testing on the functionality required.
- ▶ On each request that the JSP executes with embedded custom Domino JSP tag libraries to a remote server, a different session connection through DIIOP with the Lotus Domino server is created, which can impose a significant overhead. This is especially important when the application will be accessed by a large number of users.

The experience we had in the lab is that it takes an average of one second to display a portlet with Domino JSP tags. Considering a Portal page with four embedded portlets, it would take approximately four seconds to render, and that is for only one user.

In the next chapter we review a sample Java implementation for handling Domino sessions to make our Lotus Domino applications perform well under heavy load, using a session management pool. We show that using the

session pool will reduce, by more than half, the overhead imposed by the current implementation of the custom Domino JSP tags. In the next chapter we also discuss the DIOP task in the Lotus Domino server.

- ▶ The exposure of rich text fields on our portlets isn't straightforward. To properly address this issue we have to consider two possible scenarios:
 - If the rich text was filled from the Lotus Notes client, then we must create a reverse proxy to receive the HTML produced by the Lotus Domino server and then include it on the portal, taking care to make any link portal friendly.

We discuss this scenario further in the next chapter, where we review some methods of handling rich text fields.
 - If the rich text field will be used only from the portal, we recommend storing the information in a neutral XML format so it can be encoded and decoded with the use of XSLT stylesheets.
- ▶ Some data input custom Domino JSP tags aren't portal friendly since they insert JavaScript code and URL links or actions that are absolute and not relative to the WebSphere Portal URI structure.
- ▶ We have views with just a few documents in our examples; in real implementations we need pagination on our views. Pagination made with custom Domino JSP tags also carries URL links that are not portal friendly, so we have to implement our own Java pagination components. In the next chapter we explore a JSP-Java Hybrid alternative.

4.5.3 Implementation example

We start the implementation of this technique by creating a portlet project and creating the four basic portlets that expose the Lotus Domino application using custom Domino JSP tag libraries.

Building the portlet project with WebSphere Studio

First, we build a portlet application structure fit to handle the portlets that will communicate to our Lotus Domino application. To do this we use several of the tools introduced in previous sections. We performed the following steps to build the portlet project:

1. Open WebSphere Studio Application Developer.
2. Switch to the Portlet perspective
3. Select File → New → Portlet Application Project. A Create a Portlet Project dialog is displayed, as shown in Figure 4-14.

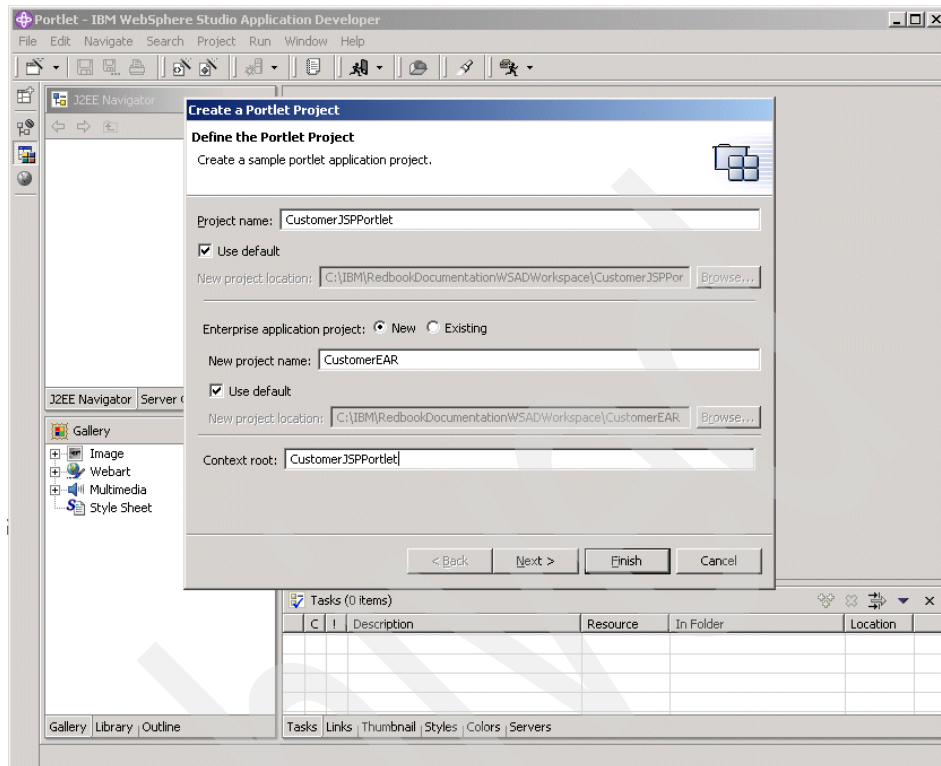


Figure 4-14 Creating the Portal project

4. Name the project CustomerJSPPortlet.
5. Click Next and select the Basic portlet. The Basic portlet will build all the structure needed to hold our portlet.
6. Click Next and create the portlet application with parameters like those shown in Figure 4-15. Note that the wizard creates an initial portlet. This portlet will be the Customer List portlet.

Create a Portlet Project

Basic Portlet Parameters
Enter the properties of the basic portlet.

Portlet application name: Customer Information application

Portlet name: Customer List portlet

Concrete portlet application name: Customer Information application

Concrete portlet name: Customer List portlet

Default locale: en English

Concrete portlet title: Customer List portlet

Portlet class name: CustomerList

Markups: html chtml wml VoiceXML

< Back Next > Finish Cancel

Figure 4-15 Parameters for creating the portlet project

7. Click Finish.

Now let's inspect what the creation wizard built.

The newly created portlet project appears in the J2EE Navigator view in the left. Within the project are two important folders: *Web Content* and *Java Source*.

In the *Web Content* folder there is a JSP directory, and inside it there is a JSP created for every state the portlet can acquire (view.jsp, help.jsp, configure.jsp, and edit.jsp). Since our portlet will only be accessed in View mode, we remove all but the view.jsp file, including the HTML folder and its contents that holds the same JSPs when an HTML consumer client consults the portlet.

Now we take a look at the *Java Source* folder. It contains a default generated portlet package. This package contains two Java files: a CustomerList.java file, which is our portlet; and a CustomerListBean.java file, which is a helper Java Bean. Since our objective is interacting with Domino directly from our JSP pages, we remove the CustomerListBean.java file to clean up the package and make it easier to understand.

Several errors are displayed on the tasks view after the removal of the helper bean. To correct them we open the CustomerList.java file which contains our Portlet code and perform the following tasks:

- ▶ Inside the portlet class are methods referring to each of the portlet states, like doView(), doHelp, doConfigure(), and doEdit() methods. Since we are working only with the view state, we remove the doHelp(), doConfigure, and doEdit() methods from the class.
- ▶ Inside the doView() method there is a reference to the helper JavaBean class we deleted. We erase the code that references the helper Bean, and on the remaining line, modify it to point to the directory structure that we will create next. It should look like Example 4-7 after the modifications.

Example 4-7 CustomerList portlet redirection

```
getPortletConfig().getContext().  
include("/jsp/CustomList/View."+getJspExtension(request), request, response);
```

- ▶ Save the CustomerList.class portlet file.

The file should look like the one illustrated in Figure 4-16.

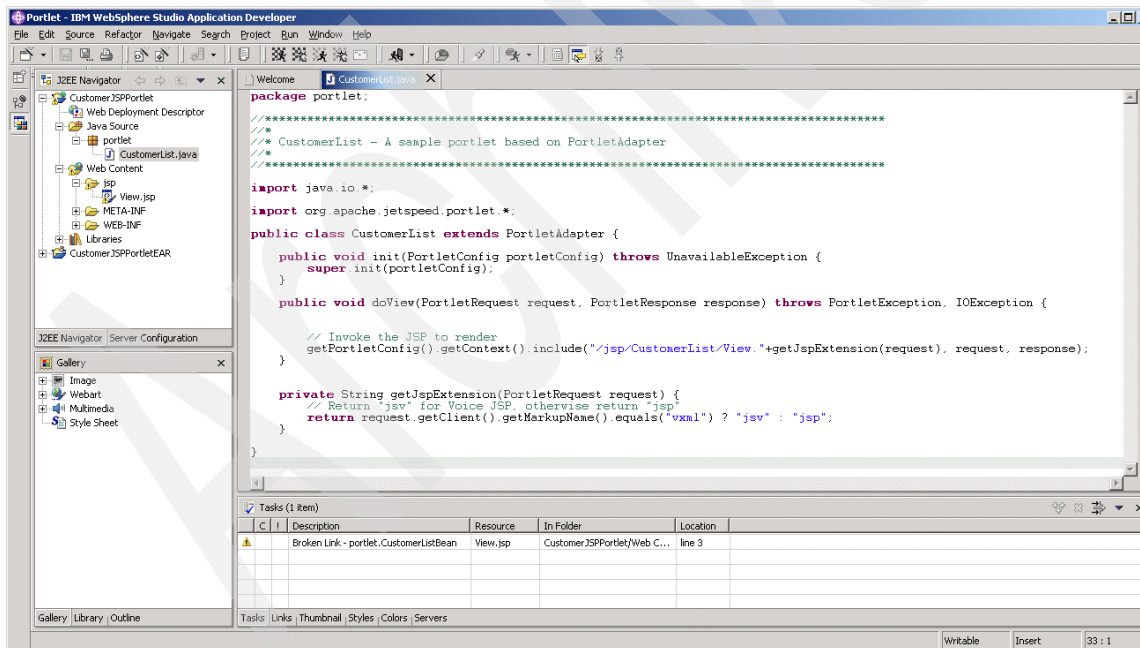


Figure 4-16 Editing the CustomerList.class portlet

- ▶ Now let's go back to the JSP folder inside the Web Content folder. There is a warning sign beside the View.jsp file, so we open the file and do the following:
 - Go to the Source tab.
 - Just leave the `<%@page contentType="text/html" %>` line.
 - Since we are going to have several portlets in our Web application, we create under the JSP directory the structure to hold our 4 portlets. This is done by adding the `/jsp/CustomDetails`, `/jsp/CustomDetails`, `/jsp/CustomContacts` and `/jsp/CustomSalesActivity` folders, like the ones displayed in the Figure 4-17 on page 172. Then we move the `/jsp/View.jsp` file to the `CustomerList` folder.

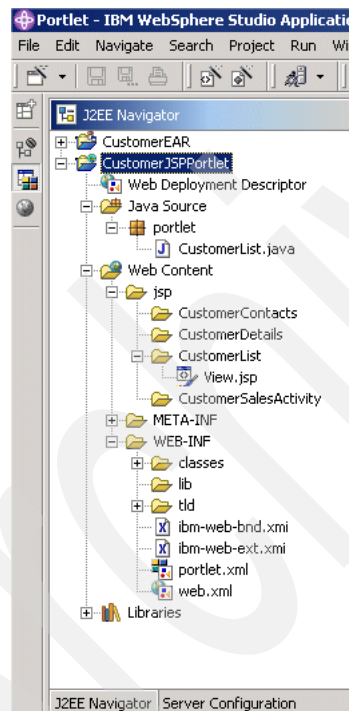


Figure 4-17 JSP directory structure

This completes the creation of the skeleton for our portlets. Next, we are going to add the Domino capabilities that are included in the Lotus Domino Toolkit for WebSphere Studio.

Adding Lotus Domino JSP tags to our portlets

To enable our project to include the JSP tag libraries, we performed the following steps:

1. Right-click the CustomerJSPPortlet project and select Properties.
2. In the Properties window, select the Web option; in the Available Web Project features field, select Include Domino Custom Tags Library, as shown in Figure 4-18.

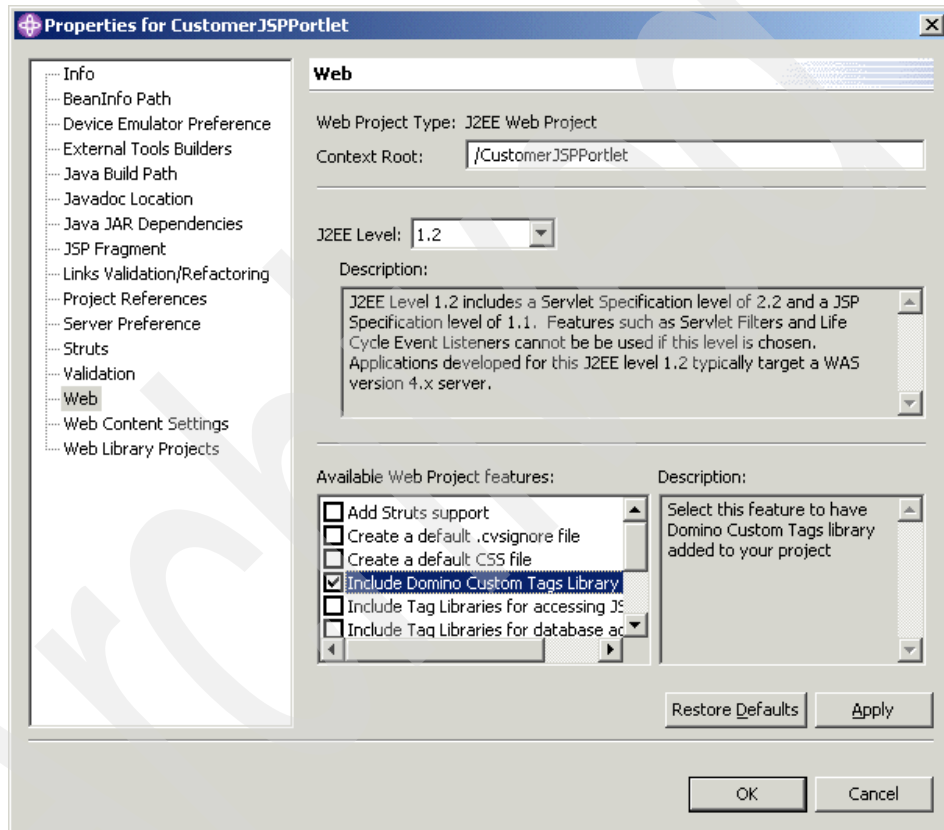


Figure 4-18 Including Domino custom JSP tags on the Web project

3. Click the Apply button and then OK.

New files were added to our Web project: two Domino custom tag libraries on the WEB-INF directory, and three JAR files on the WEB-INF/lib folder. This result is shown in Figure 4-19.

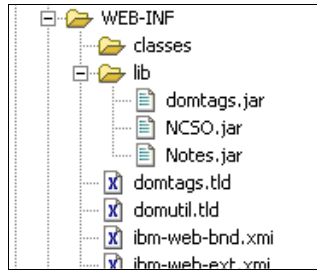


Figure 4-19 Inserting Domino custom tag libraries

4. Switch to the Web perspective.
5. Select the Domino view. If you don't see the Domino view, you don't have the Lotus Domino Toolkit for WebSphere Studio installed.
6. Right-click the white canvas and select New Database connection.

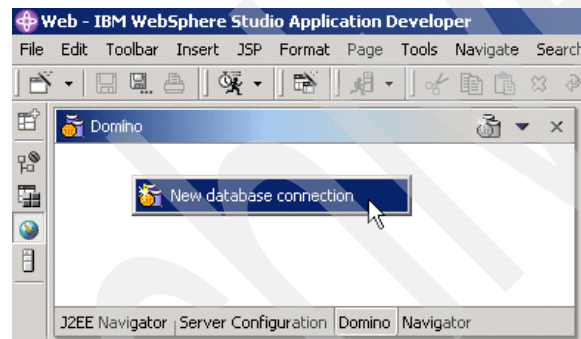


Figure 4-20 Connecting to the Domino database

7. When prompted for connectivity information, fill in the appropriate values for the Domino server and point to the customer.nsf database you are using in your project. Click Finish. You will be prompted for a password. Enter it and click OK.

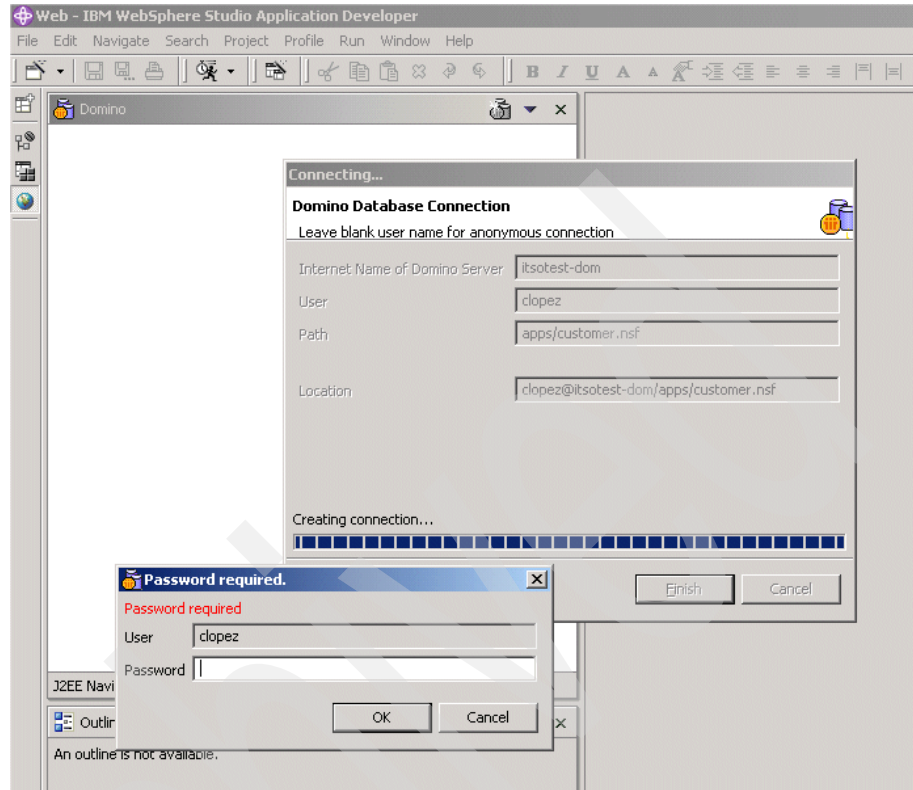


Figure 4-21 Connecting to Domino databases through Domino toolkit

When the connection is complete, a list of all the forms and views that are on the database is displayed, as illustrated in Figure 4-22 on page 176. On this initial portlet you will be including the Customers/By Customer Name view on your JSP.

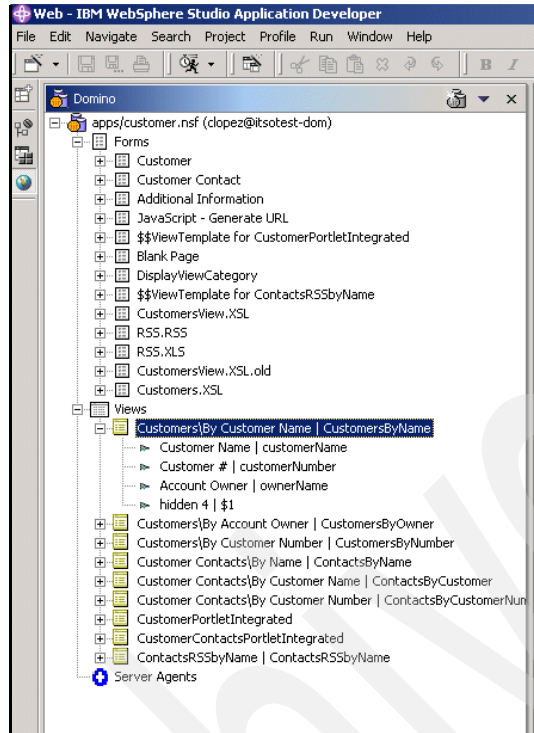


Figure 4-22 Reviewing the Customer database

8. Change from the Domino view to the J2EE Navigator view and open the View.jsp file in design mode.
9. Go back to the Domino view and drag the Customers/By Customer Name view to the View.jsp.

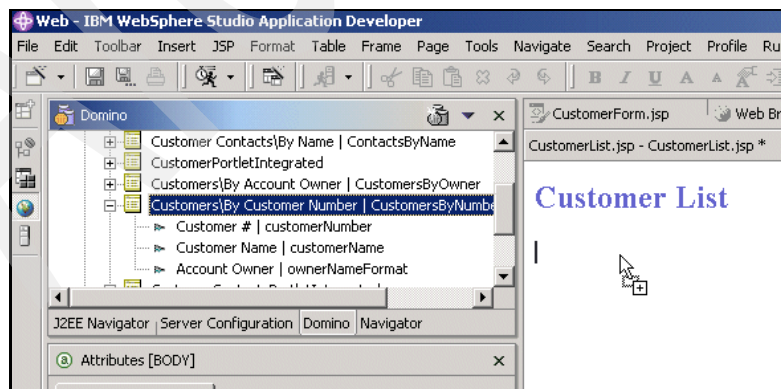


Figure 4-23 Adding the Customers/Customer By Name view to View.jsp

10. Add the simple label “Customer List” with Heading 3 format.
11. Delete the hidden (fourth) column since this is a control column.
12. On the JSP file select the top row of the table to be a header row. To do so, right click the selected row and select Attributes, and in the Attributes view change from Data to Head like shown in Figure 4-24 on page 177.

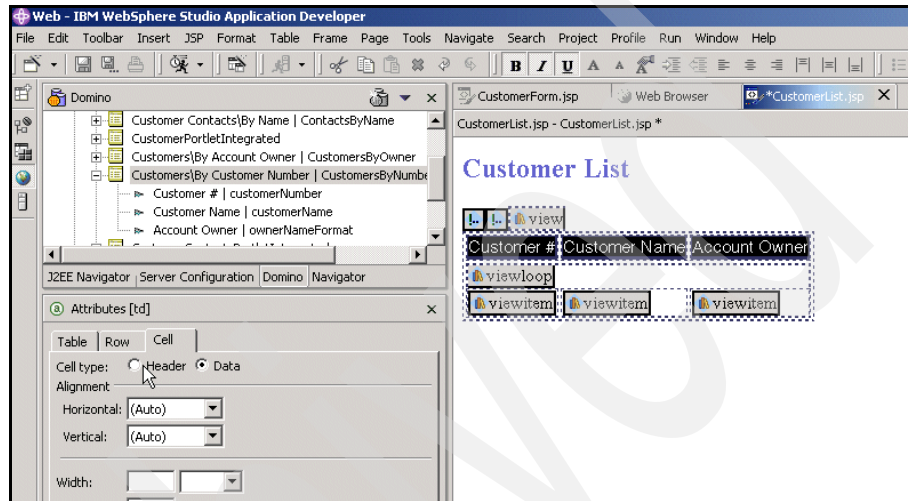


Figure 4-24 Changing the table to include headers

13. Your new JSP file should look similar to Figure 4-25.

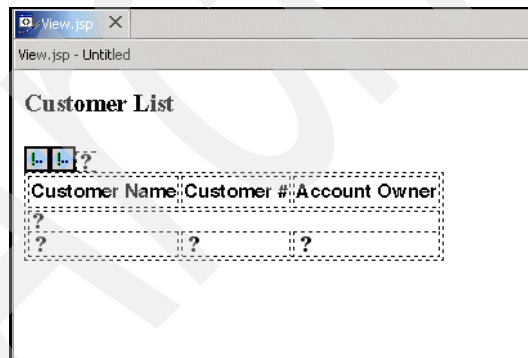


Figure 4-25 Initial portlet JSP

14. Two errors are displayed on the tasks view. To resolve them, take a look at the source code for the JSP.

Change to the source tab and you will notice that there are no JSP Tag library definitions on the JSP, so add them as well as the WPS library.

15. Include at the beginning of the JSP the following lines:

```
<%@ taglib uri="/WEB-INF/domutil.tld" prefix="util" %>
<%@ taglib uri="/WEB-INF/domtags.tld" prefix="Domino" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>
```

These includes insert the custom Domino libraries so that they can be referenced inside your code. The last line initializes the portlet tag library; there isn't any need to initialize the Domino libraries.

16. Also notice that in the source code there is a predefined `<Domino:view>` tag that will connect to the Domino server, establish a session, and extract the information required. Modify the user attribute in the tag to `*webuser`, so that rather than being hard coded, it is extracted from the single sign-on information that Domino and WebSphere Portal have. Delete the password attribute. Also, change the viewname to its alias, since spaces in the view name sometimes can have problems. It is also a best practice to use an alias instead of view name, as view names can change. The tag should look something like:

```
<Domino:view viewname="CustomersByName" dbserver="CN=itsotest-dom/0=itsoportal" dbname="apps/customer.nsf" user="*webuser" host="itsotest-dom">
```

17. Save the `/jsp/CustomerList/View.jsp` file.

We have now a working portlet that displays basic information extracted from our Lotus Domino database.

Tip: The `<Domino:view>` tag currently has a `*webuser` as the username, and no password specified. This option lets the portlet establish the connection to the Lotus Domino database with the same identity as the one used to log into the portal. Also notice that when the `*webuser` is used, no password is specified.

As a prerequisite to use the same identity, single sign-on (SSO) must be enabled.

Deploying our initial application

Use the following steps to deploy the application.

1. At the WebSphere Studio Application Developer tool, change to the J2EE Navigator view.

2. Right-click the CustomerJSPPortlet project and select Export. A wizard comes up. Select the WAR file export and basically export the project to your hard disk.
3. Install the portlet into WebSphere Portal
 - a. Log into the WebSphere Portal with an administrative user, and select the Portal Administration place.
 - b. It should open right away to the Install Portlets window. Select the newly created .WAR file.

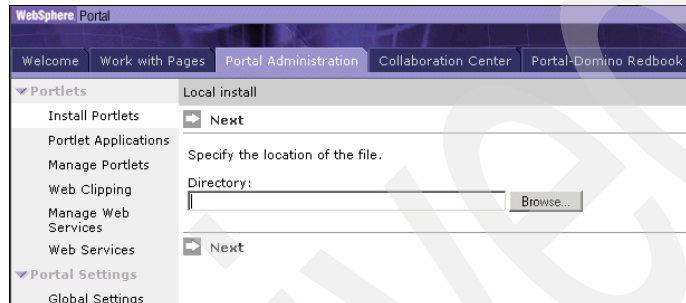


Figure 4-26 Installing the Customer List portlet

- c. Click Next. Once it recognizes your portlet, click Install. When the installation is finished, a “Portlets successfully installed” message is displayed.

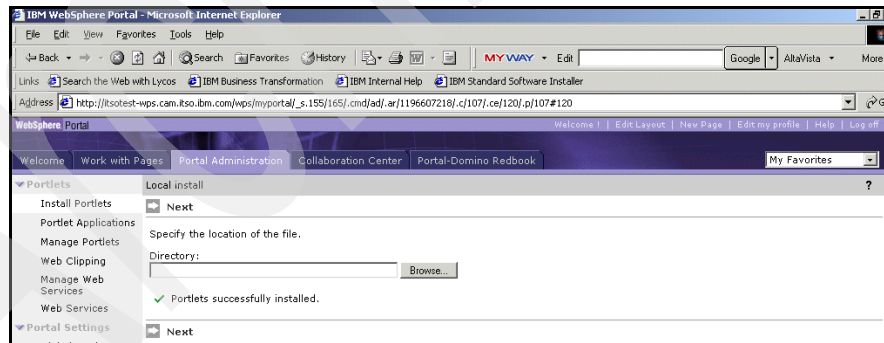


Figure 4-27 Successful installation of the portlet

4. Grant access to the portlet.

Now that the portlet is installed to the portal, grant access to the users. Go to the Security/Access Control List in the Portal Administration place. Select the required users or groups and grant them at least View access. This is illustrated in Figure 4-28.

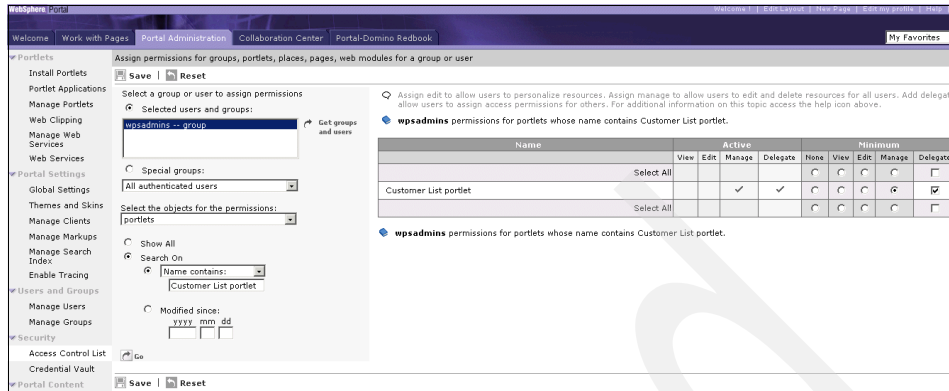


Figure 4-28 Granting access to the Customer List portlet

5. Add the portlet to the page.
 - a. Go to the Work with Pages place and select Edit Page Composition. Select the place and page where you will install the portlets.
 - b. Modify the number of columns to two, and in the first column click the Add Content button. This directs you to a portlet search page.
 - c. Search by Customer List and your portlet should come up. Select it and click OK.
 - d. Finally, click Done. Figure 4-29 on page 180 shows the result of adding the portlet into a page.

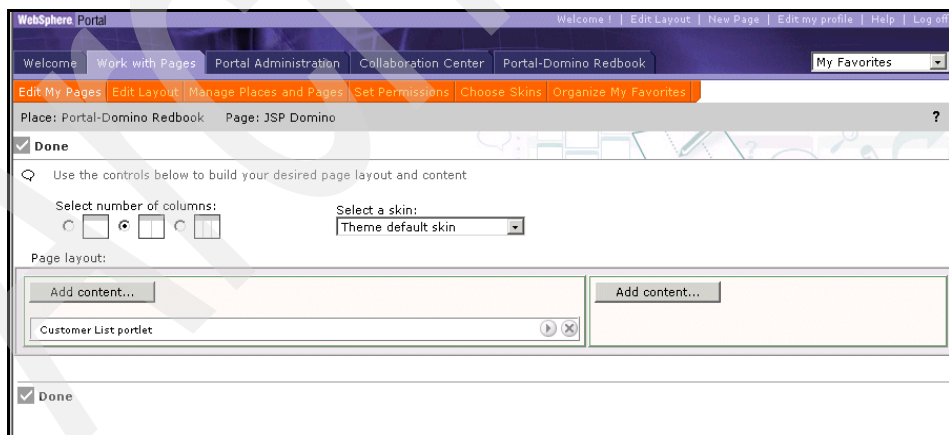
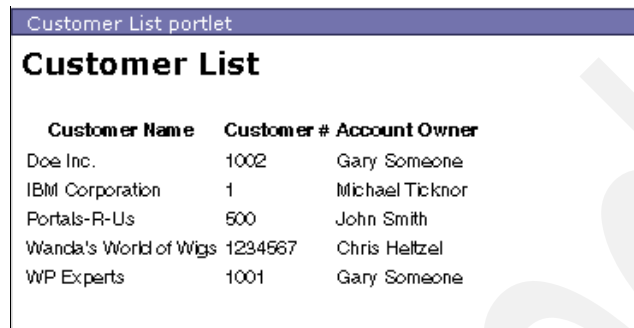


Figure 4-29 Adding the Customer List portlet to the page

6. Preview the portlet.

Log into the portal with a user name that has access to the Lotus Domino database and select the page where you added the portlet. You should be able to view the portlet running.



Customer Name	Customer #	Account Owner
Doe Inc.	1002	Gary Someone
IBM Corporation	1	Michael Ticknor
Portals-R-U's	500	John Smith
Wanda's World of Wigs	1234567	Chris Heltzel
WP Experts	1001	Gary Someone

Figure 4-30 Initial portlet with Domino functionality accessed through JSP tags

Figure 4-30 shows an example of a basic view being displayed in a portlet.

The Customer Details portlet

We now describe how to construct the Customer Details portlet. This portlet initially displays a JSP with a drop-down list of the customers available. After a customer is selected, it reloads and displays the customer information details.

Customer detail portlet Java code

1. Open the Web Project from the Portlet perspective.
2. Copy the CustomerList.java file and paste it to the same package. When prompted to rename the file, rename it CustomerDetails.java.
3. Open the CustomerDetails.java file and modify its doView() method to target the CustomerDetails folder.
4. Replace the doView() method to include the code in Example 4-8.

Example 4-8 Customer Detail portlet doView() method

```
// We are checking if the uidDoc attribute is on the session, initially it will
// be null but once a customer is selected the uidDoc will be populated.
if(request.getPortletSession().getAttribute("uidDoc") != null){
    // Creating a URI for the details button when a customer is selected.
    PortletURI detailURI = response.createURI();
    // We are creating the details action
    DefaultPortletAction detailAction = new DefaultPortletAction("details");
    detailURI.addAction(detailAction);
    // We are storing the URI in the request objects
    request.setAttribute("details", detailURI.toString());
    // Now we are creating the return URI for the back button
```

```

PortletURI returnURI = response.createReturnURI();
// We are storing the return URI in the request as well
request.setAttribute("CD_back", returnURI.toString());
// We are redirecting to the CustomerDetails.jsp file
getPortletConfig().getContext().include(
    "/jsp/CustomerDetails/CustomerDetails."+getJspExtension(request),
    request, response);
} else {
    // If no customer is selected perform this code.
    PortletURI detailURI = response.createURI();
    DefaultPortletAction detailAction = new DefaultPortletAction("details");
    detailURI.addAction(detailAction);
    request.setAttribute("details", detailURI.toString());
    // We are redirecting to the View.jsp file
    getPortletConfig().getContext().include(
        "/jsp/CustomerDetails/View."+getJspExtension(request), request,
        response);
}

```

This code selects whether you want to display the initial selection JSP or the customer details JSP, based on the `uidDoc` attribute stored in the session. Notice that we need a new JSP called `CustomerDetail.jsp` to display the customer details; we will do this later. In the code, before including the target JSP, actions are created. These will respond to users' interactions through forms and buttons.

As we saw in the introduction to this integration technique, this portlet will start with a JSP that displays a drop-down list where the user selects the customer. When the user clicks the `View Details` button it will activate an action called *details*. If you inspect the code you will find that this action URI is going to be stored as an attribute on the request object.

The inclusion of both JSP files will help make our portlet work as a standalone portlet if needed. (In the next section we describe how to enable the click to action communication between portlets, but for now they will work independently.)

5. For the Customer Details portlet we introduce a new option that is in the portlet structure, the *actions*. The actions let the presenting JSP send information to the portlet class to perform the required business logic. In our portlet the `View.jsp` will send a `details` action to the CustomerDetails portlet and as a result the portlet will redirect the request to the `CustomerDetails.jsp` page, and pass to it the UI of the document. Figure 4-31 illustrates the action's interactions in the Customer Details portlet.

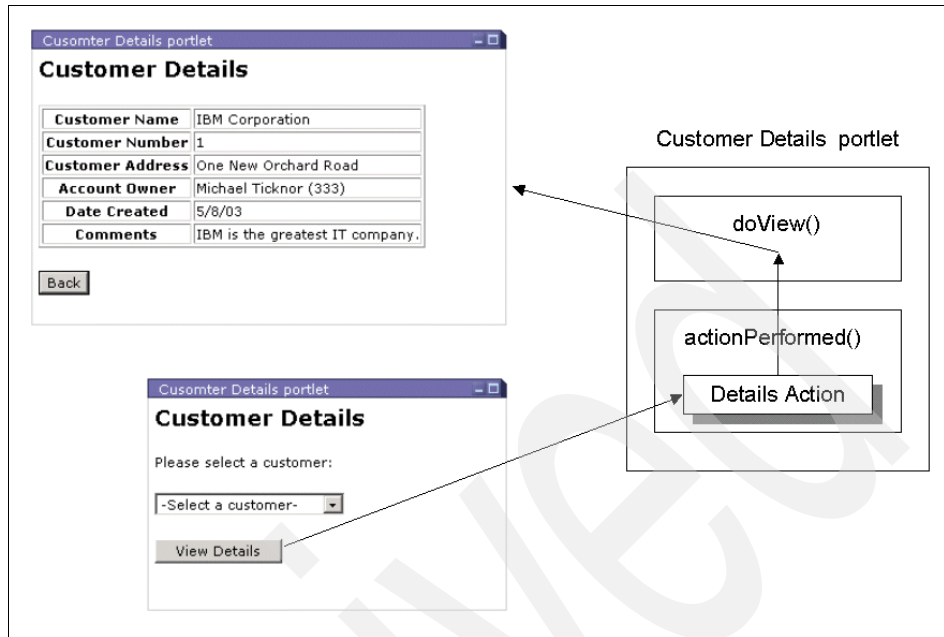


Figure 4-31 Action interaction in the Customer Details portlet

- Now we will implement action listening capabilities to our portlet. To do this we will first implement an interface called `ActionListener`. The declaration of our portlet will look something like the following:

```
public class CustomerDetails extends PortletAdapter implements ActionListener {
```

- Import into the `CustomerDetails.java` portlet the `org.apache.jetspeed.portlet.event.*`; package by adding the following line at the beginning of the class:

```
import org.apache.jetspeed.portlet.event.*;
```

- Now that our portlet will listen to requests made through our JSP pages, we have to implement a new method called `ActionPerformed`. Add the following method at the end of your portlet class:

Example 4-9 Action Performed method of the Customer Details

```
public void actionPerformed(ActionEvent event) throws PortletException{
    DefaultPortletAction action=(DefaultPortletAction)event.getAction();
    PortletRequest request=event.getRequest();
    if(action!=null){
        if(action.getName().equals("details")) {
            request.getPortletSession().setAttribute(
                "uidDoc",request.getParameter("uid"));
        }
    }
}
```

```
}  
}
```

This code basically captures the uid parameter from the request and stores it as an attribute called uidDoc in the session object.

Tip: Storing information on the Portlet session object will enable the user to switch between pages and places, and when the user comes back to this page the portlet will still be displaying the information previously selected.

If you store the information on the Portlet request object, the result of the portlet will only be available if the user changes pages and later comes back.

Now lets take care of the JSPs. First we copy and modify the existing View.jsp file, and after that we create the new CustomerDetail.jsp

View.jsp

1. Copy the View.jsp file from the /jsp/CustomerList folder to the /jsp/CustomerDetails folder.
2. Open the newly copied View.jsp file in Source mode.
3. Delete the code inside and *including* the <Domino:view> tags.
4. Modify the label from Customer List to Customer Details.
5. Insert the following code in the JSP file:

Example 4-10 Customer Detail portlet, View.jsp fragment initial contents

```
<!-- Text to indicate to the user that a customer selection is needed -->  
<P>Please select a customer:  
<!-- Creating a form with an encoded name for the portal, so if there -->  
<!-- are two forms with the same name, there won't be any conflicts. -->  
<!-- Also notice that the details action is extracted from the request. -->  
<form name="<portletAPI:encodeNamespace value='CD_form' />"  
  action="<%= (String)request.getAttribute("details") %>"  
  method="post">  
  
<SELECT size="1" name="uid">  
  <!-- Connecting to the Domino server and extracting the view -->  
  <!-- The *webuser is used to enable single sign on. -->  
  <Domino:view viewname='CustomersByName'  
    dbserver='CN=itsotest-dom/0=itsportal'  
    dbname='apps/customer.nsf'  
    user='*webuser'  
    host='itsotest-dom'>  
    <option value="" selected>-Select a customer-</option>  
    <!-- Now we are looping through the view creating options for the -->
```

```

        <!-- different customers -->
        <Domino:viewloop>
            <OPTION value="<Domino:unid/>">
                <Domino:viewitem name="Customer Name"/>
            </OPTION>
        </Domino:viewloop>
    </Domino:view>
</SELECT></P>
<!-- Standard submit HTML button -->
<P><INPUT type="submit" name="Details" value="View Details"></P>
</form>

```

This code will insert a text message instructing the user to select an available customer from the list. Followed by a `<form>` tag that will use the POST method to submit the information to WebSphere Portal, the action on this form will trigger the action event details.

Next, a custom JSP tag for a Domino view is placed. This opens a session with the Lotus Domino server, and a select HTML field will loop using the `<Domino:viewloop>`.

Inside the loop we extract two pieces of information: the universal ID of the customer document using the `<Domino:unid/>`, and the column information corresponding to the Customer Name.

CustomerDetails.jsp

1. Right click the `/jsp/CustomerDetails` folder and select New → JSP File.
2. Name the new JSP file `CustomerDetails.jsp`, and select the checkbox to specify this JSP is a fragment. Click Next.
3. Click on the Add Tag Library button.
4. Select all three tld libraries displayed. Add prefixes to the three libraries: Domino to the `domtags.tld` library, util to the `domutil.tld` library, and portletAPI to the `portlet.tld`, as shown in Figure 4-32. Click OK, then Finish to create the JSP.

The prefixes are needed so you can associate each tag with a JSP tag library.

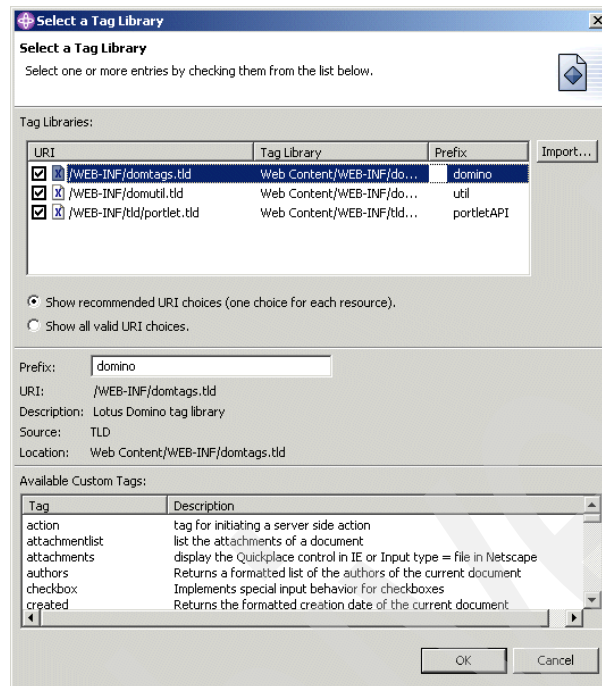


Figure 4-32 Insert Tag Libraries to the new CustomerDetails.jsp file

5. In the Source view, insert the following code beneath the Tag Library definitions:

Example 4-11 CustomerDetails.jsp fragment initial code

```
<H3>Customer Details</H3>
<P>
<!-- Connecting to the Domino Server and extracting a document with the UID-->
<Domino:document
  dbname='apps/customer.nsf'
  dbserver='CN=itsotest-dom/0=itsportal'
  host='itsotest-dom'
  user='*webuser'
  schema='Customer'
  uid='<%= (String)request.getSession().getAttribute("uidDoc")%>'>
<table border="1">
  <tr>
    <TH>Customer Name</TH>
    <!-- Here we are extracting the customerName item from the document-->
    <td><Domino:item name="customerName" /></td>
  </tr>
  <tr>
    <TH>Customer Number</TH>
```

```

        <!-- Here we are extracting the customerNumber item from the document-->
        <td><Domino:item name="customerNumber" /></td>
    </tr>
    <TR>
        <TH>Customer Address</TH>
        <!-- Here we are extracting the customerAddress item from
        the document-->
        <TD><Domino:item name="customerAddress" /></TD>
    </TR>
    <tr>
        <TH>Account Owner</TH>
        <td>
            <!-- Here we are extracting the ownerName and ownerNumber-->
            <!-- item from the document-->
            <Domino:item name="ownerName"/>
            (<Domino:item name="ownerNumber" />)
        </td>
    </tr>
    <TR>
        <TH>Date Created</TH>
        <!-- Here we are extracting the DateCreated item and formatting it-->
        <TD><Domino:item name="DateCreated" format="DATE=SHORT"/></TD>
    </TR>
    <tr>
        <TH>Comments</TH>
        <!-- Here we are extracting the Comments item.
        It is a rich text field.-->
        <td><Domino:item name="Comments" /></td>
    </tr></table>
</Domino:document>
</P>
<P>
    <!-- We are inserting a back button with the returnUrl
    created in the portlet-->
    <INPUT type="submit" name="Back" value="Back"
    onClick="window.location.href='
    <%= (String)request.getAttribute("CD_back")%>'">
</P>

```

This code opens a connection with the Lotus Domino server and extracts the document that has the UID corresponding to the uidDoc attribute we stored on the session object.

To extract the field information from the Domino document, we use the <Domino:item> tag.

6. Save the file.

Update the deployment descriptors

1. Open the web.xml file located in the /WEB-INF folder.
2. Go to the Servlets tab.
3. Click the Add button.
4. Select the CustomerDetails class.
5. Add a URL mapping; set it to /CustomerDetails/*.
6. Save and close the web.xml file.
7. Open the portlet.xml file located in the same folder.
8. Click the Add portlet button.
9. Select the CustomerDetails servlet, which is not used, and click OK.
10. Change the display name to Customer Details portlet and its ID to CustomerDetails.
11. Click the Concrete Portlet Application folder.
12. Click the Add Concrete Portlet button.
13. Click the CustomerDetails portlet, which is not used, and click OK.
14. Change its Display Name and Title to Customer Details portlet.
15. Add a description if needed.
16. Save the portlet.xml file.

Deploy the portlet

1. Export the project from the J2EE Navigator view by right-clicking over the project and selecting Export.
2. Now we can test the new portlet we have just created. We follow the same steps as the deployment of the previous portlet, except instead of using the install wizard we go to the Portlet Applications page and perform an update to the existing .WAR file.

The resulting portlet will look similar to Figure 4-33.

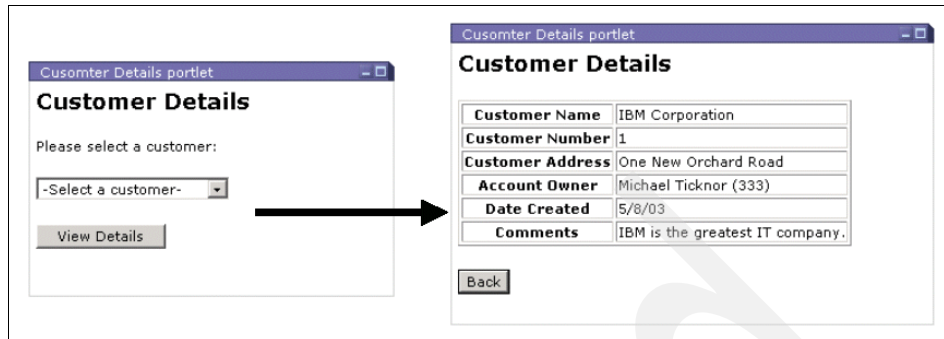


Figure 4-33 Customer Detail initial portlet

Customer Contacts portlet

Now we describe how to build the third portlet, which is the Customer Contacts portlet. This portlet displays an initial page with a drop-down list where the user selects the customer to query. After a customer is selected, the portlet performs a full text search on the Contacts By Customer view in the customer.nsf database. Finally, it displays the result of the search in the portlet. Since the behavior of this portlet is similar to that of the Customer Detail, we will reuse some of the functionality already created.

CustomerContacts.java file

1. Copy the CustomerDetails.java portlet class and paste it in the same portlet package, but with a name of CustomerContacts.java.
2. Open the newly copied CustomerContacts.java file.
3. From the doView() and actionPerformed() methods, since the lookup will be by customer name, change the attribute from uidDoc to customerNameAttr.
4. Change the name of the action from details to contacts; this will be done in the doView() and actionPerformed() methods.
5. In the actionPerformed() method, modify the request.getParameter() parameter from uid to customerName.
6. Look for the returnUrl object created and change the name of the URI stored as an attribute on the request to CC_back.
7. Finally, modify the redirection directory on the doView() methods to point to the /jsp/CustomerContacts folder and CustomerContacts JSP file.

The doView() and actionPerformed() methods should look like Example 4-12.

Note: Modify just the doView() and actionPerformed() methods; leave the getJspExtension() unmodified.

Example 4-12 Customer Contacts portlet initial doView() and actionPerformed() methods

```
public void doView(PortletRequest request, PortletResponse response) throws
    PortletException, IOException {
    if(request.getPortletSession().getAttribute("customerNameAttr") != null){
        PortletURI detailURI = response.createURI();
        DefaultPortletAction detailAction = new
            DefaultPortletAction("contacts");
        detailURI.addAction(detailAction);
        request.setAttribute("contacts", detailURI.toString());
        PortletURI returnURI = response.createReturnURI();
        request.setAttribute("CC_back", returnURI.toString());
        getPortletConfig().getContext().include(
            "/jsp/CustomerContacts/CustomerContacts."+getJspExtension(request),
            request, response);
    } else {
        PortletURI detailURI = response.createURI();
        DefaultPortletAction detailAction =new DefaultPortletAction("contacts");
        detailURI.addAction(detailAction);
        request.setAttribute("contacts", detailURI.toString());
        getPortletConfig().getContext().include(
            "/jsp/CustomerContacts/View."+getJspExtension(request), request,
            response);
    }
}

public void actionPerformed(ActionEvent event) throws PortletException {
    DefaultPortletAction action=(DefaultPortletAction)event.getAction();
    PortletRequest request=event.getRequest();
    if(action!=null){
        if(action.getName().equals("contacts")) {
            request.getPortletSession().setAttribute(
                "customerNameAttr",request.getParameter("customerName"));
        }
    }
}
```

The changes to the CustomerContacts.java file are highlighted in bold in Example 4-12 on page 190. Now that the CustomerContacts.java file is set, we will continue with the JSP files.

View.jsp file

1. Copy the CustomerDetails.jsp and View.jsp files from the /jsp/CustomerDetails folder and paste them on the /jsp/CustomerContacts folder.
2. Rename the newly pasted CustomerDetails.jsp file to CustomerContacts.jsp.

3. Open the just pasted View.jsp file.
4. Modify the page header from Customer Details to Customer Contacts.
5. Modify the form tag, action attribute from details to contacts, which is the name of the action we created on our CustomerContacts portlet.
6. Modify the Select HTML field name from uid to customerName.
7. Modify the <Domino:viewitem> tag, name attribute so that it extracts the Employer instead of the Customer Name.
8. Change the Select HTML field value from <Domino:unid/> to <Domino:viewitem name="Employer"/>.
9. On the <Domino:view> tag, modify the attribute viewname to point to the view ContactsByCustomer.
10. Just after the <Domino:viewloop> insert a <Domino:ifcategoryentry> and close the </Domino:ifcategoryentry> tag just before the </Domino:viewloop> tag.
This will filter the view and present only the category entries.
11. Modify the View Details button at the end so that it displays View Contacts.
12. Save and Close the View.jsp file.

CustomerContacts.jsp file

1. Open the /jsp/CustomerContacts/CustomerContacts.jsp file.
2. Modify the label from Customer Details to Customer Contacts.
3. Erase the code in between and *including* the <Domino:document> tag, since we are going to display a view instead of a document.
4. Insert the JSP code shown in Example 4-13 where the <Domino:document> tag resided.

Example 4-13 CustomerContacts.jsp fragment initial contents

```
<%! String searchString="";%>
<% searchString="FIELD customerName contains "
    +request.getSession().getAttribute("customerNameAttr");%>
<Domino:view
    viewname='ContactsByCustomer'
    dbserver='CN=itsotest-dom/0=itsportal'
    dbname='apps/customer.nsf'
    user='*webuser'
    host='itsotest-dom'
    ftsearch='<%=SearchString%>'>
<table border="1">
<tr>
    <TH>Contact Name</TH>
```

```

        <TH>Phone Number</TH>
    </tr>
    <Domino:viewloop id="myviewloop">
        <tr>
            <Domino:ifdocumententry>
                <%
if(myviewloop.getDocument().getItemValueString("customerName").equals(request.g
etSession().getAttribute("customerNameAttr"))) {%>
                    <td>
                        <Domino:viewitem col="2"/>
                    </td>
                    <td>
                        <Domino:viewitem col="3"/>
                    </td>
                <% } %>
            </Domino:ifdocumententry>
        </tr>
    </Domino:viewloop>
</table>
</Domino:view>

```

You will find that the `<Domino:view>` tag has a new attribute: `ftsearch`. This attribute lets us do a full text search based on a field as displayed in the example, where we will search based on a field called `CustomerName`.

Note: Use of the `ftsearch` attribute of the `<Domino:view>` tag requires that the referenced view be full text indexed prior to the search request.

Also you will find that the `<Domino:viewloop>` tag has an attribute called `name`. This allows us to define a Domino Java object and use it inside our JSP code. In this JSP we query each of the returned elements to make sure the full text search returned the exact documents that we were looking for. Other tags can have the `ID` attribute that will convert a JSP tag into a Domino Java object that can be referenced in the code. Information on the construction of portlets with these Domino Java objects is in the next chapter.

Also note that the `<Domino:viewitem>` tag is extracting information by the column number attribute; you can also use the `name` attribute to extract the column value by referencing the column name, like in the previous portlet's `View.jsp` file.

Finally, we introduced another JSP tag called `<Domino:ifdocumententry>`, which basically filters out the categorization entries from the view.

5. Modify at the end of the JSP file the Back button so that it references the `CC_back` URI created on the `CustomerContacts.java` portlet.

Update the deployment descriptors

1. Open the web.xml file located on the /WEB-INF folder.
2. Go to the Servlets tab.
3. Click the Add button.
4. Select the CustomerContacts class.
5. Add a URL mapping; set it to /CustomerContacts/*.
6. Save and close the web.xml file.
7. Open the portlet.xml file located on the same folder.
8. Select the Add portlet.
9. Select the CustomerContacts servlet, which is not used, and click OK.
10. Change the display name to Customer Contacts portlet and its ID to CustomerContacts.
11. Select the Concrete Portlet Application folder.
12. Click the Add Concrete Portlet button.
13. Select the CustomerContacts portlet, which is not used, and click OK.
14. Change its Display Name and Title to Customer Contacts portlet.
15. Add a description if needed.
16. Save the portlet.xml file.

Deploy the portlet

Now we are all set with our third portlet, just deploy it as we did with our previous portlet and you should see something like shown in Example 4-34 on page 193.



Figure 4-34 Customer Contacts initial portlet

Customer Sales Activities portlet

The Customer Sales Activities portlet displays information just like the two portlets already described, with a JSP file containing a drop-down list of the customers, but this time with sales activities. A difference from previous portlets is that it will access an independent sales database. After a customer is selected,

the portlet loads a Domino view that will filter and display the documents and not the category entries.

CustomerSalesActivities.java

1. Create the portlet class by copying the CustomerContacts.java class that resides on the portlet package.
2. Paste the file in the same package and rename it CustomerSalesActivities.java.
3. Open the newly pasted CustomerSalesActivities.java file.
4. In the doView() method, rename the DefaultPortletAction from contacts to activities.
5. On the same method, rename the key that is used to store the action URI on the request object from contacts to activities.
6. Continuing on the same method, modify the folder for the include sentences from /jsp/CustomContacts to /jsp/CustomSalesActivity. Also modify the reference to the file /jsp/CustomContacts/CustomContacts.jsp to /jsp/CustomSalesActivity/CustomSalesActivities.jsp.
7. Now on the actionPerformed() method modify the action name comparison from contacts to activities.
8. Both the doView() and actionPerformed() methods should look like Example 4-14.

Example 4-14 Customer Sales Activities portlet initial doView() and actionPerformed() methods

```
public void doView(PortletRequest request, PortletResponse response) throws
    PortletException, IOException {
    if(request.getPortletSession().getAttribute("customerNameAttr") != null){
        PortletURI detailURI = response.createURI();
        DefaultPortletAction detailAction =
            new DefaultPortletAction("activities");
        detailURI.addAction(detailAction);
        request.setAttribute("activities", detailURI.toString());
        PortletURI returnURI = response.createReturnURI();
        request.setAttribute("CSA_back", returnURI.toString());
        getPortletConfig().getContext().include(
            "/jsp/CustomSalesActivity/CustomSalesActivities."+
            getJspExtension(request), request, response);
    } else {
        PortletURI detailURI = response.createURI();
        DefaultPortletAction detailAction = new
            DefaultPortletAction("activities");
        detailURI.addAction(detailAction);
        request.setAttribute("activities", detailURI.toString());
    }
}
```

```

        getPortletConfig().getContext().include(
            "/jsp/CustomerSalesActivity/View."+
            getJspExtension(request), request, response);
    }
}
public void actionPerformed(ActionEvent event) throws PortletException {
    DefaultPortletAction action=(DefaultPortletAction)event.getAction();
    PortletRequest request=event.getRequest();
    if(action!=null){
        if(action.getName().equals("activities")) {
            request.getPortletSession().setAttribute(
                "customerNameAttr",request.getParameter("customerName"));
        }
    }
}
}

```

You can see the changes in bold in Example 4-14. Also notice that there is no change on the `customerName` parameter and `customerNameAttr`. This is because our search on the view will be done by the customer name. In addition, this will help the Click to Action broker to associate both actions as similar; this is explained in detail later in this chapter.

View.jsp file

1. Copy the View.jsp file from the /jsp/CustomerContacts folder and paste it in the /jsp/CustomerSalesActivity folder.
2. Open the file and inspect it in the source view.
3. Modify the page header from Customer Contacts to Customer Sales Activities.
4. In the form tag, modify the action attribute to reference from contacts to activities.
5. In the <Domino:view> tag, modify the attribute viewname to saByCustomer and the database to apps/sales.nsf.
6. In the option HTML tag, modify the value and name so that the <Domino:viewitem> name attribute is Customer.
7. Finally, on the button at the bottom of the JSP, change the value to View Sales Activities.

CustomerSalesActivities.jsp file

1. Copy the CustomerContacts.jsp file in the /jsp/CustomerContacts folder to the /jsp/CustomerSalesActivity folder and rename it CustomerSalesActivities.jsp.
2. Modify the file so that it has code similar to that shown in Example 4-15.

Example 4-15 CustomerSalesActivities.jsp initial contents

```
<%@ taglib uri="/WEB-INF/domtags.tld" prefix="Domino" %>
<%@ taglib uri="/WEB-INF/domutil.tld" prefix="util" %>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init/>
<H3>Customer Sales Activities</H3>
<P>
<%! String searchString="";%>
<% searchString="FIELD CustomerName contains
    "+request.getSession().getAttribute("customerNameAttr");%>
<Domino:view
    viewname='saByCustomer'
    dbserver='CN=itsotest-dom/0=itsportal'
    dbname='apps/sales.nsf'
    user='*webuser'
    host='itsotest-dom'
    ftsearch='<%=SearchString%>'>
<table border="1">
  <tr>
    <TH>Date</TH>
    <TH>Activity</TH>
    <TH>Sales Person</TH>
    <TH>Contact</TH>
    <TH>Made Sale?</TH>
  </tr>
  <Domino:viewloop id="myviewloop">
    <tr>
      <Domino:ifdocumententry>
        <%
if(myviewloop.getDocument().getItemValueString("CustomerName").equals(request.g
etSession().getAttribute("customerNameAttr"))) {%>
          <td>
            <Domino:viewitem col="2"/>
          </td>
          <td>
            <Domino:viewitem col="3"/>
          </td>
          <td>
            <Domino:viewitem col="4"/>
          </td>
          <td>
            <Domino:viewitem col="5"/>
          </td>
          <td>
            <Domino:viewitem col="6"/>
          </td>
        <% } %>
      </Domino:ifdocumententry>
```

```

        </tr>
    </Domino:viewloop>
</table>
</Domino:view>
</P>
<P>
    <INPUT type="submit" name="Back" value="Back"
        onClick="window.location.href='
            <%= (String)portletRequest.getAttribute("CSA_back")%>'">
</P>
<P></P>

```

This is a very interesting JSP since first it points to and extracts information from the Sales database, which could be unrelated to the Customers database. But as we discuss in the Click to Action topic later in this chapter, we will connect these two applications at the portal level.

Also, if you inspect the `<Domino:viewloop>` tag, you will see that there is a name attribute again. This name attribute represents a Domino Java object and is later used in the same JSP file in a scriptlet code to filter out activities not related to our customer.

Update the deployment descriptors

1. Open the `web.xml` file located on the `/WEB-INF` folder.
2. Go to the Servlets tab.
3. Click the Add button.
4. Select the `CustomerSalesActivities` class.
5. Add a URL mapping and set it to `/CustomerSalesActivities/*`.
6. Save and close the `web.xml` file.
7. Open the `portlet.xml` file located in the same folder.
8. Click Add portlet.
9. Select the `CustomerSalesActivities` servlet, which is not used, and click OK.
10. Change the display name to `Customer Sales Activities portlet` and its ID to `CustomerSalesActivities`.
11. Select on the `Concrete Portlet Application` folder.
12. Click the `Add Concrete Portlet` button.
13. Select the `CustomerSalesActivities` portlet, which is not used, and click OK.
14. Change its `Display Name` and `Title` to `Customer Sales Activities portlet`.
15. Add a description if needed.

16. Save the portlet.xml file.

Deploy the portlet

Deploy the portlet in the same manner that you used for the previous ones. The new portlet should appear something like Figure 4-35.

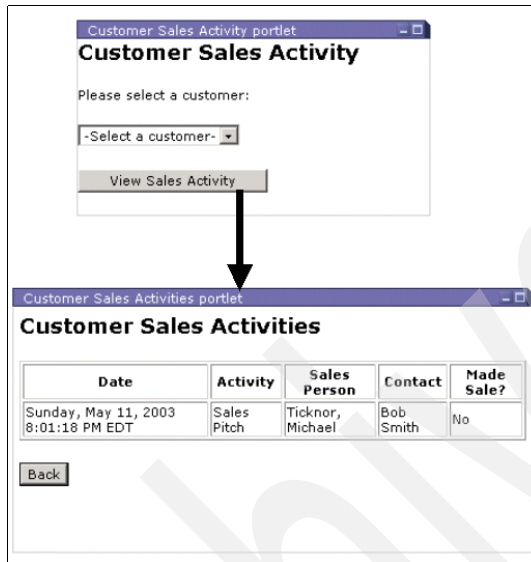


Figure 4-35 Customer Sales Activities initial portlet

We have now finished the first phase of our portlet building. We constructed four portlets from different Domino databases that extract information based on the user input.

Next we explore techniques to enhance this initial transformation. In 4.6, “Integration via Click to Action” on page 199 we describe a technique to enable our portlets to communicate between each other, even between heterogeneous Domino databases.

4.5.4 Conclusions to the custom Domino tags integration technique

The inclusion of JSP tags from different services lets us add value to our portlets. We showed how to include custom Domino JSP tags. Some of the benefits of this option are:

- ▶ The developer doesn't have to know Java programming in depth to build portlets that extract information from Domino.
- ▶ The developer doesn't have to know Domino object model in depth to access data in Domino.

- ▶ JavaServer Pages (JSP) are a J2EE technology that can offer simplicity as a programming model and the robust characteristics of Java.
- ▶ IBM has the tools necessary to develop portlets that integrate Domino applications, including collaboration features.

4.6 Integration via Click to Action

This important topic is included in the current chapter since the invocation is done through JSP tags. But as we explain later in this section, Click to Action incorporates other technologies as well, including Web services, WSDL, portlet descriptors, and others. We explain these technologies as we go through the implementation of Click to Action.

4.6.1 Click to Action

Click to Action (sometimes referred as Click2Action or C2A) provides a framework for inter-portlet communication that simplifies users' interactions with portlets on a portal page. With a simple click, a user can transfer data from a source portlet to one or more target portlets, causing the target to react to the action and display a new view with the results.

The Click to Action framework includes a runtime that automatically matches sources with compatible targets (based on type information) and inserts clickable icons associated with sources on portlet pages. When the user clicks an icon next to a particular source, they are presented with a pop-up menu containing the list of targets for the action. After the user selects a specific target, the Click to Action runtime delivers the data to the target in the form of the corresponding portlet action. The portlet does not need to distinguish between an action initiated by user interaction with its own page segment and action initiated using the Click to Action route. This keeps the programming effort to a minimum, allowing Click to Action portlets to follow the normal portlet programming model.

Features of the Click to Action model

Click to Action provides an easy, menu-driven method to transfer compatible data between portlets, eliminating manual data entry from one portlet to another and avoiding the learning normally necessary to discover actions on target portlets which are compatible with sources on the current page. Click to Action includes these additional benefits:

- ▶ Broadcast source data to all matching actions on the page.

A user can specify the target of an action or broadcast the action to all portlets on the page. This feature is available in the source portlet when the broadcast attribute is specified for the `<c2a:encodeProperty/>` tag.

- ▶ Chained propagation of data transfer.

Sending data to one portlet can cause that portlet to send data to another portlet, which can in turn transmit data. This feature supports the synchronization of multiple portlet views in a single request-response cycle. For example, transferring the order ID to the Order Details portlet also triggers the transfer of the tracking ID for the order to the tracking details portlet, which in turn triggers the transfer of the customer name associated with the order to the customer details portlet, causing all three to display information pertaining to the same order.

This feature can be enabled by specifying output parameters in the target portlet's WSDL. When the target receives data from a source, this can cause the transfer of one or more output parameters declared in the WSDL file. As pictures in Figure 4-36, the propagation of information can chain a series of events leading to a complete flow of interactions.

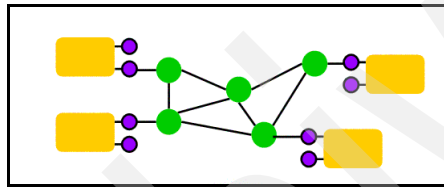


Figure 4-36 Click to Action chained propagation

- ▶ Scatter multiple related sources to targets on the page with a single click.
In addition to directing the data transfer to a particular portlet, you can broadcast the information that is sent to all the listener portlets. As a result of this broadcast, all portlets on the page display information related to the transferred information. This scatter feature, combined with the chained data transfer feature, results in the synchronization of all the information on the page through a single user click.
- ▶ Simple enablement for portlet development.
 - For a portlet to be a source of data, programmers can use a custom JSP tag library to flag sharable data on their output pages. The tags require a data type to be specified, as well as a specific value corresponding to an instance of this type.
 - For a portlet to be a target, programmers describe a subset of their portlet actions, including type information for the action parameters. The format for the action description is WSDL, with some custom extensions.

Concepts and design of Click to Action

Click to Action allows end-user triggered information interchange between independently developed portlets. Each portlet needs to provide information

about data objects that it could share with other portlets, either as a producer or a consumer. Each sharable data object is identified with an XML type. The Click to Action broker is a runtime entity that processes information about sharable data objects on a page and matches producers of the data with consumers using type matching. Based on the match information, the broker generates a special icon next to sources of data, which is used to display the pop-up menu of matching actions. When the end-user chooses an action from the menu, it is intercepted by the broker, which then transfers the data value to the chosen target.

Source portlets (portlets that contribute data objects that can be shared with other portlets) use a set of custom JSP tags to specify the type and value of the data being contributed. At the point where such a tag occurs, the Click to Action broker inserts the markup that can display the pop-up menu of action choices for the user. Advanced options on the tags allow the programmer to indicate whether a broadcast action is to be added to the menu, and allow the scattering of a set of data values (rather than a single data value) to all portlets.

Target portlets (portlets that accept data from other portlets) declare a set of actions which can be invoked on the portlet. The actions are implemented by the portlet as normal portlet actions. The actions are declared using WSDL, with a custom binding extension that specifies the mapping from the abstract action declaration to the actual action implementation. Associated with each action is a single input parameter described by an XML type and one or more output parameters, each described by an XML type. The input parameter's type is used for matching the action to sources, and its value is filled in when the end-user triggers the action using Click to Action. The output parameters, if specified, are used to automatically trigger other compatible actions (ones which can consume the same type) on other portlets every time the action executes (this may be used to trigger chains of related actions). The choice of WSDL for declaring the actions was influenced by two considerations:

- ▶ The need for a standard format rather than a custom format
- ▶ The ability in WSDL to plug in descriptions of different types of implementing entities, providing a future growth path for the Click to Action technology

Another interesting aspect of the design is the use of normal portlet actions to deliver data from other portlets. This approach allows the reuse of actions which are used to interact with the portlet directly, and in many cases allows pre-existing portlets to be made into Click to Action targets simply by declaring all or part of their actions using WSDL.

Figure 4-37 is a high-level diagram of the Click to Action sequence of events that takes place.

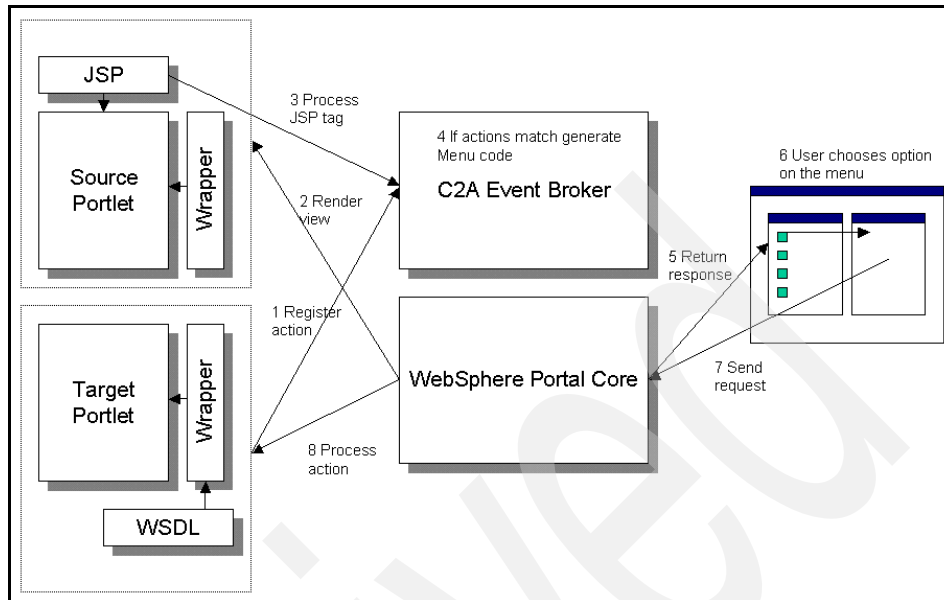


Figure 4-37 Click to Action sequence diagram

The Click to Action runtime is composed of a generic wrapper portlet, which is used to wrap each portlet enabled for Click to Action, and a broker component. These components also interface with the WebSphere Portal core runtime. The purpose of the wrapper is to intercept calls to the application portlet and interface with the broker appropriately to transparently register actions supported by the portlet, transfer data from source to target portlets, and so forth. The broker centrally maintains a repository of actions on each portlet along with the parameter type information, performs source and target matches at runtime using this information, and generates additional markup to allow users to trigger data transfers across portlets. The WebSphere Portal core runtime performs the functions of generating the portal pages, receiving browser requests, invoking callbacks on portlets, and so forth. Finally, JSPs associated with source portlets use custom tags to declare sharable sources of data, and target portlets provide a WSDL file declaring portlet actions which may be invoked by the Click to Action runtime.

The Click to Action flow is designed to account for the portlet event processing model, which is described in Portlet events. The portal programming model involves an event phase and a render phase in each request-response cycle. During the event phase, an action may be delivered on one portlet. If Click to Action is used, this may result in other actions being triggered on other portlets. The event phase is followed by the render phase, in which each portlet is asked to return markup, which is then aggregated in a single page. The markup may

embed actions that can be invoked by the user. The page is then returned to the client (such as a browser).

A typical request-response flow involving Click to Action is illustrated in Figure 4-37. This comprises the following steps:

1. During portlet initialization, the wrapper processes any action WSDL file associated with the application portlet and registers the actions with the broker.
2. During the render phase of a request cycle, JSPs associated with Click to Action source portlets are processed.
3. The custom Click to Action tags result in calls to the Click to Action event broker, which determines matching actions on the page based on type information.
4. The Click to Action broker generates additional code to display the Click to Action icon used to invoke the pop-up menu.
5. After all render phase portlet callbacks have been completed, the portal assembles the response page and returns it to the client.
6. The end user can click on the Click to Action icon for a source to view a menu of compatible actions on the page and select one.
7. A new request is generated, containing the chosen source and action information, and is sent to the portal.
8. The portal core runtime delivers the action to the target portlet. This is intercepted by the wrapper, which may interact with the broker to further process the request before delivering the action to the target.

While all portlet actions are intercepted by the wrapper, actions which are invoked through direct interaction with the portlet (as opposed to interaction through Click to Action) are passed through transparently to the portlet.

4.6.2 Considerations

We can integrate portlets that expose Domino data through the WebSphere Portal Click to Action Broker.

4.6.3 Implementation of the technique

As we mentioned in the introduction to this topic, the Click to Action technology included on the WebSphere Portal enables communication between portlets. In our case, it was used to extract information from our example Sales and Customer Lotus Domino applications.

In this section we describe how to enable the portlets with Click to Action capabilities. We focus first on the Customer List portlet, which is the source portlet, and then on enabling the target portlets, which are the remaining portlets (Customer Details, Customer Contacts and Customer Sales Activities) on the page.

Preparing the portlet project

Use the following steps to prepare the portlet project before starting the enablement of the portlets.

Open the WebSphere Studio Application Developer tool and the Web perspective.

1. Open the CustomerJSPPortlet project.
2. Right-click the /WEB-INF/tld folder and select Import.
3. On the Import dialog, select File System and click Next.
4. Select the Directory button and browse to the WPS-HOME\c2a\tld directory, where WPS-HOME is the WebSphere Portal installation directory.
5. Select the c2a.tld file and click Finish.

Figure 4-38 shows the import parameters we used.

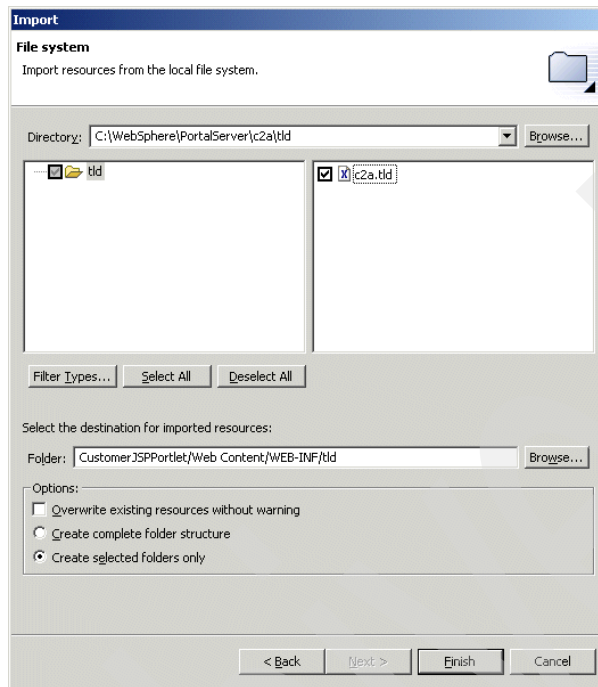


Figure 4-38 Importing the Click to Action tag library

6. Right-click the /WEB-INF/lib folder and select import.
7. On the Import dialog, select File System and click Next.
8. Select the Directory button and browse to the WPS-HOME\c2a\lib directory, where WPS-HOME is the WebSphere Portal installation directory.
9. Select the pbportlet.jar file and click Finish.

The portlet project is now ready to support Click to Action functionality.

Enabling the source portlet (Customer List)

The process of enabling the source portlet is very simple.

First, you have to know what parameters are to be transmitted to the target portlets. The parameter transmitted should be the same one expected on the action of the target portlet. The target portlet parameters and actions expected are outlined in Table 4-6.

Table 4-6 Click to Action target portlet parameters and actions

Portlet name	Parameter	Action
Customer Details	uid	details
Customer Contacts	customerName	contacts
Customer Sales Activities	customerName	activities

Use the following steps to start working on the portlet:

1. Open the /jsp/CustomerList/View.jsp file.
2. At the very beginning of the View.jsp file insert the following tag library import statement:

```
<%@ taglib uri="/WEB-INF/tld/c2a.tld" prefix="C2A" %>
```

3. Add an attribute to the existing <Domino:viewloop> tag. This attribute is named id and we give it a value "myviewloop". The tag should look like this:

```
<Domino:viewloop id="myviewloop">
```

This is done so that you can extract information from the looping documents using the Domino Java objects API.

4. On the table row displaying the resulting loop, before the <Domino:viewitem col="1"/> tag, insert the following tags:

```
<!-- Note the namespace we chose is arbitrary -->
```

```
<C2A:encodeProperty namespace="http://www.ibm.com/customer"
type="CustomerIDType"
value='<%=myviewloop.getDocument().getUniversalID()%' />
```

```
<C2A:encodeProperty namespace="http://www.ibm.com/customer"
type="CustomerNameType" broadcast="true"
value='<%=myviewloop.getDocument().getItemValueString("customerName")%' />
```

There are several things to note about the two lines inserted.

First, notice that both tags are Click to Action JSP custom tags, and they have several attributes. Table 4-7 is a description of the purpose of the available Click to Action encode property attributes.

Table 4-7 Click to Action encode property JSP tag attributes

Attribute	Description
type	Specifies the type of data sent by the source portlet. Required. The data type is defined in the WSDL, which will be defined when we enable the target portlets. The Click to Action runtime uses type and namespace attributes to match source data to actions on targets.

Attribute	Description
namespace	Specifies the namespace for the type. Required. This attribute is used to group related types in a single domain.
value	Specifies the data to be sent by the source portlet. Required. The Click to Action runtime sends the value of the user's selections to the target actions.
broadcast	Indicates whether the source data can be broadcast to all target portlets with matching actions. False is the default setting. Optional. If the value is true, the generated menu displays an additional item to broadcast the source to all matching targets.
generateMarkupWhen Nested	Indicates that this tag should generate markup even when nested within the <c2a:encodeProperties/> tag. This attribute is optional; the default is false. If this tag is not nested, the attribute is ignored. For our initial example this attribute was not used.

Also, notice that inside the value attribute you are inserting a Java scriptlet which will get the parameters required to feed the target portlets.

Since there are two portlets that will require the same customerName parameter, just one tag is required.

5. Save and Close the View.jsp file.

Note: When you save the View.jsp file, a JspTranslate error is reported by WebSphere Studio. Ignore this message since at run time the correct classes will be available.

You are now done enabling the source portlet.

Enabling the target portlets

Create a folder to contain the WSDL descriptor files for your target portlets with these steps:

1. Right-click the Web Content folder on your CustomerJSPPortlet project.
2. Select New → Folder.
3. Name the folder wsd1 and click Finish.

Target portlets requirements

There are some requirements that the target portlet class must follow:

- ▶ Portlet actions must use the DefaultPortletAction object for action processing.

- ▶ Portlet actions must accept a single basic parameter. This parameter appears in the PortletRequest object.
- ▶ The action must also be invocable at any time. That is, there should not be a case where the action is not invocable when the portlet is in a certain state. This is because the set of actions associated with a target portlet is statically declared; the C2A broker assumes that the same set is available in each request cycle.

Fortunately, we developed our portlets following these restrictions. Go through them and you will find that the actions defined are appropriate, and each action requires only one parameter.

Enabling the Customer Details portlet

You do not have to modify any Java code to enable the target portlets; the task is basically a definition process that comprises the creation of a WSDL descriptor file and the modification of the deployment descriptors.

First, create the WSDL file with these steps:

1. Right-click the /wsdl folder and select New → Other.
2. Select Simple and File in the dialog, then click Next.
3. Name the file CustomerDetails.wsdl.
4. Click Finish.
5. Copy the code in Example 4-16 into the newly created file.

Example 4-16 Click to Action CustomerDetails.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CustomerDetail_Service"
  targetNamespace="http://www.ibm.com/customer"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/customer"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <types>
    <xsd:simpleType name="CustomerIDType">
      <xsd:restriction base="xsd:string">
        </xsd:restriction>
      </xsd:simpleType>
    </types>
    <message name="customerDetailsRequest">
      <part name="custid" type="tns:CustomerIDType"/>
    </message>
    <portType name="CustomerDetail_Service">
      <operation name="customerDetails">
```

```

        <input message="tns:customerDetailsRequest"/>
    </operation>
</portType>
<binding name="CustomerBinding" type="tns:CustomerDetail_Service">
    <portlet:binding/>
        <operation name="customerDetails">
            <portlet:action name="details" caption="Customer Details"
                description="Get details for specified Customer"/>
            <input>
                <portlet:param name="uid" partname="custid"/>
            </input>
        </operation>
    </binding>
</definitions>

```

The WSDL file describes how the target portlet exposes its services. Note that in the definitions open tag we specify the namespace we are using as `http://www.ibm.com/customer`, which is the same as the one defined in the source portlet `<C2A:encodeProperty>` JSP tag.

We define a simple type called `CustomerIDType`, which is a simple xsd string type. Notice also that in the source portlet's first click to action `<C2A:encodeProperty>` JSP tag we are referencing this type.

- ▶ Also notice in the binding area that we are referencing an action with the same name, "details" that our portlet uses. It will store the value in a parameter called "uid" that will feed the portlets `actionPerformed()` method.

Note: When you save the `CustomerDetails.wsdl` file, several errors are reported by WebSphere Studio. Ignore these messages since they are corrected at run time.

Use the following steps to modify the deployment descriptors, beginning with the `web.xml` file.

1. Open in the `/WEB-INF` folder the `web.xml` file in Source mode.
2. Look for the `<servlet-class>` of the `CustomerDetails` servlet (portlet) and modify the class from `portlet.CustomerDetails` to the `com.ibm.wps.pb.wrapper.PortletWrapper` class.
3. Switch to the Servlet tab.
4. Select the `CustomerDetails` servlet, add an initialization parameter called `c2a-application-portlet-class`, and add a value of `portlet.CustomerDetails`. It should look like Figure 4-39.

▼ Details

Details of the selected servlet or JSP


Servlet class:

Display name:

Description:

▼ URL Mappings

The following URLs are mapped to this servlet:

 /CustomerDetails/*

▼ Initialization

The following initialization parameters are configured for this servlet:

Name	Value
c2a-application-portlet-class	portlet.CustomerDetail

Figure 4-39 Click to Action setting up the web.xml file

You can see that the Servlet class contains the previously modified class. This class is a wrapper that will load the class described by the c2a-application-portlet-class initialization parameter.

Next, modify the portlet.xml file:

1. Open the portlet.xml file.
2. Select the CustomerDetails concrete portlet.
3. Create a New setting parameter with a name c2a-action-descriptor and a value of /wsdl/CustomDetails.wsd1.
4. Save and Close the portlet.xml file.

Enabling the Customer Contacts portlet

Create the WSDL file using the following steps:

1. Right-click the /wsdl folder and select New → Other.
2. Select Simple and File in the dialog, then click Next.
3. Name the file CustomerContacts.wsd1.
4. Click Finish.

5. Copy the code in Example 4-17 into the newly created file.

Example 4-17 Click to Action CustomerContacts.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CustomerContacts_Service"
  targetNamespace="http://www.ibm.com/customer"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.ibm.com/customer"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<types>
  <xsd:simpleType name="CustomerNameType">
    <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </types>
<message name="customerContactsRequest">
  <part name="custName" type="tns:CustomerNameType"/>
</message>
<portType name="CustomerContacts_Service">
  <operation name="customerContacts">
    <input message="tns:customerContactsRequest"/>
  </operation>
</portType>
<binding name="CustomerContactsBinding" type="tns:CustomerContacts_Service">
  <portlet:binding/>
  <operation name="customerContacts">
    <portlet:action name="contacts" caption="Customer Contacts"
      description="Get contacts for a specified Customer"/>
    <input>
      <portlet:param name="customerName" partname="custName"/>
    </input>
  </operation>
</binding>
</definitions>
```

The WSDL file describes how the target portlet exposes its services. Note in the definitions open tag we are describing the same namespace, "http://www.ibm.com/customer" that we defined in the source portlet click to action <C2A:encodeProperty> JSP tag.

We are defining a simple type called "CustomerNameType" which is a simple xsd string type. Notice also that in the source portlet, in the second click to action <C2A:encodeProperty> JSP tag, we are referencing this type.

- ▶ In the binding area, notice that we are referencing an action with the same name "contacts" that one our portlest uses. It will store the value in a

parameter called "customerName" that will feed our CustomerContacts portlet's actionPerformed() method.

Note: When you save the CustomerContacts.wsdl file, several additional errors are reported by WebSphere Studio. Ignore these messages since they will be corrected at run time.

Next, modify the deployment descriptors, starting with the web.xml file.

1. Open in the /WEB-INF folder the web.xml file in Source mode.
2. Look for the <servlet-class> of the CustomerContacts servlet (portlet) and modify the class from portlet.CustomerContacts to the com.ibm.wps.pb.wrapper.PortletWrapper class.
3. Switch to the Servlet tab.
4. Select the CustomerContacts servlet and add an initialization parameter on the left called c2a-application-portlet-class and add a value of portlet.CustomerContacts.

Now, modify the portlet.xml file:

1. Open the portlet.xml file.
2. Select the CustomerContacts concrete portlet.
3. Create a New setting parameter with a name c2a-action-descriptor and a value of /wsdl/CustomerContacts.wsdl.
4. Save and close the portlet.xml file.

Enabling the Customer Sales Activities portlet

Create the WSDL file using the following steps:

1. Right-click the /wsdl folder and select New → Other.
2. Select Simple and File in the dialog, then click Next.
3. Name the file CustomerSalesActivities.wsdl.
4. Click Finish.
5. Copy the following code into the newly created file:

Example 4-18 Click to Action CustomerContacts.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="CustomerSalesActivity_Service"
  targetNamespace="http://www.ibm.com/customer"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:portlet="http://www.ibm.com/wps/c2a"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
```

```

xmlns:tns="http://www.ibm.com/customer"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<types>
  <xsd:simpleType name="CustomerNameType">
    <xsd:restriction base="xsd:string">
      </xsd:restriction>
    </xsd:simpleType>
  </types>
<message name="customerSalesActivityRequest">
  <part name="custName" type="tns:CustomerNameType"/>
</message>
<portType name="CustomerSalesActivity_Service">
  <operation name="customerSalesActivity">
    <input message="tns:customerSalesActivityRequest"/>
  </operation>
</portType>
<binding name="CustomerSalesActivityBinding"
  type="tns:CustomerSalesActivity_Service">
  <portlet:binding/>
  <operation name="customerSalesActivity">
    <portlet:action name="activities" caption="Customer Sales Activity"
      description="Get sales activities for a specified Customer"/>
    <input>
      <portlet:param name="customerName" partname="custName"/>
    </input>
  </operation>
</binding>
</definitions>

```

The WSDL file describes how the target portlet exposes its services. Note in the definitions opening tag we are describing the same namespace that we defined in the source portlet click to action <C2A:encodeProperty> JSP tag, "http://www.ibm.com/customer".

We are defining the simple type "CustomerNameType" which is a simple xsd string type. Notice also that in the source portlet, in the second click to action <C2A:encodeProperty> JSP tag, we are referencing this type.

- ▶ Also notice in the binding area that we are referencing an action with a name "activities" like the one our portlet uses. It will store the value on a parameter called "customerName" which will feed our portlet's actionPerformed() method.

Note: When you save the CustomerSalesActivities.wsdl file, several additional errors are reported by WebSphere Studio. Ignore these messages since they will be corrected at run time.

Modify the deployment descriptors, beginning with the web.xml file:

1. Open in the /WEB-INF folder the web.xml file in Source mode.
2. Look for the <servlet-class> of the CustomerSalesActivities servlet (portlet) and modify the class from portlet.CustomerSalesActivities to the com.ibm.wps.pb.wrapper.PortletWrapper class.
3. Switch to the Servlet tab.
4. Select the CustomerSalesActivities servlet and add an Initialization parameter on the left called c2a-application-portlet-class and add a value of portlet.CustomerSalesActivities.

Next, modify the portlet.xml file:

1. Open the portlet.xml file.
2. Select the CustomerSalesActivities portlet.
3. Create a New setting parameter with a name c2a-action-descriptor and a value of /wsdl/CustomerSalesActivities.wsdl.
4. Save and close the portlet.xml file.

Deploy the portlet

Now that all the settings are complete for your Click to Action portlets, deploy them just as you did in the previous exercises. You should see a page like the one depicted in Figure 4-40.

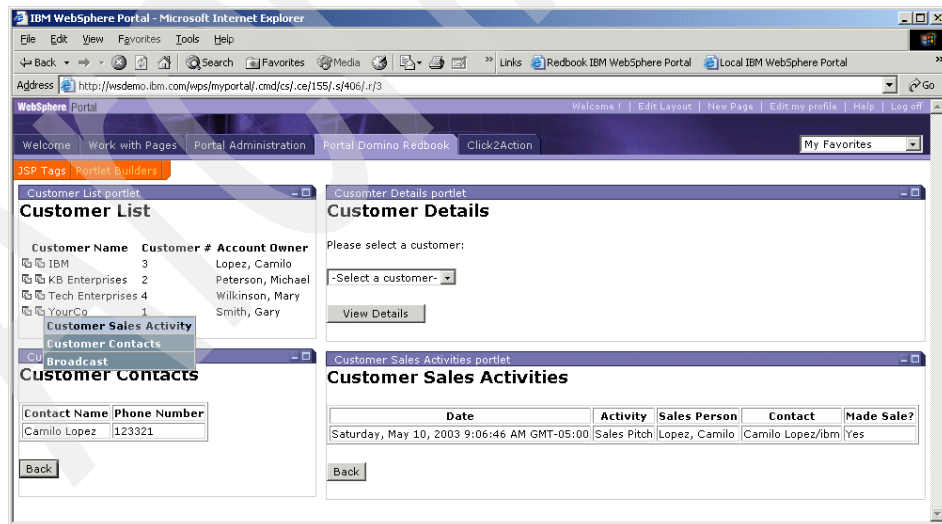


Figure 4-40 Click to Action enabled JSP tags portlets

The portlets are enabled with Click to Action, and the customers.nsf database with the sales.nsf database are integrated seamlessly at the portal interface level.

In the next section we describe how to add some finishing touches, specifically by adding people awareness to the portlets. But before we do that, let's take a look at some common pitfalls that developers run into when developing portlets that are Click to Action enabled.

Common pitfalls

- ▶ Be aware of the Namespace definition of the WSDL file; your Namespace should be unique and match the one described on the `<C2A:encodeProperty>` tag. Your namespace should be inside the definitions tag of the WSDL file in the attributes `targetNamespace` and `xmlns:tns`. Do not modify any other namespace.
- ▶ Remember that your portlets should implement the `ActionListener` interface to have the action called.
- ▶ The action inside your target portlet class should be a `DefaultPortletAction`.
- ▶ In WebSphere Portal 4.2, Click to Action is limited to Portlets that reside on the same page.

4.7 Integration via people awareness

This integration technique will add value to your portlets by exploiting the people awareness services provided by WebSphere Portal. This integration technique adds real-time collaboration among the people involved in the application.

4.7.1 People awareness

If you extend the portal to offer people awareness features, portal users will see references to people in collaborative portlets. These references to individuals and groups are links that let portal users see and contact others with whom they might want to work. Wherever people links appear, portal users can display a menu of actions for contacting and working with the individuals and groups named by the link. If you have enabled Lotus Sametime to work with the portal, people links include the online status indicator that shows whether a person is active, away, offline, or in a Do Not Disturb state.

People links

When portal users click the name of (or the icon for) a person or group, a menu appears that provides actions linking them to other portal users. The actions that are visible on people link menus depend on the following factors:

- ▶ Whether Lotus companion products for advanced collaboration are installed and enabled to work with your portal (for example, Lotus Sametime and Lotus Discovery Server).
- ▶ The online status (Active, Away, Do Not Disturb, Offline) of the person or group.
- ▶ Whether the link applies to an individual person or to a public group.

The complete set of actions that users might see from people links depends on which Lotus companion products for advanced collaboration are installed and enabled to work with the portal.

- ▶ See a person's online status
Appearing only if Lotus Sametime is enabled, the Sametime status icon indicates whether a person is active, away, offline, or does not want to be disturbed.
- ▶ Chat
Appearing only if Lotus Sametime is enabled, this action is not available if the person is offline.
- ▶ Send E-mail
- ▶ Show Profile
Appearing only if Lotus Discovery Server is enabled, this action displays the person's profile of business card information, contact information, affinities, current job, and background.
- ▶ Find Documents Authored By
Appearing only if Lotus Discovery Server is enabled, this action launches Knowledge Map Search of all documents that the person has authored.
- ▶ Add to Sametime List
Appearing only if Lotus Sametime is enabled, this action displays a window in which you can add the person to your Sametime List, either as an individual entry or as a member of a new or existing personal group.

Online presence

If you have enabled Lotus Sametime to work with the portal, portal users can see each other's online presence in their person links according to the status options they have set in their Sametime client:

- ▶ I am active.
- ▶ I am away or not using the computer now.
- ▶ Do not disturb me.
- ▶ I am offline.

Tip: Portal users can select Options → Preferences → Status in their Sametime Connect clients to customize the messages that appear for each online state and to control the period of time in which keyboard or mouse inactivity automatically switches their status from Active to Away.

In this section we describe how to use this integration technique to add some of the collaborative services included with WebSphere Portal. Before we begin, we first review what the Collaborative Components are.

Collaborative Components

Collaborative Components or *Collaborative Services* are a set of JSP tags and Java objects that enable the inclusion of Lotus collaborative functionality to new or existing portlets.

These components allow the seamless interaction of the Lotus family of products with the Java-based portlets that run on WebSphere Portal. These methods of interaction do not replace the product-specific APIs, but facilitate the integration of such products within the portal infrastructure.

The Collaborative Components provide standardized access to applications, through an easy to use API that is optimized for a collaborative portal, and a consistent security model across all Lotus Software family of products.

This API is written in Java and isn't platform-specific, so it provides a multiplatform solution, with no dependence on UI-specific implementation. Therefore, these components are very useful tools for writing pervasive portlets.

The services are the following:

- ▶ Infrastructure objects
 - CSEnvironment
 - CSCredentials
 - CSFactory
- ▶ Java Service objects
 - CSBaseService. This service is the base class for the other Java Service objects.

- DominoService
- QuickplaceService
- PeopleService
- DiscoveryServerService
- ▶ JSP custom tags
 - People tags
 - Menu tags

Further information on usage and services is available from the WebSphere Portal Infocenter.

Note: The Lotus Notes productivity portlets that are installed with Portal are not implemented with the DominoService object. Therefore, do not refer to those portlets as examples. Instead, refer to the samples that are installed with the Collaborative Components enterprise application (cs.ear).

The Java Collaboration Components API is explained in the next chapter because it requires further Java knowledge. For now, our discussion concentrates on the People tags components, which work with JSP Tag libraries.

PeopleService tags

The PeopleService tag library contains the following necessary tags:

people

- ▶ Displays person menu links according to the following factors:
 - Portlet context (that is, where the portlet resides)
 - The permission level of the specified user
 - The Lotus Software products for advanced collaboration that are installed and enabled in the portal environment.
- ▶ Added to a person's name string returned by the PeopleService object, the person tag generates the HTML that renders the person link menu that provides the following basic action for collaborating with the named person:
 - Send e-mail
- ▶ If Sametime is installed and enabled to work with the Portal, then the person tag also provides online status for the person link and the following additional actions on the person menu:
 - Chat
 - Share

- Available tools
- Add to Sametime List
- ▶ If Discovery Server is installed and enabled to work with the Portal, then the person tag provides the following additional actions on the person menu:
 - Show expertise profile
 - Find documents authored by

peopleinit

Determines whether Sametime or Discovery Server, or both, are enabled to work with a portal and generates the correct HTML and JavaScript for initializing Sametime or enabling Discovery Server, or both. Establishes the server connections between the Portal, the Sametime server, and the Discovery Server, and provides automatic log-in to all servers.

Both the people and peopleinit tags generate HTML and require Java and JavaScript on the client that are compatible with the WebSphere Portal.

4.7.2 Implementation of the technique

Now that your portlets are Click to Action enabled, the next step is to implement people awareness by including one the PeopleService tags which is available on WebSphere Portal. People awareness services are available at the JSP level and are incredibly easy to use.

The enablement consists basically of two steps on each JSP you want to enable:

- ▶ Insert a JSP taglib directive on the JSP.
- ▶ Place the people tag around the names in the JSP of which you want to be aware.

Enabling the CustomerList portlet

The CustomerList portlet consists only of one JSP file. Follow these steps to enable the portlet:

1. Open the /jsp/CustomerList/View.jsp file.
2. Copy the following line at the beginning of the file:


```
<%@ taglib uri="/WEB-INF/tld/people.tld" prefix="peopleservice" %>
```
3. In the JSP file, in the results file of the display table, insert in the third column, surrounding the <Domino:viewitem col="3"/> tag, the peopleservice tags as follows:

```
<peopleservice:person><Domino:viewitem col="3"/></peopleservice:person>
```

Important: Be careful when inserting the peopleservice tags: *do not* include a line break or space between the tags and the name.

4. Save and close the View.jsp file.

Note: You can ignore the errors displayed in the tasks view since they will find the reference once deployed on the WebSphere Portal.

Enabling the CustomerDetails portlet

The CustomerDetails portlet is composed of two JSPs. The View.jsp file doesn't display any names, so you do not need to enable people awareness on this file. The CustomerDetails.jsp file displays people names, so you need to enable this JSP.

1. Open the /jsp/CustomerDetails/CustomerDetails.jsp file.
2. Copy the following line at the beginning of the file:

```
<%@ taglib uri="/WEB-INF/tld/people.tld" prefix="peopleservice" %>
```

3. In the JSP file on the results table there is a row called Account Owner. Enable this field by including the following peopleservice tags:

```
<peopleservice:person><Domino:item  
name="ownerName"/></peopleservice:person>
```

Important: Be careful when inserting the peopleservice tags: *do not* include a line break or space between the tags and the name.

4. Save and close the CustomerDetails.jsp file.

Note: You can ignore the errors displayed in the tasks view since they will find the reference once deployed on the WebSphere Portal.

Enabling the CustomerContacts portlet

The CustomerContacts portlet is composed of two JSPs. The View.jsp file doesn't display any names, so there is no need to enable people awareness in this file. The CustomerContacts.jsp file displays people names, so you need to enable this JSP using the following steps:

1. Open the /jsp/CustomerContacts/CustomerContacts.jsp file.
2. Copy the following line at the beginning of the file:

```
<%@ taglib uri="/WEB-INF/tld/people.tld" prefix="peopleservice" %>
```

3. In the JSP file on the results table there is a column called Contact Name. Enable this field by including the peopleservice tags as follows:

```
<peopleservice:person><Domino:viewitem col="2"/></peopleservice:person>
```

Important: Be careful when inserting the peopleservice tags: *do not* include a line break or space between the tags and the name.

4. Save and close the CustomerContacts.jsp file.

Note: You can ignore the errors displayed in the tasks view since they will find the reference once deployed on the WebSphere Portal.

Enabling our CustomerSalesActivities portlet

The CustomerSalesActivities portlet is composed of two JSPs. The View.jsp file doesn't display any names, so there is no need to enable people awareness in this file. The CustomerSalesActivities.jsp file displays people names, so you need to enable this JSP using the following steps:

1. Open the /jsp/CustomerSalesActivity/CustomerSalesActivities.jsp file.
2. Copy the following line at the beginning of the file

```
<%@ taglib uri="/WEB-INF/tld/people.tld" prefix="peopleservice" %>
```

3. In the JSP file on the results table there are two columns called Sales Person and Contact Name. Enable these fields by including the peopleservice tags as follows:

```
...  
<peopleservice:person><Domino:viewitem col="4"/></peopleservice:person>  
</td>  
<td>  
<peopleservice:person><Domino:viewitem col="5"/></peopleservice:person>  
...
```

Important: Be careful when inserting the peopleservice tags: *do not* include a line break or space between the tags and the name.

- ▶ Save and close the CustomerSalesActivities.jsp file.

Note: You can ignore the errors displayed in the tasks view since they will find the reference once deployed on the WebSphere Portal.

All of your portlets are now enabled, and they can be deployed in the WebSphere Portal just as you did previously. You should see your portlets working as shown in Figure 4-41.

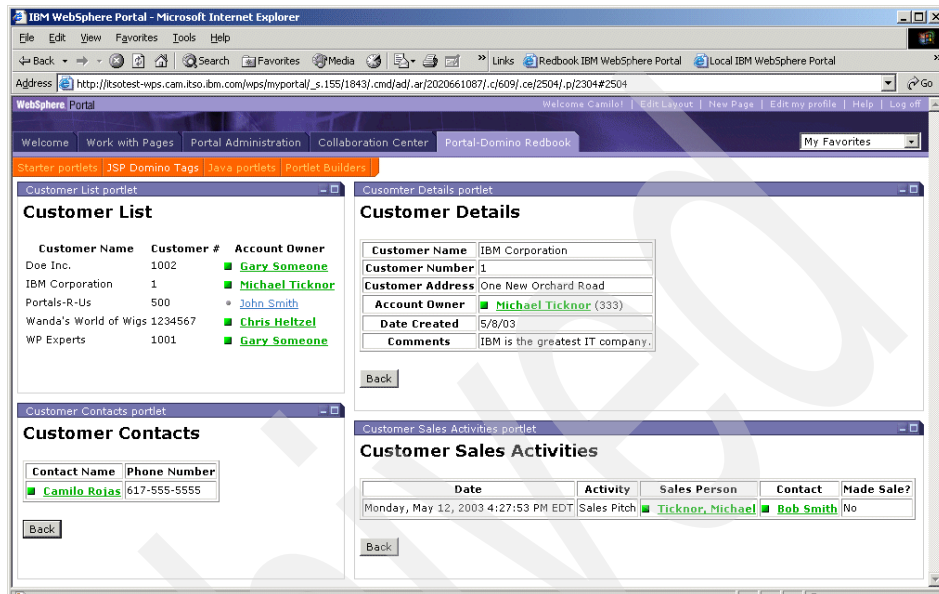


Figure 4-41 Customer Information portlets with people awareness

4.8 Reference Material

The following references are helpful when exploring J2EE:

- ▶ Sun Microsystems J2EE Web site
<http://java.sun.com/j2ee/>
- ▶ Sun Microsystems JSP Web site
<http://java.sun.com/jsp>
- ▶ *Introduction to JavaServer Pages* - developerWorks®
<http://www-105.ibm.com/developerworks/education.nsf/java-onlinecourse-bytitle/882707E838C672A185256770004BDE72?OpenDocument>
- ▶ *Using JSPs and custom tags within VisualAge for Java and WebSphere Studio* - developerWorks
<http://www7b.software.ibm.com/wsdd/library/tutorials/vajwebsph353/Part-I/JS P11Part-I.html&origin=cmp>
- ▶ *Domino and WebSphere Together, Second Edition*, SG24-5955

<http://www.redbooks.ibm.com/redbooks/sg245955>

- ▶ *WebSphere Studio Application Developer Programming Guide*, SG24-6585
<http://www.redbooks.ibm.com/redbooks/sg246585>

For WebSphere Portal information relevant to this chapter, consult the following:

- ▶ WebSphere Portal 4.2.1 Infocenter
<http://publib.boulder.ibm.com/pvc/wp/42/index.html>
- ▶ *Guide to WebSphere Portal*
<ftp://ftp.software.ibm.com/software/websphere/portal/pdf/Guide-to-WebSphere-Portal.pdf>
- ▶ WebSphere Portal Product Documentation
<http://www7b.software.ibm.com/wsd/zones/portal/proddoc.html>
- ▶ WebSphere Portal for Multiplatforms - Library
<http://www-3.ibm.com/software/genservers/portal/library/>
- ▶ *Access Integration Patterns using IBM WebSphere Portal*, SG246267
<http://www.redbooks.ibm.com/redbooks/sg246267>
- ▶ *Portlet Development Guide*
<http://www7b.software.ibm.com/wsd/zones/portal/portlet/portletdevelopmentguide.html>
- ▶ *Portlet Best Practices Guide*
<http://www7b.software.ibm.com/wsd/zones/portal/portlet/portletcodingguidelines.html>

When working with the Lotus Domino JSP tag libraries, the following reference is helpful:

- ▶ Lotus Domino Designer 6 Help
There is a chapter on the JSP Tag libraries that has further explanations to the tags covered on this chapter.

Archived



Portlet development using Java: Technology review

This chapter describes a method of integrating Domino applications using Java as a programming language. When you develop your own portlets in Java, you are able to overcome the limitations that are inherent in the other options described in this redbook.

Domino development and J2EE development generally employ different skill sets. Therefore, when planning this chapter and the next, we considered questions like: What does a Domino developer need to learn to get started with J2EE-based programming, and what unique perspective can the Domino developer bring to the J2EE world? The opposite point of view was also considered: What does a J2EE developer need to know about Domino and how to access data in Domino?

This chapter answers these questions by providing a very detailed review of basic portal concepts, with specific reference to Java technology; as well as an in-depth discussion Java development techniques.

Also in this chapter are some guidelines for portlet development in general, but concentrating on topics pertinent to Domino portlet developers.

5.1 Overview

Use Java to portalize Domino applications when you need fully customizable portlets. Advantages of this approach are that Java offers: a rich class library, multithreading, code reuse, platform-independent applications, integration with 3rd party Java applications, object-oriented language features, advanced architecture, inheritance and design patterns, just to mention a few.

Java is the base programming language, ultimately used by all portlets running on WebSphere Portal. You are not limited by the functionality or the user interactions provided by portlet builders or existing portlets, or by any scalability limitations. When you create your own Java programs, you can deal with any issues that might come up.

You can develop portlets which are very fast and scalable in accessing Domino. To build such effective, scalable portlets, you should be aware of issues like session management, object pooling, how Domino Objects are handled, memory management, and so on. These issues are discussed in this chapter.

The disadvantages of developing in Java are that extensive Java programming skills and portlet programming skills are required, and in many cases the development time can be a lot longer than that of other integration techniques. In addition, a fair understanding of Domino objects structure is required.

5.1.1 Technologies involved

The technologies discussed in this chapter are the following:

- ▶ Portlet
- ▶ Portlet API
- ▶ Domino Java architecture and Domino Java API
- ▶ How to handle Rich Text fields
- ▶ Collaborative Components API
- ▶ Object Pooling
- ▶ Logging
- ▶ Struts framework

In the following chapter, we put this information to use by providing examples of adding functionality to portlets using these techniques.

5.2 Technical introduction to portlets

A portlet is a server-side application that runs in the context of the WebSphere Portal. It inherits from the `javax.servlet.http.HttpServlet` class, and as such, is treated as a servlet by the application server.

The portlet is executed inside a Web container managed by the application server. In the Portlet API, this container is referred to as the Portlet container.

It is not possible to directly execute the portlet functionality by addressing the portlet via HTTP. Though a portlet may provide dual functionality as both a servlet and a portlet, it is certainly a best practice to keep these Controller functions separate. A portlet is visible on a portal page as a single small window; each portal page may have many. The portlet is the content inside the window, not the window itself. The window is defined by the selected skin.

More in-depth information about portlets is in *IBM Websphere Portal V4 Developer's Handbook*, SG24-6897.

5.2.1 Basic portlet terms

In order to fully understand some of the introductory topics, it is necessary to define a few of the most basic terms used when discussing portlets. (Additional information about basic portlet topics is in 1.3.1, "Introduction to portlets" on page 9.)

Portlet window

This is the window that surrounds the portlet, including the title bar and any border images.

State

This is the current state of the portlet window. Valid states are Normal, Minimized, and Maximized.

Mode

This defines the current condition of the portlet. The modes that are available for any particular user depend on the permissions for that user, the device they are using to access the portlet and the configuration and implementation of the portlet. The supported modes are View, Edit, Configure, and Help.

5.2.2 Model-view-controller (MVC) design pattern

To help you understand the role of a portlet and to help prepare you to develop effective and well-designed portlets, a review of the Model View Control (MVC)

architecture is necessary. Several benefits of the portlet architecture are available to you only if you employ a good MVC design.

The Model View Control architecture is concerned with separation of responsibilities. The objective, no matter how it is applied or to what type of application, is to separate a system into tiers. Each tier should be small, identifiable, self-contained, and reusable. These tiers are identified by the role they play in the system. Each role in that system may have several classes working in conjunction to achieve the goal of that role. This section covers the three roles of MVC: Model, View and Control.

Though the MVC architecture was originally applied to Java Swing applications, it has gained popularity and widespread acceptance throughout the J2EE community.

The correct portlet design is broken down into three distinct parts: the model, the view, and the controller (MVC). This design follows classical object-oriented design patterns where each part is self-contained and modular, easing maintenance, extensions, and advancements.

The *model* is the data to which the portlet provides an interface. Common data models are XML documents, database tables, and even other Web applications. The Java classes accessing the data model should have no knowledge of the form that the data is in, the idea being that the model can be changed without affecting the rest of the portlet application.

The *view* is the interface to the data model, presenting it in some usable format. The view accesses the data to be rendered through the model interfaces and thus should not care what format the model takes. It should also not understand the relationships between data models or represent any of the business logic for manipulating the data. Like the data model, the view should be independent and interchangeable, allowing other views to be substituted without affecting the business logic or the model. The typical embodiment of the view is through a series of Java Server Pages (JSPs), but can also be through other rendering techniques such as using XSL stylesheets to format XML documents.

The *controller* is the glue that ties the model to the view and defines the interaction pattern in the portlet. The controller handles user requests from the view and passes control to the appropriate model objects to complete the requested action. The results of the action are then rendered back to the user by the controller using appropriate view objects and perhaps model objects that represent the data results of the completed action.

The controller resides in the portlet Java classes themselves. It knows the data model only through the model interfaces and it knows the view only in that it dispatches the view to render the data. Therefore, the controller logic can be just

as easily replaced as the view and the model. Typical controller implementations utilize intrinsic functions in the portlet API for coordinating action sequences around user input, model data calculations, and result rendering.

As you design your portlets, it is extremely important to hold true to the MVC design principles. Portlets typically evolve over time and are largely reused as the foundation for new portlets. The ability to adapt a portlet to a new back-end data provider, or add markup support for mobile devices, or enhance the portlet to include user personalization, requires that each part of the portlet be self-contained and extensible.

5.2.3 Portlet API overview

The WebSphere Portal is based on a portlet container that provides a runtime environment in which portlets are instantiated, used, and finally destroyed. Portlets rely on the portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, lookup credentials, and to store persistent data. The Portlet API provides standard interfaces for these functions. The portlet container is not a stand-alone container like the servlet container. Instead, it is implemented as a thin layer on top of the servlet container and reuses the functionality provided by the servlet container.

IBM is working with other companies to standardize the Portlet API, making portlets interoperable between portal servers that implement the specification. The Portlet API offered in WebSphere Portal Version 4.2 is the first step toward the Portlet API standardization. For more information about the portlet specification, see:

<http://jcp.org/jsr/detail/168.jsp>

5.2.4 Portlets and the Servlet API

The abstract Portlet class is the central abstraction of the Portlet API. The Portlet class extends `HTTPServlet`, of the Servlet API. All portlets extend this abstract class indirectly, and inherit from `HttpServlet`, as shown in Figure 5-1.



Figure 5-1 Portlet object inheritance structure

Therefore, portlets are a special type of servlet, with properties that allow them to easily plug in to and run in the portal server. Unlike servlets, portlets cannot send redirects or errors to browsers directly, forward requests, or write arbitrary markup to the output stream. The portlet container relies on the J2EE architecture implemented by WebSphere Application Server. As a result, portlets are packaged similar to J2EE Web applications and are deployed like servlets.

Generally, portlets are administered more dynamically than servlets. For example, the following changes can be applied without having to start and restart the portal server:

- ▶ Portlet applications consisting of several portlets can be installed and removed using the portal administration user interface.
- ▶ The settings of a portlet can be changed by an administrator with appropriate access rights.
- ▶ Portlets can be created and deleted dynamically by administration portlets. For example, the clipping portlet can be used to create new portlet instances whenever an administrator creates a new clipping.

The portlet container relies on the J2EE architecture implemented by WebSphere Application Server. As a result, portlets are packaged in WAR files similar to J2EE Web applications and are deployed like servlets. Like other servlets, a portlet is defined to the application server using the servlet deployment descriptor (web.xml). This file defines the portlet's class file and read-only initialization parameters.

The initialization parameters are set by the portlet developer and can be read by the portlet using the PortletConfig object. The servlet deployment descriptor can contain multiple Web applications, each defined by the <servlet> element. In addition, each servlet definition can point to the same portlet class file, thus creating different PortletConfig objects with different initialization parameters for each portlet class instance.

5.2.5 Portlet concepts

The following figure shows different variations of a portlet as it is created, placed on a page, and accessed by users. Notice that the first two steps involve the use of persistent data, but for the third step, the data is available only for the duration of the session.

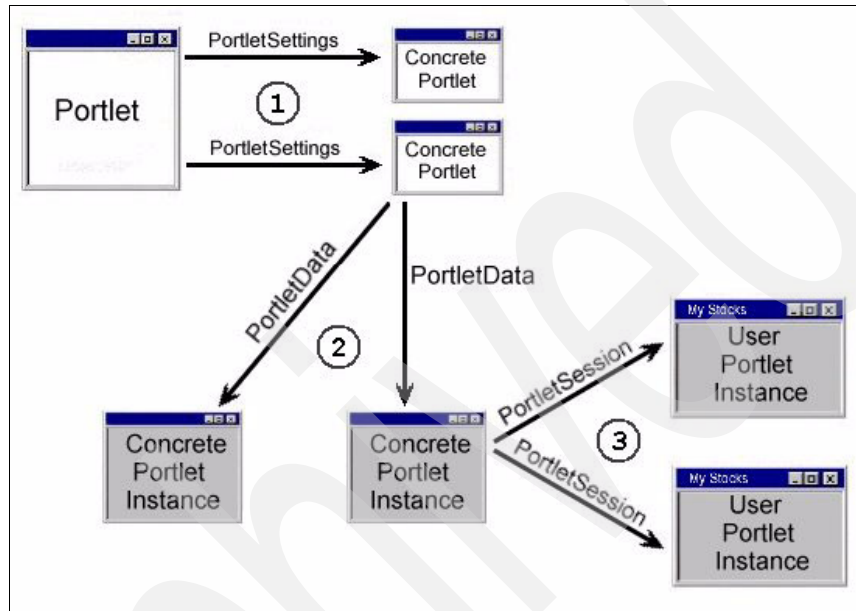


Figure 5-2 Portlet life cycle

1. The portal administrator uses the administrative interface to deploy a new portlet application WAR file or install a copy of a portlet. Either action creates a concrete portlet, which is a portlet parameterized by a single `PortletSettings` object. There can be many concrete portlets for each portlet. `PortletSettings` are read/write accessible and persistent. The `PortletSettings` contain configuration parameters initially defined in the portlet deployment descriptor. The use of concrete portlets allows many instances of a portlet to run with different configurations, without creating extra portlet class instances. During the lifecycle of a single portlet, many concrete portlets can be created and destroyed. There is no object that explicitly represents the concrete portlet. The same concrete portlet can be shared across many users.
2. The portlet is placed on a page by a user or an administrator. This creates a concrete portlet instance, which is a concrete portlet parameterized by a single `PortletData` object. There can be many concrete portlet instances per concrete portlet. `PortletData` stores persistent information for a portlet that

has been added to a page. For example, a user can edit a stock quotes portlet and save a list of stock symbols for the companies to track.

3. The scope of the PortletData depends on the scope of the page that the concrete portlet is on.
 - a. If an administrator puts a concrete portlet on a group page, then the PortletData object contains data stored for the group of users. This holds true for a group of users who have view access to the page. However, if users have edit access to the portlet on a group page, then a new concrete portlet instance is created for each user that edits the portlet. In this case, PortletData contains data for each user that edits the portlet.
 - b. If a concrete portlet is put on a user's page, the PortletData contains data for that user.

When a user accesses a page that contains a portlet, that creates a user portlet instance. When a user logs into the portal, the portal server creates a PortletSession for each of the user's portlets. A concrete portlet instance parameterized by a PortletSession is known as a user portlet instance. There can be many user portlet instances per concrete portlet instance.

A user portlet instance is a concrete portlet instance parameterized by a single PortletSession. There can be many user portlet instances per concrete portlet instance. The PortletSession stores transient information related to a single use of the portlet.

5.2.6 Portlet applications

Portlet applications provide the means to package a group of related portlets that share the same context. The context contains all resources, for example, images, properties files, and classes. All portlets must be packaged as part of a portlet application.

Concrete portlet application

A concrete portlet application is a portlet application parameterized with a single PortletApplicationSettings object. For each portlet application, there may be many concrete portlet applications. PortletApplicationSettings are read/write accessible and persistent. There is no object that explicitly represents the concrete portlet application.

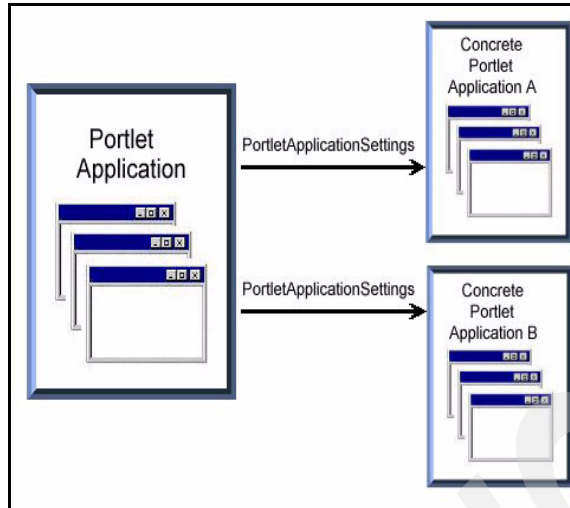


Figure 5-3 Concrete portlet applications

A concrete portlet application contains at least one concrete portlet from the portlet application, but it is not required to contain all of them.

Portlet applications provide no code on their own but form a logical group of portlets. Beside this more logical gain, portlets of the same portlet application can also exchange messages.

5.2.7 Basic elements of the Portlet API

This section describes the basic elements of the Portlet API. Figure 5-4 shows a map of many of the common objects in the Portlet API.

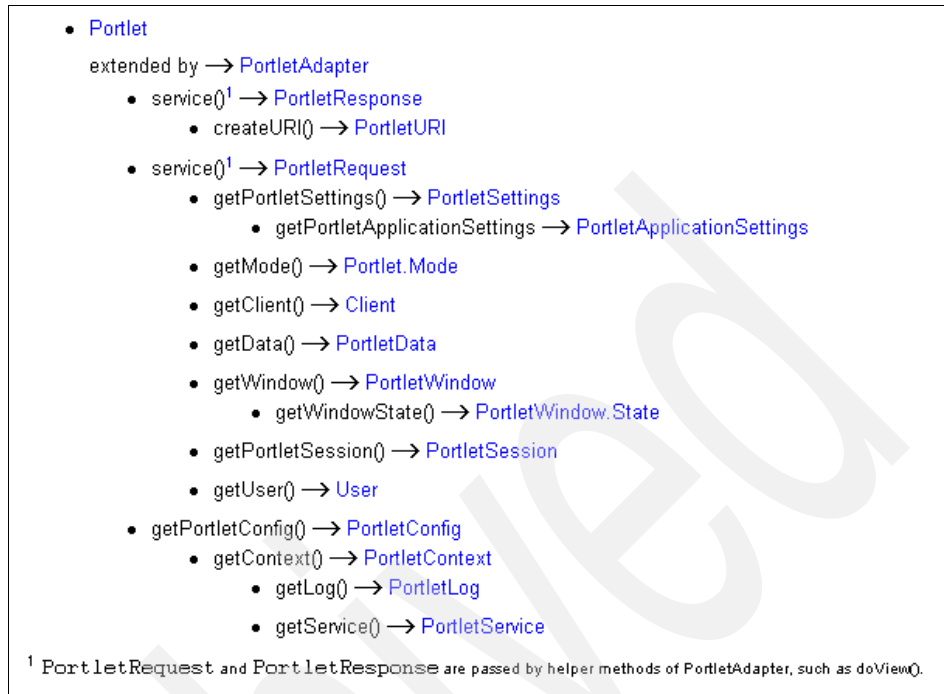


Figure 5-4 Portlet API common objects

Portlet class

The abstract **Portlet** class is the central abstraction of the Portlet API. All portlets extend this abstract class by extending one of its subclasses, such as **PortletAdapter**, that provide methods helpful for the developer.

Portlet life cycle

The portlet container calls the following methods of the abstract portlet during the portlet's life cycle:

▶ **init()**

The portlet is constructed, after portal initialization, and then initialized with the **init()** method. The portal always instantiates only a single instance of the portlet, and this instance is shared among all users, exactly the same way a servlet is shared among all users of an application server.

▶ **initConcrete()**

After constructing the portlet and before the portlet is accessed for the first time, the concrete portlet is initialized with the **PortletSettings**.

- ▶ `service()`
The portal calls the `service()` method when the portlet is required to render its content. During the life cycle of the portlet, the `service()` method is typically called many times. For each portlet on the page, the `service()` method is not called in a guaranteed order and may even be called in a different order for each request.
- ▶ `destroyConcrete()`
The concrete portlet is taken out of service with the `destroyConcrete()` method. This can happen when an administrator deletes a concrete portlet during runtime on the portal server.
- ▶ `destroy()`
When the portal is terminating, portlets are taken out of service, then destroyed with the `destroy()` method. Finally, the portlet is garbage collected and finalized.

Wrapper classes

Portlets do not extend the abstract `Portlet` class directly, but rather extend `PortletAdapter` or any other helper class that in turn extends `Portlet`. Extending one of these classes helps protect your portlet from changes in the abstract `Portlet` class. Moreover, it saves you the work of having to implement all of the methods of the `Portlet` interface, even if your portlet does not need to use them all. Using the `PortletAdapter` class, you only have to overwrite the methods you really need.

In its `service()` method, the `PortletAdapter` class invokes methods corresponding to the portlet mode. Portlets that extend this class can overwrite the `doView()`, `doEdit()`, and `doHelp()` methods without having to test the mode or write a specific `service()` method.

5.2.8 Frequently used objects

The following objects are most often used by a portlet:

- ▶ `PortletRequest`
- ▶ `PortletResponse`
- ▶ `PortletSession`

Each of these objects is an extension of its counterpart in the Servlet API.

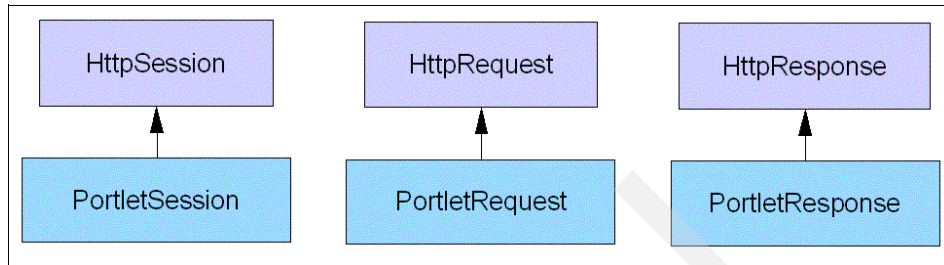


Figure 5-5 Portlet objects and their counterparts in the Servlet API

PortletRequest

The PortletRequest object is passed to the portlet through the login(), beginPage(), endPage(), and service() methods, providing the portlet with request-specific data and the opportunity to access further important information as listed below.

Attributes

Attributes are name/value pairs associated with a request. Attributes are available only for the scope of the request. The portlet can get, set, and remove attributes during one request. In the following example we are adding, getting, and removing an attribute.

```

portletRequest.setAttribute("key", objectToStore);
portletRequest.getAttribute("key");
portletRequest.removeAttribute("key");
  
```

Parameters

Parameters are name/value pairs sent by the client in the URI query string as part of a request. Often the parameters are posted from a form. Parameters are available for the scope of a specific request. The portlet can get but not set parameters from a request. Notice that it differs from the attribute, since you can only get the parameters that are sent from the client, and the attributes can be set and modified at the Java code, so use the attributes when you are trying to send special attributes to the redirected JSP page. The extraction of a parameter would be similar to the following:

```

portletRequest.getParameter("nameOfTheHTMLfield");
  
```

Client

The Client object encapsulates request-dependent information about the user agent of a specific request. Information from the Client includes the manufacturer of the user agent or the type of markup that the client supports. The Client is

extracted from the `PortletRequest` using the `getClient()` method. The following information can be obtained from the Client:

▶ User agent

The portlet can get the String sent by the user agent to identify itself to the portal. The following code will return the user agent information:

```
portletRequest.getClient().getUserAgent();
```

▶ Markup name

The portlet can get the String that indicates the markup language that the client supports, for example, "wml".

```
portletRequest.getClient().getMarkupName();
```

▶ MIME type

The portlet can get the String that indicates the MIME types supported by the client (for example, `text/vnd.wap.wml`). If the portlet supports multiple types of devices, it should get the markup name rather than the MIME type.

```
portletRequest.getClient().getMimeType();
```

▶ Capabilities

The `Capability` object contains more detailed information than the markup type about what the client can support, such as the level of HTML, JavaScript, or WML tables.

```
portletRequest.getClient().isCapableOf(Capability);
```

User data

The `PortletData` object represents data for a concrete portlet instance that is saved to persistent store. For example, a user can set a portlet e-mail application to check for new mail every 30 minutes. This preference is stored for the instance in the `PortletData` object. This is an excellent tool for storing user-portlet information without having to mind the persistence infrastructure.

```
portletRequest.getData().setAttribute("key", serializedObject);  
portletRequest.getData().store();  
portletRequest.getData().removeAttribute("key");  
portletRequest.getData().removeAllAttributes();
```

Session

The `PortletSession` represents user-specific, transient data for more than one request. In contrast with the request, which does not retain data after the request is completely processed, session attributes can be remembered and saved over more than one request.

```
portletRequest.getPortletSession().setAttribute("key", object);  
portletRequest.getPortletSession().getAttribute("key");
```

```
portletRequest.getPortletSession().removeAttribute("key");
```

Portlet settings

The PortletSettings object represents the configuration for a concrete portlet that is saved to persistent store. For example, an administrator can set to which host and port a Stock portlet should connect to get live data. This preference is stored for the concrete portlet in the PortletSettings object.

```
portletRequest.getPortletSettings().setAttribute("key",object);
portletRequest.getPortletSettings().store();
portletRequest.getPortletSettings().getAttribute("key");
portletRequest.getPortletSettings().removeAttribute("key");
```

Mode

Portlet.Mode provides the current or previous mode of the portlet.

```
portletRequest.getMode();
//Will return a Portlet.Mode static value
```

PortletWindow

The PortletWindow object represents the state of the current portlet window. The portlet can access this object to determine if the portlet is currently maximized, minimized, or rendered in its normal view.

ModeModifier

This object can be used in a portlet action to set the portlet mode to its current, previous, or requested mode before the portlet is rendered. For example, a portlet in edit mode could process a user action and return the portlet to edit mode for more input before returning to view mode.

```
portletRequest.setModeModifier(Portlet.ModeModifier);
```

PortletResponse

The response object encapsulates information to be returned from the server to the client. PortletResponse is passed via the beginPage(), endPage(), and service() methods and can be used by the portlet to return portlet output using a Java PrintWriter. The response also includes methods for creating the PortletURI object or qualifying portlet markup with the portlet's namespace.

Use one of the following methods to create the PortletURI:

createURI()

Creates a PortletURI object pointing to the calling portlet with the current mode.

```
portletResponse.createURI();
```


createURI(PortletWindow.State state)

Creates a PortletURI object pointing to the calling portlet with the current mode and given portlet window state.

```
portletResponse.createURI(PortletWindow.State);
```

createReturnURI()

Creates a portletURI pointing at the caller of the portlet. For example, createReturnURI() can be used to create a back button in an edit mode.

Each portlet runs in its own unique namespace. The encodeNamespace() method is used by portlets to bring attributes in the portlet output to avoid name clashes with other portlets. Attributes can include parameter names, global variables, or JavaScript function names.

```
portletResponse.createReturnURI();
```

PortletSession

The PortletSession holds user-specific data for the concrete portlet instance of the portlet, creating a portlet user instance. Concrete portlet instances differ from each other only by the data stored in their PortletData. Portlet user instances differ from each other only by the transient data stored in their PortletSession. Any persistent data must be stored using PortletData. Information stored in a portlet's instance variables is shared between all concrete portlet instances and even between all concrete portlets—with read and write access. Make sure you do not use instance attributes for user-specific data.

On the other hand, you have to be cautious about what the portlet adds to the session, especially if the portlet ever runs in a cluster environment where the session is being serialized to a shared database. Everything being stored in the session must be serializable, too.

Like the HttpSession, a PortletSession is not available on an anonymous page. However, in cases where an administrator places a portlet on an unauthenticated page, such as the Welcome page shipped with WebSphere Portal, the portlet should provide code to handle it.

During login, a PortletSession is automatically created for each portlet on a page. To get a PortletSession, the getSession() method (available from the PortletRequest) has to be used. The method returns the current session or, if there is no current session and the given parameter "create" is true, it creates one and returns it. For an example inspect the Portlet Request object.

5.2.9 Configuration objects

The following objects are used by the portlet to retrieve and store data, depending on how the data is used:

- ▶ PortletConfig
- ▶ PortletSettings
- ▶ PortletApplicationSettings
- ▶ PortletData

PortletConfig

PortletConfig provides the non-concrete portlet with its initial configuration. The configuration holds information about the portlet class. This information is valid for every concrete portlet derived from the portlet.

A portlet's configuration is initially read from its servlet deployment descriptor. This information is set by the portlet developer. The configuration is read-only and cannot be changed by the portlet.

The PortletConfig is passed to the portlet in the `init()` method of the abstract Portlet and is used to access portlet-specific configuration parameters using `getInitParameters()`. PortletConfig parameters are name/value pairs available for the complete life cycle of the non-concrete portlet. Non-concrete portlet parameters are defined by the `<init-param>` tag in the servlet deployment descriptor.

```
getPortletConfig().getInitParameter("parameterName");  
getPortletConfig().getContext().send("string",PortletMessage);  
getPortletConfig().supports(Portlet.Mode);  
getPortletConfig().supports(PortletWindow.State);
```

PortletSettings

The PortletSettings object provides the concrete portlet with its dynamic configuration. The configuration holds information about the concrete portlet. This information is valid for every concrete portlet instance of the concrete portlet.

A concrete portlet's configuration is initially read from the portlet deployment descriptor. The configuration is read-only and can be written by the portlet only when the portlet is in configure mode. This information is normally maintained by the portal administrator and may be changed while the portal server is running. The portlet can get, set, and remove attributes during one request. To commit the changes, the `store()` method has to be called.

The `PortletSettings` object can be accessed with the `getPortletSettings()` method, available from the `PortletRequest`. Often, it is used to access portlet-specific configuration parameters using `getAttribute()`. Attributes are name/value pairs available for the complete life cycle of a concrete portlet. Concrete portlet attributes are defined by the `<config-param>` tag in the portlet deployment descriptor.

```
portletRequest.getPortletSettings().setAttribute("key",object);
portletRequest.getPortletSettings().store();
portletRequest.getPortletSettings().getAttribute("key");
portletRequest.getPortletSettings().removeAttribute("key");
```

PortletApplicationSettings

The `PortletApplicationSettings` object provides the concrete portlet application with its dynamic configuration. The configuration holds information about the portlet application that is shared across all concrete portlets included in the application.

A concrete portlet application's configuration is initially read from the portlet deployment descriptor. The configuration is read-only and can be written by the portlet only when the portlet is in configure mode. This information is normally maintained by the portal administrator and may be changed while the portal server is running. A portlet in the application can get, set, and remove attributes during one request. To commit the changes, the `store()` method has to be called.

The `PortletApplicationSettings` can be accessed with the `getApplicationSettings()` method, available from the `PortletSettings` object. It is used to access portlet-specific configuration parameters using `getAttribute()`. Attributes are name/value pairs available for the complete life cycle of a concrete portlet application. Concrete portlet application attributes are defined by the `<context-param>` tag in the portlet deployment descriptor.

```
portletRequest.getPortletSettings().getApplicationSettings().setAttribute("key",object);
portletRequest.getPortletSettings().getApplicationSettings().store();
portletRequest.getPortletSettings().getApplicationSettings().getAttribute("key");
portletRequest.getPortletSettings().getApplicationSettings().removeAttribute("key");
```

PortletData

`PortletData` holds data for the concrete portlet instance. For each occurrence on a page there is a concrete portlet instance. The `PortletData` contains persistent information about the concrete portlet instance, while the `PortletSession` contains only the transient data of the user portlet instance.

There is one concrete portlet instance for each occurrence of a portlet on a page. A page can be owned by either a single user (personal page) or by a single group of users (group page). PortletData contains user-specific data on a personal page and group-specific data on a group page.

The PortletData object stores attributes as name/value pairs. The portlet can get, set, and remove attributes during one request. To commit the changes, the store() method has to be called. The data is read-only and can be written by the portlet only when the portlet is in edit mode.

```
portletRequest.getData().setAttribute("key", serializedObject);
portletRequest.getData().store();
portletRequest.getData().removeAttribute("key");

portletRequest.getData().removeAllAttributes();
```

5.2.10 Miscellaneous objects

The following miscellaneous objects are used by portlets:

- ▶ PortletContext
- ▶ PortletWindow
- ▶ User

PortletContext

The PortletContext interface defines a portlet's view of the portlet container within which each portlet is running. PortletContext also allows a portlet to access resources available to it. For example, using the context, a portlet can access the portlet log, access context parameters common to all portlets within the portlet application, obtain URL references to resources, or access portlet services.

The most important information related to the PortletContext is described in the following paragraphs.

InitParameters

Parameters are name/value pairs available to all portlets within the Web application. These are defined in the Web deployment descriptor under the <context-param> element. For example, if a group of portlets share a context parameter called "Webmaster" that contains the portal site's administrator e-mail, each portlet could get that value and provide a "mailto" link in their help.

Attributes

Attributes are name/value pairs available to all portlets within the Web application. The portlet can get, set, and remove attributes. Attributes of the context are stored on a single machine and are not distributed in a cluster.

```
getPortletConfig().getContext().setAttribute("key", object);
getPortletConfig().getContext().getAttribute("key");
getPortletConfig().getContext().removeAttribute("key");
```

Localized text

The `getText()` method is used by the portlet to access resource bundles within a given locale.

```
getPortletConfig().getContext().getText("bundle", "key", "locale");
```

Resources

It is through the `PortletContext` that a portlet can load or include resources located in the portlet's application scope. Available methods are `include()` and `getResourceAsStream()`. The `include()` method is typically used to invoke JSPs for output.

```
getPortletConfig().getContext().include("JSP", request, response);
getPortletConfig().getContext().getResourceAsStream("path");
```

Messaging

Through messaging it is possible to communicate between portlets and share data or send notifications. A message is sent by using the `send()` method. For more information, see [Portlet messaging](#).

```
getPortletConfig().getContext().send("portletName", PortletMessage);
```

Portlet services

The `PortletService` object allows portlets to use pluggable services via dynamic discovery.

```
getPortletConfig().getContext().getService(service);
```

PortletWindow

The `PortletWindow` object represents the window that encloses a portlet. For example, on an HTML page, the portlet window can typically be rendered as a table cell. The portlet window can send events on manipulation of its various window controls, like when the user clicks minimize or close. The portlet, in turn, can interrogate the window about its current state. For example, a portlet may render its content differently depending on whether its window is maximized or not. The `PortletWindow` is available using the `getWindow()` method of the `PortletRequest` object.

```
PortletWindow.getWindowState();
PortletWindow.setWindowState(PortletWindow.State);
```

User

The User class represents the users of the portal. The User class contains methods for accessing attributes that make up the user profile, such as the user's full name or the username. The User class abstracts the underlying physical implementation of the one or more data stores which actually hold the user information.

In WebSphere Portal, the User class is part of the `com.ibm.wps.puma` package. This class and several others represent the portal server's API to the user subsystem in Member Services. To obtain the Javadoc for this API, locate the document titled "Javadoc for the WebSphere Portal User and Group objects" on the software support site:

<http://www.ibm.com/software/support>

The classes in the `com.ibm.wps.puma` package provide the only API available to access attributes of the user. In subsequent releases of WebSphere Portal, a new API will become available to access the new features of the user subsystem. In that time frame, the `com.ibm.wps.puma` package will be maintained for a time for backward compatibility, but will no longer be enhanced to accommodate new function.

Note: The `getUser()` method is located at the `PortletRequest` and `PortletSession`. The `portletsession.getUser()` method is deprecated and, in subsequent releases, will only return null.

```
portletRequest.getUser().getFamilyName();
portletRequest.getUser().getFullName();
portletRequest.getUser().getGivenName();
portletRequest.getUser().getID();
portletRequest.getUser().getLastLoginTime();
portletRequest.getUser().getNickName();
portletRequest.getUser().getUserID();
```

5.2.11 Portlet events

Portlet events contain information about an event to which a portlet might need to respond. For example, when a user clicks a link or button, this generates an action event. To receive notification of the event, the portlet must have an event listener implemented within the portlet class.

- ▶ Action events: Generated when an HTTP request is received by the portlet container that is associated with an action, such as when the user clicks a link.
- ▶ Message events: Generated when another portlet within the portlet application sends a message.

- ▶ Window events: Generated when the user changes the state of the portlet window.

A portlet has a different processing and rendering sequence than a servlet. A servlet does all of its processing in the `service()` method. A portlet, on the other hand, uses a two-phase processing that is split between an action processing and service. This split is necessary to accommodate communication between portlets before rendering output in the service stage. The action processing is guaranteed to complete before a portlet is called to render.

During action processing, the portlet implements an `ActionListener` interface. The `ActionListener` interface provides the `actionPerformed()` method, to which an `ActionEvent` object is passed. When a user clicks on a link or a submit button, an `ActionEvent` can be generated. The portlet action can be obtained from the `ActionEvent`, which describes the triggering event. When the `actionPerformed()` method is invoked, a response object is not available because this is not a rendering step. All state changes should be handled during action processing.

Portlets should use the service phase only to render portlet output. The `service()` method is not only called following the `actionPerformed()` processing when a user clicks on a link or button in a portlet, but is also called when the portal page is refreshed. Thus, given a page with two portlets, A and B, when the user clicks on a link in portlet A, `actionPerformed()` and `doView()` is called for portlet A, but only the `doView()` method is called for portlet B. Once the content generation phase has started, no further events will be delivered. For example, messages cannot be sent from within the `beginPage()`, `service()`, and `endPage()` methods. The resulting message event would not be delivered and is essentially discarded.

The event listener is implemented directly in the portlet class. The listener can access the `PortletRequest` from the event and respond using the `PortletRequest` or `PortletSession` attributes.

Action events

An `ActionEvent` is sent to the portlet when an HTTP request is received that is associated with a portlet action. To receive action events, the portlet class must implement the `ActionListener` interface and a portlet action. A portlet action can be one of the following types:

- ▶ Simple portlet action String
- ▶ `PortletAction` object

These actions are explained in the following paragraphs.

Simple portlet action String

Actions created as simple actions can be executed multiple times, enabling a user's back button to work. Links created with simple portlet actions are represented in the URL rather than in the session. Therefore, portlets with simple actions can be placed on an anonymous page where no session exists. Simple portlet actions are associated with action events using the `getActionString()` method.

```
PortletURI.addAction(String simpleAction);  
String ActionEvent.getActionString();
```

Simple portlet actions are not available in the Portlet API prior to WebSphere Portal Version 4.2. A portlet can determine if the portal server it is running on supports simple actions or not by checking the Portlet API version. The version of the Portlet API on servers that support simple actions has changed from 1.1 to 1.2. Here is example code which illustrates how to check for simple action support:

```
if ( (portletContext.getMajorVersion() <= 1 ) &&  
    (portletContext.getMinorVersion() <= 1 ) )  
    {  
        // cannot use simple actions  
    } else {  
        // simple action support is present on this server  
    }  
}
```

PortletAction object

The `PortletAction` object has been deprecated in favor of simple portlet action strings. It is maintained in the Portlet API to support existing portlets that use `PortletActions`.

Window events

A `WindowEvent` is sent by the portlet container whenever a user clicks on one of the control buttons that change the window's state, such as maximize, minimize or restore. A `WindowEvent` can be used, for example, to display more information when the user maximizes the portlet than would be shown in its normal state. To receive window events, the `WindowListener` interface must be implemented at the portlet class.

The Portlet API provides a `WindowAdapter` class that implements empty methods of the `WindowListener`. By extending `WindowAdapter`, the portlet developer needs to implement only those callback methods needed by the portlet. Without the `WindowAdapter`, you must implement all callback methods, even if the method is empty.

Message events

Message events can be sent from one portlet to others if the recipient portlets are members of the same portlet application and are placed on the same page as the sending portlet. Additionally, a `DefaultPortletMessage` can cross portlet application boundaries and may be sent to all portlets on a page. A `MessageEvent` can be sent actively to other portlets only when the portlet is in the event processing cycle of the portlet container, otherwise an exception is thrown. There are two different types of messages:

- ▶ Single addressed messages: Messages sent to a specific portlet by specifying the portlet name on the `send()` method
- ▶ Broadcast messages: Messages sent to all portlets on the page

Message events are useful when changes in one portlet should be reflected in another one. An object with the type `PortletMessage` has to be implemented and is passed via the `MessageEvent`. The portlet receiving the message must implement the `MessageListener` interface and an object with the type `PortletMessage`.

5.3 Accessing Domino data from portlets using Java and CORBA

You can access data residing in a Domino server from a portlet using Java language, where CORBA serves as a middleware distributed environment for a Domino Object Model (DOM) back-end classes. In practice, you will implement this through the set of Java interfaces contained in the `lotus.domino` package.

Overview

Figure 5-6 on page 248 illustrates the components of portlet integration to Domino.

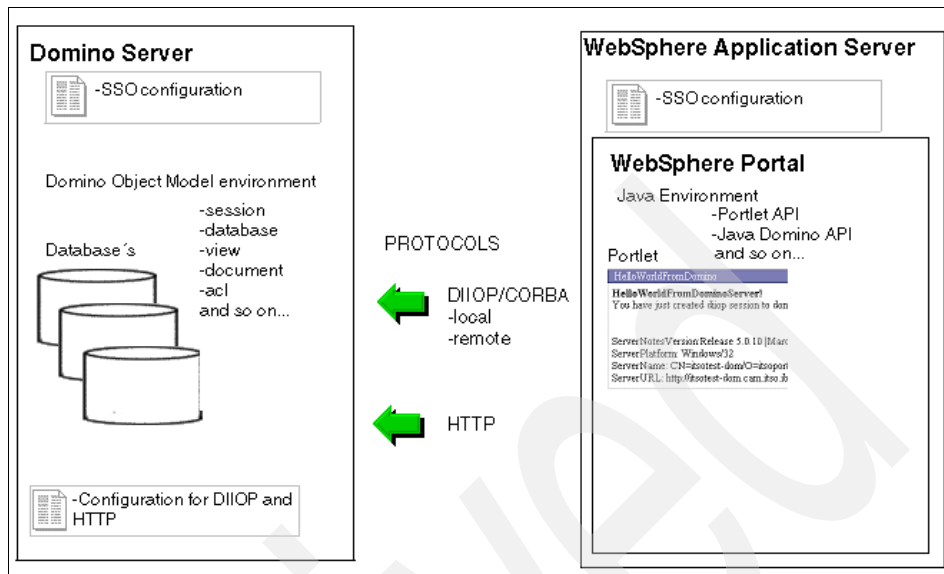


Figure 5-6 Overview of integration from portlet to Domino data

- ▶ You have to enable single sign-on (SSO) between the Domino server and the Websphere Application server. Details about implementing SSO are in the Lotus Domino Administrator 6 Help database.
- ▶ The Domino environment consists of Domino Object Model, such as session, database, views, documents. Protocols (DIIOP and HTTP) are used in implementing physical connection between Domino and the Portlet. You need to configure your Domino server to use DIIOP.
- ▶ In the Portal, you have a Java environment to develop portlets using Portlet API, and Domino Java API for accessing Domino Objects and rendering Domino data to the portlet.

We describe these components in more detail in this section to make this integration technique clear.

Important: The set of Java interfaces contained in the lotus.domino package will implement integration from the portlet to Domino.

Java

Java is one of the most important and commonly used programming languages, and as we have already mentioned, portlets are built with Java language.

Domino offers you the option to write your applications in Java. Domino 6.0 and later supports Java programs written in JDK 1.3 and SDK 2.0. (These tools are discussed in more detail in 6.1, “Software and tools used” on page 306.

CORBA

CORBA, or Common Object Request Broker Architecture, is an open standard defined by the Object Management Group (OMG). CORBA serves as middleware for a distributed computing environment whereby clients can invoke methods on remote APIs residing on other computers. CORBA uses Internet Inter-ORB Protocol (IIOP) for communication over a TCP/IP network.

CORBA/IIOP support enables developers to create portlets that can be remotely invoked in Domino services. In addition, it enables information to be processed efficiently over networks within an open standards-based framework and to distribute work effectively between clients and servers, ultimately lowering the cost of ownership.

Advantages to using CORBA are:

- ▶ You can use Domino Object Model (DOM) back-end classes.
- ▶ The client does not have to deal with issues such as networking or security.
- ▶ CORBA allows many different portlets to use the same objects (not copies of the objects). The latest version of the object is always used.

DIIOB

DIIOB, or Domino Internet Inter-ORB, is the Domino task that allows external CORBA programs to attach to, and manipulate Domino databases. Most notably this allows Java programs, or portlets to connect to Domino. The remote methods use CORBA classes included in the file NCSO.jar/NCSOW.jar shipped with Domino and require that the DIIOB task is running on the server where the session object is to be obtained.

Attention: Portlets can use DIIOB to access objects located in Domino.

Local or remote access to Domino

Domino supports both local and remote access using virtually identical object models. Both options are available through the identical set of Java interfaces contained in the lotus.domino package. The only difference is the .jar file you put in your portlet development environment: notes.jar for local access, or ncsso.jar for remote access. Ncsso.jar is for all Java environments except WebSphere; there you should use ncsow.jar. All of these files are shipped with Domino server.

The coding is the same whether your objects are on the same machine (local) as your Portlet Java code (or JSP), or on some other machine on the network (remote). This is a big win in terms of development simplicity.

Keep in mind the following issues related to thread management:

- ▶ When using the local Domino classes from any Java program, you're essentially calling through a thin layer of Java code into the Domino back-end code, which is implemented in C++. The Java wrapper classes use a standard Java-to-C calling mechanism, known as Java Native Interface (JNI), to access the real Domino classes in the product's .dlls (or whatever the platform-specific equivalent of a .dll is). The Domino code is loaded "in process" in the Java Virtual Machine (JVM), which is great from the point of view of performance: You're getting the best possible speed out of the hook-up between the Java and C code—everything is right there in the computer's memory. On local use you need to manage threads, and initialize and terminate them properly.
- ▶ If you use the remote object library for Domino (the CORBA classes), you aren't accessing any Domino C/C++ code from within the JVM's process space, and there aren't any special requirements for thread initialization or termination. You can instantiate a Session object (or any object in the Domino hierarchy) and keep it around for re-use later. This is a real advantage, although there is a performance hit by having all calls remote across a network.

Even if you're using the CORBA classes to communicate with a Domino server on the same machine as your WebSphere program, you still pay for having all method invocations processed remotely. Extra time is used for: marshalling the arguments, formatting data according to the IIOP wire protocol specification, transmitting the call, de-marshalling the parameters, finding the remote object to invoke, and so on. The bytes may not actually go out over the network, but they have to travel through your network adapter card and be processed on both ends of the conversation as if there were a network cable in between.

In a real-life production portal environment you probably will never face an actual case where you have installed WebSphere Portal server together with Domino Server on the same machine. However, this may be the case in a very small production environment or in a testing and development environment. From a performance point of view it's not ever recommend to install them on the same machine.

To summarize:

- ▶ Local access has much faster access to Domino calls, but does things more repetitively.

- ▶ Remote access is a “cleaner” architecture, requiring fewer actual calls.

Mechanism of the CORBA sessions for Domino

When CORBA is used to access the Domino server, the objects that are instantiated in your portlet code do not contain any functional methods in the same way as local objects do.

The portlet is the client, and the CORBA objects are the Session, Database, View, and Documents that the portlet creates at various times.

When CORBA objects are created, what you actually get is a *stub* to manipulate on the client. A second object—the real one if you like—is created on the server and linked with your stub. This stub contains all the same methods that the real object contains, but instead of the same functional code, they simply contain code to serialize the method call over the network so that the actual method is executed on the server by the *real* object.

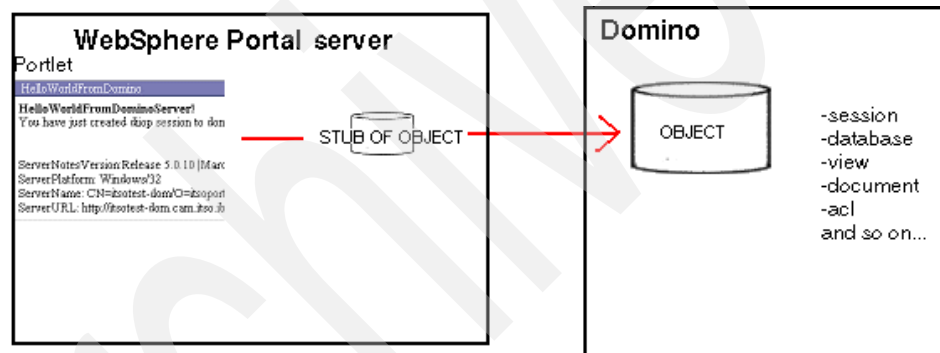


Figure 5-7 Mechanism of CORBA, stub of object versus real object

For example, the Session instance hangs around until you recycle it, or until the connection to Domino times out. All Domino objects created using this Session likewise resides on the remote Domino machine. Any method invocations you make are sent over the network to the server and executed there. What look like local Java object instances are really just instances of proxy objects, whose only job is to communicate with the real objects over on the Domino server. Recycling is a really important issue when you are using Domino Java API. For more information, see “Recycling Domino objects” on page 257.

Enabling Domino server for DIIOP connection

DIIOP can be loaded manually at the server console by typing `load diiop`, or it can be started automatically by including it in the line in `notes.ini` which starts with `ServerTasks=`. It can also be started through the Domino Administrator client on the Server tab.

DIIOp needs some slight configuration changes in the server document to ensure that it can bind to the port specified (63148 by default) on the local host machine. The DIIOp settings are shown in Figure 5-8.

The fully qualified host name of the server in the server document must also be resolvable to the machine's IP address. You may need to add a line in the hosts file of the server's operating system (/etc/hosts on most UNIX/Linux machines, or c:\winnt\system32\etc\drivers\hosts on Windows).

Remote Java/Domino IIOp	
TCP/IP port number:	63148
TCP/IP port status:	Enabled
Enforce server access settings:	No
Authentication options:	
Name & password:	Yes
Anonymous:	Yes
SSL port number:	63149
SSL port status:	Disabled
Authentication options:	
Client certificate:	N/A
Name & password:	No
Anonymous:	Yes

Figure 5-8 DIIOp settings in the server document

You have to also specify access control for the DIIOp. This can be done on the Security tab of the server document. Enter the required user names to the fields shown in Figure 5-9.

Run restricted Java/Javascript/COM:	administrators
Run unrestricted Java/Javascript/COM:	administrators

Figure 5-9 Setting up ACL for DIIOp in server document

More information about how to configuring DIIOp for Domino is found in the Lotus Domino Administrator 6 Help database.

When loaded at the Domino server, the DIIOp task creates a file named DIIOp_IOR.TXT in the HTML directory, under the Domino directory. The file should be in ASCII format. DIIOp provides to the client an IOR number that is basically readable, but may make no immediate sense. The starting signature is "IOR:". IOR numbers presents information about the Domino server; you can

also use this IOR number to create a session to Domino instead of using the server name.

DIIOp appears to need DNS for name resolution. This was more apparent on the iSeries™ and zSeries® machines. When using a Host file or no known name resolution, loading DIIOp may respond that it is starting on host.domain.root, then end with the error "Cannot determine hostname or host ip address." If this happens, fix your DNS resolution and try again.

DIIOp-related console commands

The **Te11** commands defined in Table 5-1 relate to DIIOp tasks. You can issue them from the Domino Server console.

Table 5-1 DIIOp-related console commands

Command	Result
Te11 DIIOp Dump Config	Provide a list of the configuration data that DIIOp is using from the Domino Directory. Using dump, the configuration is written to the file diiopcfg.txt in the server's data.
Te11 DIIOp Show Config	Provide a list of the configuration data that DIIOp is using from the Domino Directory. Using show, the configuration is displayed on the server console.
Te11 DIIOp Log= <i>n</i>	This command determines the amount of information the DIIOp will log about it's operation. Valid values for <i>n</i> are as follows: 0 Show Errors & Warnings only 1 Also show informational messages 2 Also show session init/term messages 3 Also show session statistics 4 Also show transaction messages The setting of this command is saved in the NOTES.INI variable DIIOpLogLevel. Any change that is made to the DIIOp log level will be used the next time the server is restarted.
Te11 DIIOp Refresh	Use this command to reload the configuration data that DIIOp is using from the Domino Directory and from notes.ini. By default DIIOp incorporates changes from the Domino Directory every 3 minutes, or as often as specified in the NOTES.INI parameter DIIOpConfigUpdateInterval. The Refresh command will force DIIOp to look for changes in the configuration and apply them immediately.

Command	Result
Tell DIIOP Show Users or Tell DIIOP Show Users D	<p>Show all the current active users known to the DIIOP task. This list is similar to the server console command show tasks but it includes more information. Appending D to this tell command means the list of current users will also include the databases the user has open, along with a count of objects that are in use.</p> <p>Example: tell diiop show users d</p> <pre> UserName ClientHost IdleTime ConnectTime SessionId Anonymous 9.95.74.178 0:00 0:00 SN00048DE22 perf/user1.nsf Objects in use: Databases: 1 Views: 0 Documents: 0 Items: 0 Others: 0 Users: 1, NetworkConnections: 1 IOP Show Users D </pre>

DIIOP debugging methods

If you are have problems while working with DIIOP, there is a set of debug parameters which might help you to solve them.

- ▶ **DIIOP_DEBUG=1** sets the server to report verbose information about DIIOP.

You can also debug in certain sections inside of the debug task like this:

```

diiop_debug_connmgr=1
diiop_debug_cookie=1
diiop_debug_cwbase=1
diiop_debug_leaks=1
diiop_debug_objmgr=1
diiop_debug_refdata=1
diiop_debug_sslcert=1
diiop_debug_userobj=1

```

- ▶ Other debug parameters you might find useful when working with CORBA are:

```

DEBUG_ORB_OI=1
DEBUG_ORB_SOCKETS=1
DEBUG_ORB_PARAMS=1
DEBUG_ORB_THREADS=1
DEBUG_ORB_SHRED=1
DEBUG_ORB_SERVER=1
DEBUG_TCP_ALL=1

```

DIIOP transactions

Another good way to administer DIIOP is to look at transactions the server may already have done with DIIOP. This is done by typing **SH STAT DIIOP** from the server console. The server will display the following for each type of transaction:

- Total number of NRPC transactions (Count)
- Minimum duration of the transaction (Min)

- Maximum duration of the transaction (Max)
- Total time to perform all transactions (Total)
- Average time to perform the transaction (Avg)

All times are reported in milliseconds. This command identifies transactions that require excessive amounts of time. An example of the output from this command is in Figure 5-10.

```

itsotest-dom6/itsodom6: Lotus Domino Server
> sh stat diiop
DIIO.Objects.Database.Current = 0
DIIO.Objects.Database.Peak = 2
DIIO.Objects.Database.Total = 2
DIIO.Objects.DocumentCollection.Current = 0
DIIO.Objects.DocumentCollection.Peak = 0
DIIO.Objects.DocumentCollection.Total = 0
DIIO.Objects.Document.Current = 0
DIIO.Objects.Document.Peak = 0
DIIO.Objects.Document.Total = 0
DIIO.Objects.Item.Current = 0
DIIO.Objects.Item.Peak = 0
DIIO.Objects.Item.Total = 0
DIIO.Objects.OTHER.Current = 0
DIIO.Objects.OTHER.Peak = 1
DIIO.Objects.OTHER.Total = 1
DIIO.Objects.Session.Current = 0
DIIO.Objects.Session.Peak = 3
DIIO.Objects.Session.Total = 3
DIIO.Objects.ViewEntryCollection.Current = 0
DIIO.Objects.ViewEntryCollection.Peak = 0
DIIO.Objects.ViewEntryCollection.Total = 0
DIIO.Objects.ViewNavigator.Current = 0
DIIO.Objects.ViewNavigator.Peak = 0
DIIO.Objects.ViewNavigator.Total = 0
DIIO.Objects.View.Current = 0
DIIO.Objects.View.Peak = 0
DIIO.Objects.View.Total = 0
DIIO.Requests.HTTP.Current = 0
DIIO.Requests.HTTP.Peak = 0
DIIO.Requests.HTTP.Total = 0
DIIO.Requests.IIOP.Current = 0

```

Figure 5-10 DIIO transaction information

5.4 Domino objects for Java API

Java Domino classes are created by modifying some of the LotusScript Extension (LSX) architecture to include a Java “adapter” to compose the new Java Domino classes. The Java Domino classes have similar functions to some of the LotusScript Domino back-end objects. You can use these classes from any Java program. Internally, Java Notes classes execute the same C++ code as the LotusScript Domino back-end objects, only the language syntax is different.

Domino objects architecture

The Domino objects class architecture is based on a conceptual containment model, where the containment model defines an object's scope. A container

object is always used to access objects it contains. For example, you use a Session object to get Database objects, and a Database object to create Document objects. In Java, you cannot create lotus.domino objects using the “new” modifier. All lotus.domino objects must be created with lotus.domino methods that flow from the root Session object. This is illustrated in Figure 5-11.

Since one Domino object may be contained by several others, a full object diagram is beyond the scope of this document. Complete details about all Domino objects is in the documentation for Domino Toolkit for Java. However, some of the key containment relationships are as shown in Figure 5-11.

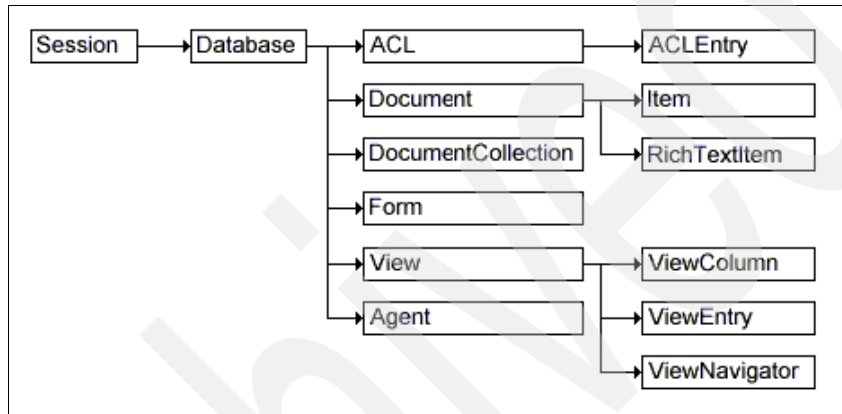


Figure 5-11 Domino Object Model

In practice, you will implement Notes API to your portlet code, adding a NCSO.jar/NCSOW.jar file (for remote connection) or NOTES.jar (for local connection) to the library. These files can be found in the Domino Toolkit for Java or on the Domino server.

Type `import lotus.domino.*;` into your portlet code, and you are able to use Notes API classes in your code.

All Domino classes have methods and properties.

Methods

Method names are written with the first character in lower case (for example, `getFirstDocument`). Of course, there are exceptions (such as `FTSearch`). One of the most important methods every Domino object has is `recycle`. The `recycle` method unconditionally destroys an object and returns its memory to the system.

Properties

To access properties in Java, you also have to use methods. In Java, properties are implemented through methods, known as accessors, which use the following

naming convention: The name of a method used to get the value of a non-boolean property is the name of the property prefixed with `get`.

Recycling Domino objects

The `recycle` method unconditionally destroys an object and returns its memory to the system. All `lotus.domino` classes contain the following method:

```
public void recycle()
```

`Session` contains the following method, where the vector contains the Domino Objects to be recycled. This method effectively batches `recycle` operations and is especially efficient for remote (IIOP) calls.

```
public void recycle(java.util.Vector objects)
```

You recycle an object by calling the `recycle` method, for example `doc.recycle()`; where `doc` is the object to be recycled.

Important: Java has no knowledge of the heavyweight back-end Domino objects, only the lightweight Java objects representing them. Garbage collection has no effect on Domino objects unless you first explicitly recycle them.

If your system appears to have memory problems, try to recycle, but adhere to the following guidelines:

- ▶ Recycle an object only if it is no longer needed.
- ▶ Recycle an object in the same thread in which it is created.
- ▶ Recycling a parent recycles all the children.
- ▶ In `Session`, call `recycle` only after all threads exit (local).
- ▶ Loops enumerating documents or items are good candidates for recycling.

Important: In a remote (IIOP) environment, *recycle releases resources* on the server. Although a client-side cache exists, the Java object can no longer communicate with its remote counterpart.

In a remote (IIOP) environment, `recycle` can be called from any thread on any object. Results are undefined if you attempt to use a recycled object. No error is guaranteed.

Commonly used Domino classes

Here we have gathered some information about the most commonly used Domino classes, which you may need in your Domino portlet developing work.

Complete details about the Domino Java API are in Domino Toolkit for Java product documentation.

Session class

Session class is the root of the Domino Objects containment hierarchy, providing access to the other Domino objects, and representing the Domino environment of the current Java program.

Important: You must always define at least session to Domino to get any further.

For stand-alone applications, like portlets, use one of the `NotesFactory.createSession()` methods.

For a local session, the Session object represents the machine on which the code is running. A reference to the current server, such as a null server parameter, means the local Notes/Domino environment. You can access servers connected to the local environment by specifying their names.

For a remote (IIOP) session, the Session object represents the Domino server handling the remote requests. A reference to the current server, such as a null server parameter, means that Domino server. You cannot access other servers.

You have different ways to do that; several are illustrated in the following examples. More samples are in the Domino Toolkit for Java documentation.

Example 5-1 Creating session to Domino using servername

```
session = NotesFactory.createSession("<servername>", "<username>", "<password>")
```

Example 5-2 Creating session to Domino using IOR

```
session = NotesFactory.createSessionWithIOR("<IOR>", "<username>", "<password>")
```

Session class has properties like `Addressbooks`, `NotesVersion`, `Platform`, or `UserName`.

Some of the most useful methods are `getDatabase`, `getDbDirectory`, `getEnvironmentalValue`, and `recycle`.

Database class

Database class represents a Domino database. A database must be open before you can use all the properties and methods in the corresponding database object. In most cases, the class library automatically opens a database for you.

Use `isOpen` properties to do the proper checking and add handling for the exceptions.

`Notes` throws an exception when you attempt to perform an operation for which the user does not have appropriate access. The `NotesException` class is discussed later in this section.

The properties and methods that you can successfully use on a Database object are determined by the user's access level to the database, as specified in the database access control list. The ACL determines if the user can open a database, add documents to it, remove documents from it, modify the ACL, and so on.

The user's access level to the server on which the database resides is determined by the Server document in the Domino Directory.

Example 5-3 Using the `isOpen` method

```
Database db = session.getDatabase("servername", "names");
    if (!db.isOpen())
        getPortletLog().debug("Database does not exist on server");
    else
        getPortletLog().debug("Title of the database is : \" +
db.getTitle()+ \"\");
```

The Database class has properties like `getACL`, `Agents`, `Views`, and `Forms`, just to mention a few.

The Database class includes methods like `createDocument`, `getDocumentByID`, `getView`, and `replicate`, among others.

View class

The View class represents a view or folder of a database and provides access to documents within it.

You access a view or folder through the Database object that contains it, in one of the following ways:

- ▶ To access a view or folder when you know its name or alias, use `getView`.
- ▶ To access all the views and folders in a database, use `getViews`.

`Returned` is a View object or a vector of View objects that represent public views and folders in the database. To access a view or folder when you have a view entry, use `getParent` in `ViewEntry`.

A View object provides access to ViewEntry, ViewEntryCollection, and ViewNavigator objects:

- ▶ A ViewEntry object represents a row in the view and can be a document, category, or total. A document entry provides a handle to the associated Document object.
- ▶ A ViewEntryCollection object provides access to selected or all document ViewEntry objects. (Excluded are category and total ViewEntry objects.)
- ▶ A ViewNavigator object provides methods for navigating through selected or all ViewEntry and Document objects.

A View object provides access to ViewColumn objects, which contain information on each column in the view.

The code in Example 5-4 finds the "\$All" hidden view in a database, gets the first document in the view, and finally gets an item value from the document.

Example 5-4 Accessing the views of the database.

```
Database db = session.getDatabase ("servername", "names");
View view = db.getView("$All");
Document doc = view.getFirstDocument();
getPortletLog().debug( (doc.getItemValueString("Subject")));
```

The View class has properties that include aliases, columns, iscalendar, and Rowlines, among others.

Some of the View class methods are FTSearch, getdocumentByKey, getFirstDocument, and getChild.

Document class

The Document class represents a document in a database.

To create a new Document object, use createDocument in Database.

To access existing documents, do one of the following:

- ▶ To get all the documents in a database, use getAllDocuments in Database.
- ▶ To get a document based on its position in a view, use a View object.
- ▶ To get a document based on its position in a response hierarchy, use a View object. To get all documents that are responses to a particular document, use getResponses in Document. To get a response document's parent document, use getParentDocumentUNID in Document followed by getDocumentByUNID in Database.

- ▶ To get all the documents that match a full-text search query, use `FTSearch` in `Database` or `FTSearch` in `View`.
- ▶ To get all the documents in a database that meet search criteria, where the criteria are defined using the formula language, use `search` in `Database`.
- ▶ To get a document based on its Note ID or UNID, use `getDocumentById` or `getDocumentByUNID` in `Database`.

Once you have a view, you can navigate to a specific document using methods in the `View` class.

Once you have a collection of documents, you can navigate to a specific document using methods in the `DocumentCollection` class.

Important: After you create, modify, or delete a document, you must save the changes by calling the `save` method. If you don't call `save` before the program finishes, all of your changes to a `Document` are lost.

If you create and save a new document without adding any items to it, the document is saved with one item `"$UpdatedBy"`. This item contains the name of the creator of the document.

The code in Example 5-5 first creates a new document and then sets values for two fields in the document. Finally, the code saves the document.

Example 5-5 Use of `Document` class

```
Document doc = db.createDocument();
doc.replaceItemValue("Form", "Main Topic");
doc.replaceItemValue("Subject", "New building");
doc.save(true,true);
```

Item class

The `Item` class represents a discrete value or set of values in a document.

The client interface displays items in a document through fields on a form. When a field on a form and an item in a document have the same name, the field displays the item (for example, the `Subject` field displays the `Subject` item).

All items in a document are accessible programmatically, regardless of what form is used to display the document in the user interface.

To create a new `Item` object:

- ▶ To create a new `Item` object from scratch, use `replaceItemValue` in `Document`. The method `appendItemValue` also creates an item, but creates another item

of the same name if the specified item exists. Use `replaceItemValue` unless your intent is to create another item with the same name (not recommended).

- ▶ To create a new `Item` object from one that already exists, use `copyItemToDocument`, `copyItem`, or `replaceItemValue` in `Document`.

You must call `save` on the document if you want the modified document to be saved to disk. The document won't display the new item in the user interface unless there is a field of the same name on the form used to display the document.

Explicitly call `setSummary` and specify `true` if you want the value to be displayed in a view or folder.

To access `Item` objects:

- ▶ To access an item when you know its name, use `getFirstItem` in `Document`.
- ▶ To access all the items in a document, use `getItems` in `Document`.

`Document` has methods to access items without creating an `Item` object. You need to know the name of the item.

- ▶ To get an item's value, use `getItemValue`.
- ▶ To create a new item or set an item's value, use `replaceItemValue`.
- ▶ To check for the existence of a particular item in a document, use `hasItem`.
- ▶ To delete an item from a document, use `removeItem`.

`RichTextItem` inherits the properties and methods of `Item` and has additional properties and methods you can use to manipulate rich text.

After you create or modify an item, you must save the changes by calling the parent document's `save` method.

If you don't call `save` before the program finishes, all of your changes to an `Item` object are lost.

The code in Example 5-6 first creates a document, creates a text item without specifying data type and then one with a data type. `isSummary(True)` specifies that the item can be shown in a view. Finally, the document is saved.

Example 5-6 Using Item class in code

```
Document doc = db.createDocument();
// Create text item with implied data type
doc.replaceItemValue("Subject", "Creating items ...");
// Create text item explicitly specifying data type
Item textItem = doc.replaceItemValue("textItem", null);
```



```
textItem.setValueString("Finland");
textItem.setSummary(true);
// Save the document
doc.save(true, true);
```

Item class includes properties such as `isAuthors`, `Lastmodified`, `Reader`, and `ValueString`.

Some of the Item class methods are `copyItemToDocument`, `getMIMEEntity`, `parseXML`, and `transformXML`.

ACL class

ACL class represents the access control list (ACL) of a database. Every Database object contains an ACL object representing the access control list of that database. To get it, use `getACL` in Database, as shown in Example 5-7. `getFirstEntry()` returns an ACL entry object.

Example 5-7 Example using ACL class

```
Database db = session.getDatabase("servername", "names");
if (!db.isOpen())
    getPortletLog().debug("Database does not exist on server");
else
    getPortletLog().debug("Title of the database is : \" +
db.getTitle()+ "\"");
ACL acl = db.getACL();
ACLEntry entry = acl.getFirstEntry();
do {
    getPortletLog().debug(entry.getName()); }
while ((entry = acl.getNextEntry(entry)) != null);
```

The Database class has three methods you can use to access and modify an ACL without getting an ACL object: `queryAccess`, `grantAccess`, and `revokeAccess`. However, using these methods at the same time that an ACL object is in use may produce errors.

ACL class properties include `Roles`, `InternetLevel`, and `UniformAccess`.

Among the ACL classes are methods like `addRole`, `createACLEntry`, `RemoveACLEntry`, and `save`.

NotesException class

The NotesException class extends `java.lang.Exception` to include exception handling for Domino. The NotesError class defines public static final variables for Domino error codes.

To catch a Domino exception, specify the parameter of the catch clause as type `NotesException`. The `NotesException` class contains the following public variables:

- ▶ `NotesException.id`, of type `int`, contains the error code.
- ▶ `NotesException.text`, of type `String`, contains the error text.

The following code example demonstrates how to catch a Domino exception. The code prints the error code and error text if a Domino exception is thrown.

Example 5-8 Using NotesException

```
public Collection testViews()
    throws NotesException
ArrayList views = new ArrayList();
    try {
        View view = db.getView("ViewName");
        Document doc = view.getFirstDocument();
        while (doc != null) {
            String systemName =doc.getItemValueString("FieldOne");
            String showName =doc.getItemValueString("FieldTwo");
            doc = view.getNextDocument(doc);
        }
    } catch (NotesException e) {
        Log.error(this.getClass(),"Could not get the list of views.
NotesException occured: id = "
            + e.id+ ", text = "+ e.text,e);
    }
}
```

5.5 Domino Rich Text

Domino Rich Text fields can contain formatted text, collapsible sections, embedded objects, file attachments, and pictures. Rich text is widely used in Domino applications, so it is important to be able to display it in a Portlet.

In Chapter 1, “Introduction to portalizing Domino applications” we introduced the concept of Portlet patterns. We outlined that in a Portal environment there are two main ways to display content:

- ▶ Following the Display pattern by opening content in a new browser window.
- ▶ Embedding content inside of the Portlet. This can be achieved with the Integrated Portlet pattern.

In this section we discuss both options with respect to Domino rich text display.

Using the Portlet Display pattern

Following the Display pattern and opening content in a new browser window has the following requirements for the Domino application:

- ▶ You need a Web-enabled form.

You also need to consider that:

- ▶ Content is displayed outside of the Portlet
- ▶ The browser directly communicates with the Domino HTTP server. This might involve firewall issues.

Using iFrames

An alternative to opening a new browser window is the use of an iFrame Portlet. This approach still follows the Display pattern, but has a more seamless integration with the Portal. But you need to be aware of the following limitations of iFrame Portlets:

- ▶ iFrame Portlets are not real Portlets.
- ▶ They do not support Portlet cooperation.
- ▶ They can't use Portal themes.
- ▶ iFrame Portlets can't keep their state. Consider the workplace example that was introduced in "Case study objective: Sales Workplace" on page 50. There we have two Portlets. One is displaying a customer search Portlet and the other the customer details. If we search for a customer and select the link, the details Portlet displays the customer details information. If we now change the place to do some other work and later come back to our Sales Workplace, the details Portlet does not display anything anymore. It is displaying its default page, which is blank. This example shows the state problem that iFrame Portlets have: they are acting outside of the Portal context and therefore can not keep the state of the user's actions.

Embedding content into a Portlet

In this section we discuss integration methods that follow the Integrated pattern.

What options do we have to embed rich text inside of a Portlet?

One option is to display rich text as plain text only. This is useful, for example, for displaying abstracts. To accomplish this we can use:

- ▶ The Item's `getText` method in Java
- ▶ The `richtext` tag from the Domino JSP tag library

Note: The Domino 6 Tag library has a <Domino:richtext> tag to display rich text, but formatting is displayed only if the rich text was created using the RichText Java applet or the Microsoft rich text control.

In this case the rich text is saved as MIME (Multipurpose Internet Mail Extensions) inside of a Rich Text field. Programmatically you can access these items as Item, RichTextItem, or MIMEEntity objects.

Using the richtext tag to display rich text that was created with a Notes client will show plain text only. That means all formatting, images, sections, and URLs are lost.

To keep all the formatting and features of Domino Rich Text we have to take a different approach. Domino Rich Text is a very complex structure that can be programmatically accessed by using the Domino backend classes for Java.

Nevertheless the only way to convert Rich Text into HTML is to use the Domino HTTP task.

Displaying rich text using HTTP

The following steps outline how you can get the HTML representation of the Domino Rich Text. This approach has to be implemented in Java. Figure 5-12 illustrates the process.

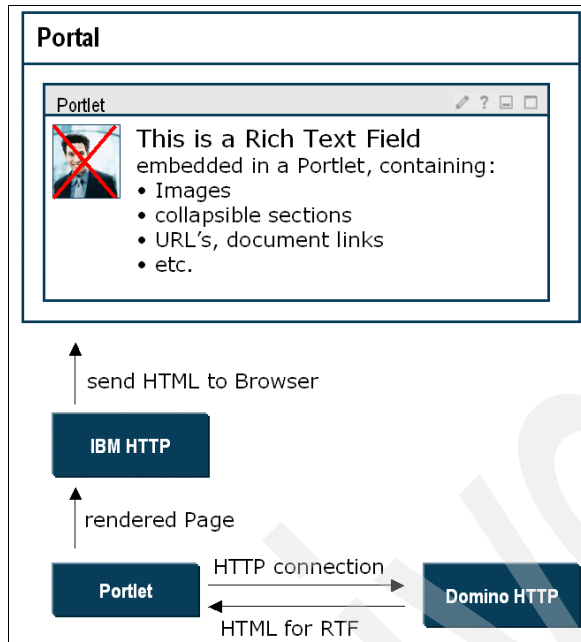


Figure 5-12 Embedding Domino rich text inside of a Portlet

1. Open a URL connection to retrieve the HTML for an entire Domino document.
2. Parse the HTML for the rich text field that you want to display.

To find the HTML that corresponds to the rich text field, it is best practice to indicate the start and end of the field by adding an HTML comment around the field on the Domino form.

3. Pass the resulting HTML to the JSP page for display.

After you have implemented this solution you will find that the formatting of the text is now displayed correctly, but that all Images are broken. The reason for this is that the HTML you retrieved from the Domino HTTP included relative URL references only. Since you are now displaying the captured HTML relative to the Portal server's HTTP task, the URLs can't be resolved.

You can solve this problem in either of the following ways:

- ▶ The IBM HTTP plug-in can be used as a reverse proxy to Domino 6 servers. For this communication to occur, the appropriate WebSphere Application Server 4.0.3 or later plug-in must be installed on the front-end server. These plug-ins recognize HTTP requests for Domino applications and pass them along to the Domino server. Other HTTP requests are handled by the front-end server itself.

- Parse the Domino HTML and make all URL references absolute to the Domino HTTP server. Using regular expressions can help to implement this process.

Either of these techniques will solve the image URL problem and display the images correctly.

URL rewriting

The second option (URL rewriting) has the disadvantage that the user's browser will need to be able to communicate with the Portal server HTTP and the Domino server HTTP as well. This communication pattern might be prevented by firewall rules, as illustrated in Figure 5-13.

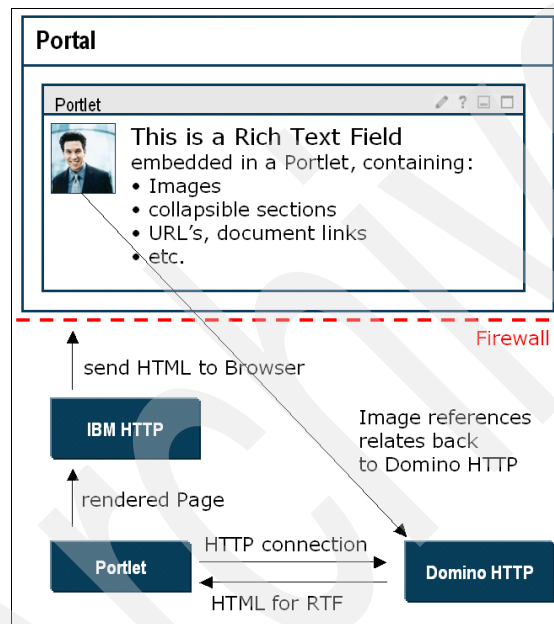


Figure 5-13 Potential firewall issues

Since the image reference is referring back to the Domino server's HTTP task, the browser needs to be able to access the Domino HTTP server to display the image.

This also implies that the user needs to have access rights to the Domino server and database in which the image resides.

The URL challenge

But we still have an even bigger challenge ahead of us. Specifically, all the URLs in the HTML produced by Domino HTTP server are not Portal compliant. That

means that if a user clicks on a link inside of the Rich Text, the called URL will take over the entire browser URL. As a result, the Portal interface is lost.

The workaround for this problem is illustrated in Figure 5-14.

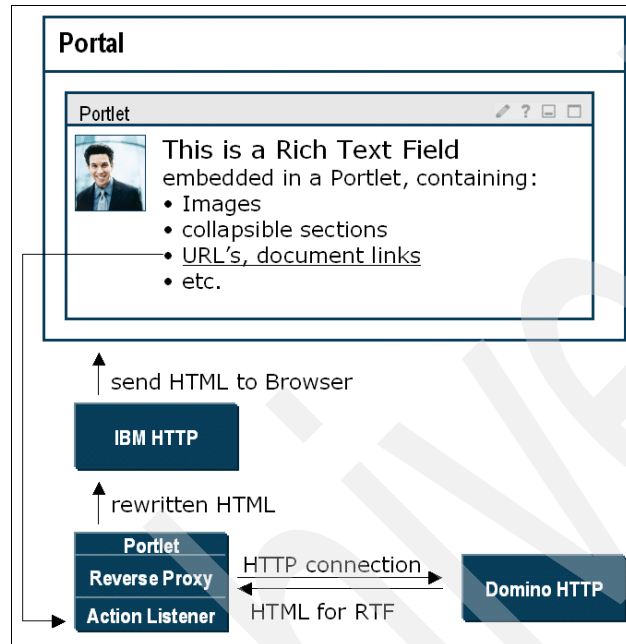


Figure 5-14 Rich Text display with reverse proxy and URL re-writing

- ▶ Implement a reverse proxy in Java that handles, for example, the display of resources with relative URLs.
- ▶ Parse the captured HTML using regular expressions and rewrite URLs into Action URIs.
- ▶ By implementing an action listener, you can then respond to the Action URIs and allow the user to navigate links in the HTML. This ensures that all following pages will be displayed inside the Portlet as well.

The discussion shows that Domino rich text integration into Portlets is a complex and potentially challenging procedure.

IBM is currently working on a solution that will allow easy Domino rich text integration. Third party vendors like CONET have also implemented solutions for seamless rich text embedding in their products.

5.6 Lotus Collaborative Components API

Lotus Collaborative Components are building blocks (APIs and JSP tag libraries) for integrating the functionality of Lotus Domino, Lotus Sametime, Lotus QuickPlace, and Lotus Discovery Server into the portal. The IBM redpaper *WebSphere Portal 4.12 Collaboration Services*, REDP 0319 discusses this topic in more depth.

The Collaborative Components can be divided into Java Classes and Methods (cs.jar) and JSP, JavaScript tag libraries (people.tld and menu.tld). JSP tag libraries were discussed in Chapter 4, “Using custom Domino JSP tag libraries”. This section covers only the core API of the Collaborative Components.

Collaborative Components are implemented in Java and include no platform-specific code, so they can be used on any J2EE-compliant server.

Overview of key objects

The CS.jar, Collaboration Service package contains all the Java implementations of the collaborative components. There are classes and methods for leveraging Domino, QuickPlace, Sametime, and Discovery Server.

For more detailed information about these classes, see the documentation provided as a Javadoc within the Collaborative Components enterprise application: the cs.ear file.

Some key objects are the following:

- ▶ **CSEnvironment**: Initializes the server environment and retrieves credentials for the logged-in user.
- ▶ **CSCredentials**: Represents the authentication information for the logged-in user. CSFactory uses these credentials to authenticate against the worker servers (Sametime, QuickPlace, and so forth).
- ▶ **CSFactory**: Constructs the Java service objects and generates a connection to the appropriate worker server.

The CSFactory object instantiates these service objects. The developer is responsible for calling the cleanup() member function on each object before they go out of scope. This cleanup function releases any resources a service object holds.

- ▶ **DominoService**: Provides standardized access to all versions of Domino from R4.67.
 - ViewInfo: Represents information about a view in a Domino database.
 - ColumnInfo: Contains information about the columns in a Notes view.

- RowInfo: Represents data from one row in a Notes view.
- CalendarDayInfo: Encapsulates all the calendar entries for a particular day.
- EntryInfo: Contains information about a single calendar entry.
- ▶ QPService: Provides the ability to create a QuickPlace.
- ▶ PeopleService: Provides Sametime awareness (users logged onto Sametime) plus the ability to obtain information about a specified person, such as their e-mail address.
- ▶ DiscoveryServerService: Provides the ability to discover, search for, and retrieve Categories, People, Places, and Documents from a Discovery Server.

How to get started

1. Set up a development environment.

At a minimum you must install the Collaborative Components jar files on your development machine and make them available to WebSphere Studio Application Developer. The required jar files are:

- cs.jar
- commres.jar
- kdsapi.jar
- kdsw.jar
- ncsow.jar
- stcommsrvrtk.jar

The simplest way to accomplish this task is to load the cs.ear file into WebSphere Studio Application Developer. This provides the additional benefit of installing the tag libraries and the sample programs that come with the Collaborative Components package. To install cs.ear on your machine, choose the **Import** option from the WebSphere Studio Application Developer file menu and select **EAR File** as the import source. On the EAR Import screen enter a new project name and enterprise application project name. When you are finished, your screen should look similar to the one in Figure 5-15 on page 272.

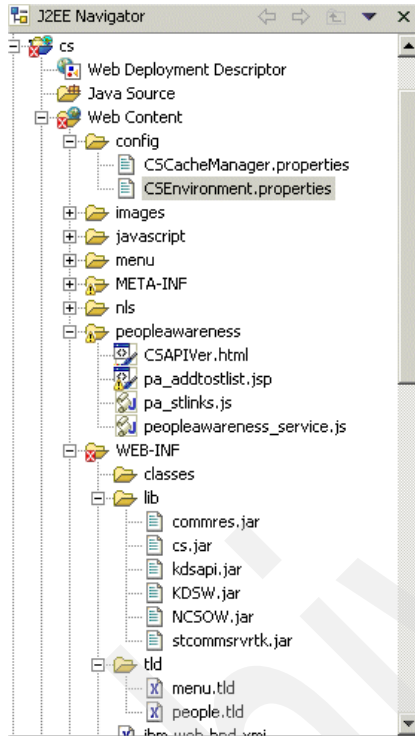


Figure 5-15 Structure of portlet after cs.ear file has been imported

2. Configure the properties file according to your Domino servers.

In the project you just created, expand the folder <project name>/Web Content/config. Edit the CSEnvironment.properties file and enter the host names for the Discovery Server, Domino Directory, and QuickPlace servers.

Example 5-9 CSEnvironment.properties file example in config folder

```
#####
# SAMETIME properties
# Only valid property is enabled. Other values configured
# through hostAddress.xml
#####

CS_SERVER_SAMETIME.enabled=true

#####
# QUICKPLACE properties
# enabled indicates whether or not QP creation is allowed.
#####
```

```

CS_SERVER_QUICKPLACE.enabled=true
CS_SERVER_QUICKPLACE_1.hostname=my.server.com
CS_SERVER_QUICKPLACE_1.version=3.0

#####
# DISCOVERY SERVER properties
#####

CS_SERVER_DISCOVERY_SERVER.enabled=true
CS_SERVER_DISCOVERY_SERVER_1.hostname=my.server.com
CS_SERVER_DISCOVERY_SERVER_1.version=2.0

#####
# DOMINO DIRECTORY properties
# (LDAP server)
# Only valid property is hostname. Leave enabled flag as true.
#####

CS_SERVER_DOMINO_DIRECTORY.enabled=true
CS_SERVER_DOMINO_DIRECTORY_1.hostname=my.server.com
#####
#
# LDAP server performance properties
# (LDAP server)
# Leave enabled flag as true, if using WMM.
# Setting this flag to false will improve performance
# while using Domino as the primary (and only) LDAP server
# in the Portal.
#####

CS_PERF_PROP_USEWMM.enabled=true

```

5.7 Domino 6 new features for DIIOP

In this section we mention some of the DIIOP-related new features of Domino 6. A complete discussion of all new Domino 6 features is in the Lotus Domino Administrator 6 Help database.

- ▶ New DIIOP task

DIIOP has been substantially rewritten to improve its performance and scalability. The new task now uses the same code for network operations as some of the other server tasks, for example POP3, LDAP, and IMAP. Consequently, DIIOP now also takes advantage of the same thread pooling algorithms, and will use I/O completion ports on the supported platforms.

- ▶ HTTP Support for diiop_ior.txt

DIIOp now has limited support for the HTTP protocol. As part of the `NotesFactory.createSession(hostname,...)` method that is available for creating sessions to the Domino server, the HTTP protocol was being used to deliver the Initial Object Reference to the client program. Traditionally, this request was handled by the HTTP task. This meant that the HTTP task had to be configured and running to use the Domino classes in any substantial way.

With this release, DIIOp has enough HTTP support to handle the request for the Initial Object Reference. So, if HTTP is not desired on the Domino server, a Java program can do the following to create a `Session` to the Domino server:

```
Session s = NotesFactory.createSession("<hostname:63148>", "<username>",  
"<password>");
```

where `:63148` is the syntax that is used to specify the port number of the task that you want to issue the HTTP request for Initial Object Reference.

5.8 Object pooling

Pooling Domino objects like sessions and views can help to significantly improve performance in a Portal environment. In a typical workplace we usually have more than one Domino Portlet on a page. In addition, it is usual for many users to have access to the same content. These circumstances can lead to limited-resource problems.

Limited resources can cause performance bottlenecks when there are not enough resources to meet clients' demands. For example, DIIOp sessions to Domino servers require non-trivial amounts of time to create and destroy. Often, in high-throughput scenarios, Portlets must wait for a session object to become available, creating a bottleneck in the flow of the application.

With object pooling, many Portlets can share a limited resource such as a session, using it only when they need it. In this way, the performance cost of creating and destroying the session is reduced.

Implementing an Object Pool can have the following advantages:

- ▶ More scalable, resourceful

You can share Domino objects between multiple Portlets and reuse objects for users in the same user group.

- ▶ Enables load balancing

DIIOp can't be used to access a cluster of Domino servers. A pool manager can manage sessions to multiple Domino servers at a time and allow load balancing and fail-over.

- ▶ Thread safe

The Pool Manager is completely thread safe, meaning that it can be accessed safely from any number of threads—a situation that is common for Portlets.

- ▶ Helps with DIIOp session time-out

DIIOp session time-out occurs when a session is idle longer than the DIIOp session time-out setting. By default this is 60 minutes. After this time the session and all its associated objects become invalid.

- ▶ Keeps Domino server healthy

It is good practice *not* to use remote Sessions for long periods of time. Recycling remote Session objects and all associated objects can help to keep the Domino server healthy, but Session recycling is tricky to manage.

- ▶ Ease of use

You do not have to consider the possible state of a Domino object; you simply use the object. For example, if you want to retrieve data from a view, you do not need to know whether a session and database connection have been previously established, you can just retrieve our data.

Object pool

An object pool is a set of limited resources, such as sessions, that can be reserved for use by clients and then returned to the pool (for probable reuse) when the object is no longer needed. Reserving and returning pooled objects avoids the overhead of separately creating and destroying an object each time a client requests it. Multiple object pools can be used. For example, one object pool might contain session connection objects, and another pool might contain database or view objects.

Virtual and physical objects

Without object pooling, whenever a Portlet requests an object, a physical object is created, and is destroyed when no longer needed.

By contrast, when a Portlet uses object pooling, the request for a poolable object generates a virtual object instead. The virtual object supports all the methods of the requested object, but the Portlet sees only the virtual object.

When a Portlet calls an interface method from the virtual object, the virtual object's implementation requests a physical object from the pool and delegates

the request to the physical object. When the request is complete, the Portlet returns the physical object to the Object Pool Manager for use by other virtual objects.

Client

In the context of object pooling, a client is the Portlet code that calls the Pool Manager. The Portlet requests objects, and the Object Pool Manager fulfills the requests or relays an error back to the calling Portlet.

Object Pool Manager

The Object Pool Manager is a service used by one or more Portlets. In response to clients' requests for objects, the Object Pool Manager controls one or more pools (for example, sessions and views) by reserving and releasing the objects in the pool. The Object Pool Manager performs the following tasks:

- ▶ Queues virtual objects' requests for physical objects.
- ▶ Marks physical objects as either free or reserved.
- ▶ Attempts to create physical objects when necessary.
- ▶ Destroys physical objects in a pool, based on idle time or usage limits.

Object pooling process

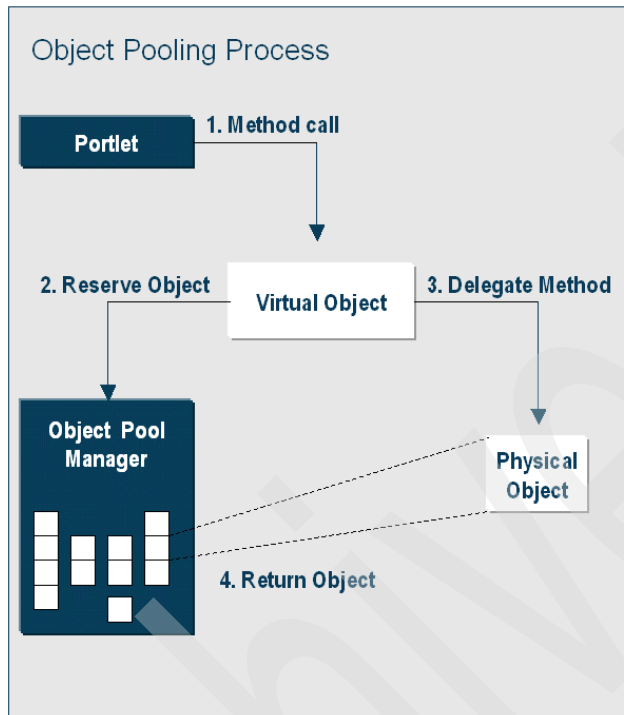


Figure 5-16 Object pooling process

The object pooling process, illustrated in Figure 5-16, is as follows:

1. The Portlet calls an interface method on the virtual object.
2. The virtual object's implementation reserves a matching physical object from a named pool.
3. The method call is delegated to the physical object.
4. When the method call is completed, the physical object is returned to the appropriate pool for use by other virtual objects.

The Object Pool Manager uses a timer thread that periodically releases unused physical objects after a time-out.

How the Object Pool Manager works

As illustrated in Figure 5-17, when a Portlet makes a request to reserve an object, the Object Pool Manager processes the request through the following possible phases:

- ▶ Matching a pooled object
- ▶ Creating a pooled object
- ▶ Replacing a pooled object
- ▶ Queuing a request
- ▶ Returning a FAILURE state

Details about these phases follow the figure.

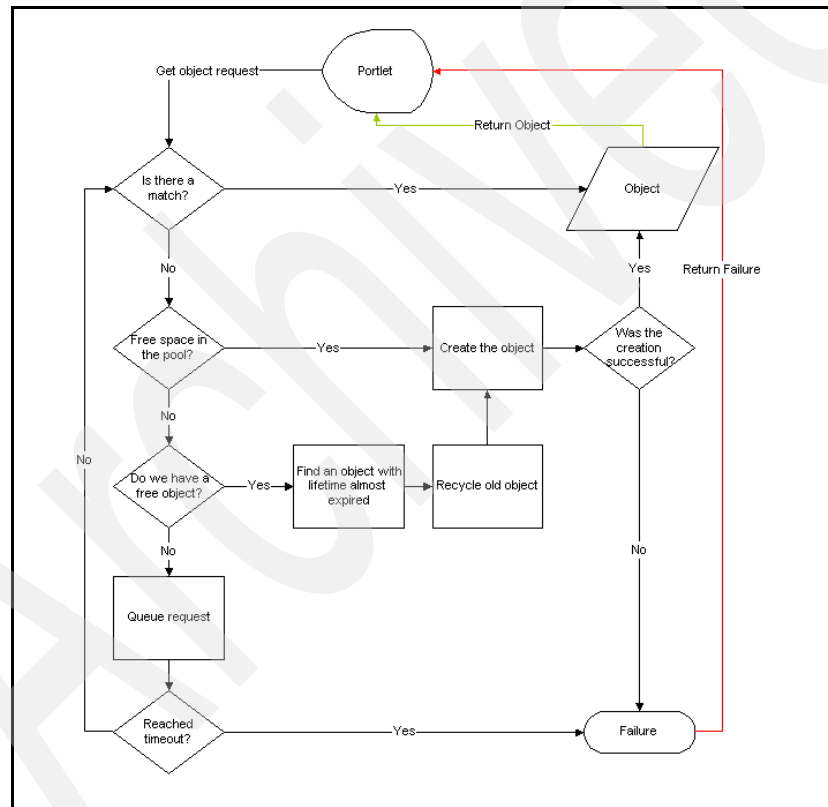


Figure 5-17 Pool Manager process flow

Matching a Pooled Object

A client makes a request of a virtual object. It communicates with the Object Pool Manager to bind the virtual object to a matching physical object. The virtual

object then repeats the client's request, making this same request of the matched physical object.

If a match is found, the Object Pool Manager reserves the matching physical object and sends it back to the requesting client. If no matching object is found, then the Object Pool Manager proceeds to the next phase of processing logic—attempting to create a physical object.

Creating a Pooled Object

If no matching object is found, the Object Pool Manager checks its `MaxPoolSize` setting before trying to create a physical object. `MaxPoolSize` determines the maximum number of objects a pool can contain.

If the number of objects in the pool is less than `MaxPoolSize`, then the pool is allowed to grow. As a result, the Object Pool Manager calls the `CreateObject()` method. A successful call returns a physical object to the requesting client. An unsuccessful call indicates that a critical resource is unavailable. For example, if an application requests a session but the Domino server is down, then the call to `CreateObject()` would fail, and a `FAILURE` state would be returned immediately.

Replacing a Pooled Object

If an object pool is at the limit set by `MaxPoolSize`, then the pool is considered full. As a result, instead of trying to create an object, the Object Pool Manager attempts to recycle an almost expired object and create a new one that matches the one requested.

Only unreserved objects can be replaced, so the Object Pool Manager first checks whether any objects are free. Assuming there are unreserved objects, the Object Pool Manager next calls the `StealObject()` method to determine which object to replace.

The Object Pool Manager calls the `ReleaseObject()` method to destroy the object to be replaced. As a result, the pool now contains one slot to fill, and the `CreateObject()` method is immediately called.

Queuing a Request

The fourth phase of processing occurs under the following conditions: if no matches are found, if the pool is full, and if none of the objects is free to be replaced. Under these conditions, the Object Pool Manager queues the request. The request waits until an object is returned to the pool. Pooled objects are returned to their originating pool when a client is done using them.

If the waiting period is within a maximum allowable idle time, then the request processing starts over from the beginning, with an attempt to match the newly

returned object. The maximum allowable idle time is configurable through the MaxWait variable in the registry.

Returning a FAILURE State

If the request is queued for longer than the MaxWait time, then the request finally fails. Frequent request failures may indicate a need for the server administrator to review an object pool's configuration settings.

Maintenance features of the Pool Manager

In addition to processing object requests, the Pool Manager is also responsible for maintenance of the Object Pool. The Pool Manager prevents session time-outs and destroys physical objects in the pool, based on idle time or usage limits.

What the Pool Manager can do for you

In the context of Domino Portlet development you can use the Pool Manager to manage, for example, DIIOP sessions and associated view objects. The following scenarios illustrate the advantages of using the Pool Manager:

- ▶ One user, multiple Portlets

In this case pooling session objects will help you to reduce the access time to a page since the session object is created only once and then shared by all Portlets.

- ▶ User group, multiple Portlets

This shows the efficiency of the Pool Manager best. Besides sharing sessions for users of the same group, we can share view objects as well.

On Portal places you will find that this is a common scenario. Users having access to a Place usually have the same access rights to the underlying Domino applications as well.

- ▶ One user one Portlet

In this case we do not have any advantage, aside from the fact that implementing the Object Pool once is going to reduce coding time for Domino Portlets in the future.

You can see that implementing a Pool Manager has a lot of advantages over handling Domino objects manually. It especially helps to avoid problems like session time-out and DIIOP memory leaks. It also ensures thread-safe access to view objects, and allows object sharing and the implementation of load-balancing to ensure high performing and scalable environments.

Taking the Pool Manager to the next level

So far we have used the Pool Manager only to manage physical Domino objects. Earlier in this chapter we introduced the Model-View-Controller design pattern. If you want to implement your Portlets in this manner, you should consider transforming the physical Domino objects into abstract objects. You then can use these abstract objects to render the output on the JSP pages.

Introduction to Queries

If you look at a Portlet you see that most of the time it displays information in a view-like manner. The data can be the result of displaying a view or a search result. In many cases you will also find navigation that enables the user to navigate through the data.

A representation for this can be seen as a *Query object*.

The query is a collection that is used on the JSP page to display the result of an executed Query Definition.

To create a Query Definition the Portlet specifies the server name, database, and view name. It can set page size and maximum result size. It also defines the columns to be included in the results.

The Query Definition is then submitted to the Pool Manager that executes it and sends back the resulting Query object. This is then passed to the JSP for rendering. The Pool Manager also returns a query ID. This can be used in subsequent calls to the same query to retrieve, for instance, following pages without having to execute the query again.

Caching query results is also possible and another way in increasing Portlet performance. In addition, you can envision the reuse of queries between users in the same user group.

Access to Domino documents

From the query object, the next logical step is to allow access to Domino documents. This can be implemented using the Pool Manager as well. If you have access to a Domino document it is also easily possible to implement write access.

Summary of object pooling

Abstracting Domino objects through the use of virtual objects is an elegant approach to implement the Model-View-Designer design pattern. It also helps in building better performing Domino Portlets.

You can find further information on creating object pools from the Jakarta “Commons Pool” project at

<http://jakarta.apache.org/commons/pool/>

This project provides an Object-pooling API, with three major features:

- ▶ A generic object pool interface that clients and implementors can use to provide easily interchangeable pooling implementations.
- ▶ A toolkit for creating modular object pools.
- ▶ Several general purpose pool implementations.

We can think of many ways to extend the concept of query objects even further. Consider, for example, a Compound Query that was able to aggregate the results of multiple underlying query objects. This would allow you to build views across multiple Domino views and databases.

The discussion of this topic is out of the scope of this redbook. Third-party vendors like CONET have implemented very sophisticated data management concept in their products. For example, their Domino Warehouse Model allows for combining of data from multiple Domino databases, as well as append, merge, and mix of Domino data.

5.9 Logging from portlets

Testing provides quality assurance and confidence in an application. Logging, on the other hand, equips the developer with detailed context for application failures. Logging and testing should not be confused; they are complementary techniques. When logging is wisely used, it can prove to be a valuable tool, and it is one of the key issues to consider in portlet development work.

Inserting log statements into your portlet code is a low-tech method for debugging it. However, some people argue that log statements pollute source code and decrease legibility. In the Java language, where a preprocessor is not available, log statements increase the size of the code and reduce its speed, even when logging is turned off. Given that a reasonably sized application may contain thousands of log statements, speed is of particular importance.

In this section we introduce two different mechanism to do logging. First we describe how the WebSphere Portal server itself can handle portlet logging. Then we explain how to build your own flexible mechanism to log from portlets.

WebSphere Portal-based log

WebSphere Portal uses the Logging Toolkit for Java (JLog) to help operate, maintain, and troubleshoot the portal and portlets. JLog creates detailed log and debug files while working in the background, thereby reducing the use of system resources.

JLog supports the logging of messages as errors and status information, and the logging of debugging messages called traces. These traces are useful for fixing problems; however, to save system resources, they are switched off by default.

Message and trace logging

Portlets can write message and trace information to log files, which are maintained in the `wps_root/app/web/WEB-INF/log/` directory. The log files help the portal administrator investigate portlet errors and special conditions and help the portlet developer test and debug portlets. The Portlet API provides the `PortletLog` class, which has methods to write message and trace information to the logs:

- ▶ `debug()` writes trace information to `PortletTraces.log`
- ▶ `info()` writes informational messages to `Portlet.log`
- ▶ `error()` writes error messages to `Portlet.log`
- ▶ `warn()` writes warning messages to `Portlet.log`

If you access the portlet log multiple times in a method, it is a good idea to assign the log reference to a variable, for example:

```
private PortletLog myLogRef = getPortletLog();
```

Since logging operations are expensive, `PortletLog` provides methods to determine if logging is enabled for a given level. Your portlets should write to the log of a given level only if the log is tracking messages on that level. For example:

```
if( getPortletLog().isDebugEnabled() )
{
    myLogRef.debug("Warning, Portlet Resources are low!");
}
```

Location of log files

The log files created by JLog are located in the directory `wps_root/app/web/WEB-INF/log`, and they are updated if an error occurs.

Each component of WebSphere Portal uses its own log file. These can be found when you log into the portal as administrator and select **Portal Administrator** → **Portal Settings** → **Enable tracing**. The most useful traces to switch on for portlet developers are traces that start with “Portlet,” for example, `PortletTraceLogger`.

To enable the creation of these messages for uses such as debugging during portlet development, you need to change the configuration files.

All configuration files ending with `TraceLogger.properties` contain the key `isLogging` which is set to `false`. To activate the logging of debug messages for the

relevant WebSphere Portal component, set this key to true. Any changes you make to the configuration files become active only after restarting the application server.

Example 5-10 Example code using logging

```
public void init (PortletConfig portletConfig) throws UnavailableException
{
    super.init( portletConfig );
    if ( getPortletLog().isDebugEnabled() ) {
        getPortletLog().debug("HelloWorld: init called");
    }
    // The default Hello String is obtained from the portlet configuration
parameters
    defaultString = portletConfig.getAttribute("defaultHelloString");
}
```

Creating your own logging mechanism with log4j

The portal's standard way to log from portlets can be hard to use, since all the important portlet logging is put in the same big file. It can be a time consuming and frustrating task to find out which entry came in from which portlet, as they all share the same log file.

With log4j (log for Java) you can build your own logging mechanism individually for each portlet. With log4j it is possible to enable logging at runtime without modifying the application binary. Furthermore, the log4j package is designed so that these statements can remain in shipped code without incurring a heavy performance cost.

Implementing this tailored logging is easy: you import the log4j.jar file into your project, add some coding, and set parameters, and you have a working logging mechanism. In the integration techniques section we give an example.

The target of the log output can be a file, an OutputStream, a java.io.Writer, a remote log4j server, a remote UNIX Syslog daemon, or even an NT Event logger, among many other output targets.

Tip: Logging behavior can be controlled by editing a configuration file, without touching the application binary.

One of the distinctive features of log4j is the notion of inheritance in loggers. Using a logger hierarchy, it is possible to control which log statements are output at arbitrarily fine granularity, and to do so with great ease. This helps reduce the volume of logged output and minimize the cost of logging.

Inserting log requests into the application code requires a fair amount of planning and effort. Observation shows that approximately 4 percent of code is dedicated to logging. Consequently, even moderately sized applications will have thousands of logging statements embedded within their code. Given their number, it becomes imperative to manage these log statements without the need to modify them manually.

The log4j environment is fully configurable programmatically. However, it is far more flexible to configure log4j using configuration files. Currently, configuration files can be written in XML or in Java properties (key=value) format.

Basics of log4j

The use of log4j revolves around 3 main concepts:

1. Public class *Logger*. Logger is responsible for handling the majority of log operations.
2. Public interface *Appender*. Appender is responsible for controlling the output of log operations.
3. Public abstract class *Layout*. Layout is responsible for formatting the output for Appender.

Logger

The logger is the core component of the logging process. In log4j, there are 5 normal levels of logger available, not including custom Levels.

As identified in the log4j API, the levels are the following:

- ▶ Static Level DEBUG: The DEBUG Level designates fine-grained informational events that are most useful to debug an application.
- ▶ Static Level INFO: The INFO level designates informational messages that highlight the progress of the application at a coarse-grained level.
- ▶ Static Level WARN: The WARN level designates potentially harmful situations.
- ▶ Static Level ERROR: The ERROR level designates error events that might still allow the application to continue running.
- ▶ Static Level FATAL: The FATAL level designates very severe error events that will presumably lead the application to abort.

In addition, there are two special levels of logging available:

- ▶ Static Level ALL: The ALL Level has the lowest possible rank and is intended to turn on all logging.
- ▶ Static Level OFF: The OFF Level has the highest possible rank and is intended to turn off logging.

The preceding descriptions are from the log4j API at:

<http://jakarta.apache.org/log4j/docs/api/index.html>

The behavior of loggers is hierarchical. Figure 5-18 illustrates this.

		Will Output Messages Of Level				
		DEBUG	INFO	ERROR	WARN	FATAL
Logger Level	DEBUG	Green	Green	Green	Green	Green
	INFO	Red	Green	Green	Green	Green
	ERROR	Red	Red	Green	Green	Green
	WARN	Red	Red	Red	Green	Green
	FATAL	Red	Red	Red	Red	Green
	ALL	Green	Green	Green	Green	Green
OFF	Red	Red	Red	Red	Red	

Figure 5-18 Log4j logging levels

A logger will only output messages that are of a level greater than or equal to it. If the level of a logger is not set it will inherit the level of the closest ancestor.

Appender

The public interface Appender controls how the logging is output. The Appenders are described in the log4j API. There are 10 established appenders, and you can also implement the Appender interface to create your own ways of outputting log statements. The list of appenders is found on the Apache Web site at:

<http://jakarta.apache.org/log4j/docs/api/index.html>

We used FileAppender to append log events to a file.

Layout

The Appender must have an associated Layout so it knows how to format the output. There are three types of Layout available:

1. HTMLLayout formats the output as a HTML table.
2. PatternLayout formats the output based on a conversion pattern specified, or if none is specified, the default conversion pattern.
3. SimpleLayout formats the output in a very simple manner: it prints the Level, then a dash '-' and then the log message.

We provide examples of using log4j in the next chapter.

Note: In addition to log4j, there are other technologies to accomplish this type of logging.

5.10 Struts Portal framework

Web applications have been evolving into complex interactions between the user and back-end systems. Without the right tools these applications can be complex to write and extremely difficult to maintain. The Struts framework has become a de facto standard in the industry for authoring Web applications. Because of this, enterprises want to use this framework on WebSphere Portal, both for migrating existing applications and for developing new ones. In this section we discuss adding support for Struts applications in the Portal server.

Why use the Struts framework

The Jakarta Struts framework is intended to provide a portlet application developer with two main advantages:

- ▶ A controlled logic to your portlet applications

As you have realized by now, the development of portlet applications requires the developer to coordinate several technologies to work together. In our examples we have seen that all requests are sent to the portlet class `doView()` method, and depending on the request and state of the portlets, it redirects to the appropriate JSP pages for rendering.

This is accomplished with a redirection map embedded in the portlet class. Basically, it's a tightly coupled application, since any change on the control logic would require recompiling and making sure that your application doesn't lose its integrity. The Struts framework solves this problem, especially in big and complex portlet applications, since it takes out the control logic map from the portlet class and places it on a flexible deployment descriptor, giving you a loosely coupled structure. Also, this map can be edited with graphical tools like the ones included in WebSphere Studio, Struts tools.

So the developer should consider Struts Portal framework if he is placing control logic into the portlet applications and requires a flexible and manageable control logic for the portlet.

- ▶ A strong foundation for portlet development

The Struts framework incorporates proven development patterns, like the MVC model 2, that will make your portlet application perform and scale appropriately. So if you are considering introducing a hybrid approach in the development of your portlets, and you want to provide a strong structure, Struts can give you exactly what you need.

The Struts framework will add some development components that have to be well understood and properly used in order to exploit its full advantages. So if your portlet application will just expose the Domino control logic of your applications, it would be appropriate to reconsider using this framework.

Also bear in mind that the standard Struts framework is accommodated to work in the WebSphere Portal infrastructure through the Struts Portal framework, so in this section we discuss the considerations for incorporating the Struts technology into your portlet application.

Struts defined

Struts is a Jakarta open source project that provides a framework based on the Model-View-Controller (MVC) pattern, which allows developers to efficiently implement their Web applications, keeping the business logic and presentation aspects separate. The developer community has embraced Struts, and the initial release is continually being enhanced. In addition to enhancements to the base Struts functionality, there are numerous extensions and tools available, for example, the Struts editor in WebSphere Studio. In the Portal environment, the ability to use Struts is a logical extension. The portlets on a Portal page can be essentially thought of as servlet applications in their own right. Thus, for many of the same reasons one would use Struts to implement a Web application, one would also like to be able to use Struts in a portlet.

Basics elements of Struts

A Struts application is basically made up of the following elements:

- ▶ **Actions**
The Actions represent processing that occurs prior to going to another page.
- ▶ **Pages**
These are usually JSPs, but sometimes are HTML pages.
- ▶ **Action Form Beans**
ActionForms are Beans associated with Actions, supplying the data to fill or collect various fields on a page.

The application writer creates these objects and, in the configuration file `struts-config.xml`, defines the relationships between these objects and the transitions that occur. The configuration of an `ActionMapping` associates a path with an Action and with ActionForms, and can list one or more destination pages following execution of the Action. As a result of executing the Action, an `ActionForward` object is returned which contains the path to the next request. Typically, the returned path is to a page, but it is also possible that the path is to another Action. Figure 5-19 depicts the normal structure of a Struts application.

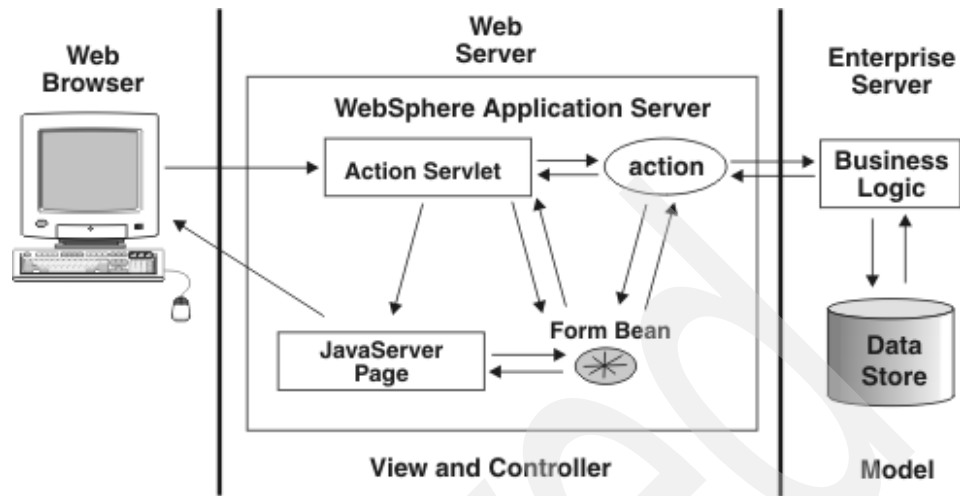


Figure 5-19 Struts structure

Struts 1.1

The design of Struts has evolved into a more modular implementation in version 1.1. There are two main new features in Struts 1.1:

- ▶ The addition of the Request Processor

The processing of a request has been moved out of the Action Servlet into the RequestProcessor class. The Request Processor is split into numerous process methods that can be overridden in a derived class to control the processing. The new modular implementation facilitates the support of Struts in the Portal server by allowing Request Processor methods to be extended as necessary. The modular implementation limits the number of changes necessary to the base Struts itself.

- ▶ Enhanced sub-application support

Struts allows configurations that are prefixed, which allows different configurations based on prefixes. Portal server makes use of the prefixes to support Portal server's modes and device support.

Struts in a Portal server

A Struts application in Portal server is similar to the servlet-based Struts application. A WAR file that contains the Struts jars, the JSPs, actions, and configuration is built. The WAR file in Portal server has some additional requirements. There are some additional jar files, and a portlet descriptor file. There are also some necessary changes to the Web deployment descriptor and the Struts configuration.

To support existing Struts applications and offer a migration path to the Portal server for existing Struts applications, several differences between Struts servlet-based applications and portlets exist, specifically:

- ▶ The portlet itself is a servlet and has its own processing and servlet mapping. The portlet servlet must be configured as the main servlet rather than the Struts Action Servlet.
- ▶ Portlet action processing (for example, handling of user events like selecting links or submitting forms) is done in a step prior to rendering the display view. During this processing the response object is not available.
- ▶ Display view rendering is done separately from action processing, and may be done multiple times without intervening actions. This occurs because all portlets on a page are essentially asked to refresh, or re-render, themselves when the entire Portal server page is refreshed.
- ▶ URIs under Portal server have a special format. A special API is used to generate the URI and add the desired information to be passed to the portlet. If portlet URIs were not used, control would not get passed to the correct portlet. Thus, the portlet URIs must be used to get control passed to the correct portlet, with the additional path information needed by the Struts application made available. The Struts tags have been modified to automatically provide this needed functionality.
- ▶ Portal server does not support forwards or redirects. Either alternatives have to be used, or the functionality emulated.

The differences enumerated result in the following conclusions with regard to supporting Struts applications as portlets:

- ▶ The portlet's servlet mapping has to be used. Routing of Struts actions back to the Struts application processing has to be done using portlet URIs. The way the Struts mapping is configured is described later in this section.
- ▶ The processing of Struts Actions must occur during portlet action processing, and the information necessary to render a display view must be stored away so it is available during the later view rendering phase, which may occur multiple times without intervening actions.
- ▶ The Struts application paths, both to Actions and to pages, must be sent and retrieved in a different way.
- ▶ Forwards to pages (for example, JSPs) are done with include operations rather than with forward operations. Forwards to other Actions can be handled by recursively executing the Struts Action processing. Redirects are treated in the same way as a forward.

Two-phase processing

A portlet has a different processing and rendering design than a servlet. A servlet does all of its processing in the service method. A portlet, on the other hand, uses two-phase processing that is split between an action processing and service. The reason that a Portal server needs this split is because portlets may need to communicate with other portlets before the rendering stage. The action processing is guaranteed to complete before a portlet is asked for a rendering.

The processing of events by a portlet is handled in a separate, earlier step from the rendering step. The typical action handling is achieved by implementing the `ActionListener` interface. The `ActionListener` interface provides the `actionPerformed` method, to which an `ActionEvent` object is passed. When a user clicks on a link or a submit button, an `ActionEvent` can be generated. The `PortletAction` can be obtained from the `ActionEvent`, which describes the triggering event. When the `actionPerformed` method is invoked, a response object is not available because this is not a rendering step.

The rendering step is handled in a separate method, the service method, which calls the `doView` method in the `PortletAdapter`. The `doView` method is not only called following the `actionPerformed` processing when a user clicks on a link or button in a portlet, but is also called when the portal page is refreshed. Thus, given a page with two portlets, A and B, when the user clicks on a link in portlet A, `actionPerformed` and `doView` is called for portlet A, but only the `doView` method is called for portlet B.

The two-phase processing causes some significant challenges as it relates to Struts applications.

- ▶ The methods called during Struts processing expect both a request and response to work with.

The first issue of the lack of a response object during `actionPerformed` processing can be handled by creating a pseudo response. The response object used prior to actually invoking the JSP is not heavily used. However, during action processing within a Struts application, it is not unusual for an application to report an error using the `sendError` method.

- ▶ The Struts rendering of the page is usually immediately preceded by action processing; they are essentially part of one step.

The second issue is the fact that the rendering step is separate from the event handling step. Note that some of the information that was available during the processing of the `actionPerformed` method, namely the request parameters, is no longer available when the `doView` method is invoked. Additionally, since `doView` can be called when the portlet page is refreshed (without a new event occurring for that portlet), all information required to render the page must be available every time `doView` is called. Since normal Struts processing allows

the updating of the response object during almost all steps, including during action processing, there are clearly situations where the two-phase model would cause problems in the application. However, a typical well-behaved Struts application does not write directly to the response object and instead forwards to a JSP. It is assumed that the JSPs do not themselves need access to the original event information, request parameters, which initially led to the current rendering. Instead, it is assumed that the Action processing stores information in a bean, from which the JSP extracts the information. As long as those beans are available during the rendering, the rendering can be repeated. Since those beans, most notably the ActionForm, are usually stored in the original request attributes, the request attributes from the original request processing must be saved and made available when doView is invoked.

No response object

The typical well-behaved Struts application has no need to access the response object during the Action processing. The lone exception encountered is the need to report an error through the sendError method. When an application checks and finds some state information invalid, it is common to use response.sendError to generate an error page. Failing to support this would break a large number of legacy Struts applications. The Struts portlet framework intercepts the calls to sendError and saves the error information for display at a later time. During the later view rendering phase, this error information is found, and the error information displayed instead of displaying other content for the portlet.

The Struts URL paths

Struts Action mappings are defined in terms of paths. Also, the name and location of page objects (for example, JSPs) are defined via paths as well. Thus, although portlets have their own form of URI, it is still necessary to be able to associate the Struts path with an action sent to a portlet and to retrieve that Struts path when the portlet action is handled. It is not unusual to pass parameters on such a path via the query string on the HTTP URL. It is also important to realize that often the actions containing these paths are generated from tags provided by Struts. The most obvious examples of these are the tags for the HTML elements LINK and FORM. Obviously, in order to support such tags when the Struts Application is executed in a portlet, they have to be modified to create a portletURI with the required additional information.

The Struts tags that create links have been modified to pass the created paths directly through to the Portal server as a parameter on the portlet URI. This allows a portlet URI to be created that will cause the interaction with the portlet.

Forwards and redirects

As mentioned earlier, the Struts Action objects return an ActionForward object, which contains a path to be either forwarded or redirected to. Portal server does not support forward operations because the response object has already been committed. If the path is not an action, the page is included instead. If the path is an action, then a forward to the Action is simulated. The include of a page uses the PortletContext include method.

A forward for an Action is simulated by recursively sending the new Action URI through the Struts base processing of an Action.

Note that not only can we have this issue of forwards in ActionForward objects returned from Action objects, but also in tag handlers as well. In tag handlers, it is possible to access the PageContext object and invoke the forward method as well. An alternative method to the PageContext.forward is available via the PortletApiUtils class, which provides a mechanism to emulate the forward functionality.

It is important to understand that although you can write portlets using Struts, there are some things that you can do in a Struts servlet that you cannot do in a Struts portlet. For example, Struts provides support for things like redirects and forwards to other servlets. These are provided because they are functions generally available to servlets. However, these capabilities are not available in portlets.

What not to do in an action

Following are two examples of what you shouldn't do in an action:

- ▶ The typical use of the response object is to call sendError when an error condition is detected. Portal server does not support writing to the response object during the action processing. Therefore, a response object is not available, as discussed previously. A pseudo response object is made available during the processing in the Request Processor. If the Struts Action writes to this response object, that text will be lost. The Action should return an ActionForward object. This is important so the Request Processor goes through the normal processing.
- ▶ The ForwardAction and IncludeAction, normally shipped with Struts, are examples of what should not be done in a Struts application in Portal server. These actions create a RequestDispatcher and try to forward or include the path in the Action execute. The ForwardAction should return an ActionForward so the RequestProcessor can process the Action. The Struts portlet framework provides its own versions of ForwardAction and IncludeAction to deal with this problem. It is hoped that the issue will be corrected in the base Struts implementation sometime in the future. However, as mentioned previously, the use of the sendError function on the response

object is supported by the Struts portlet framework because the function is so commonly used.

Roles support

Struts uses *roles* to determine access to an Action. Portal server does not support roles at this time. The Struts portlet can be configured to use group names as if they were role names, in addition to group names, to emulate role-based access control. The following init parameter in the Web deployment descriptor enables this support:

```
<init-param>
<param-name> UseGroupsForAccess</param-name>
<param-value>true</param-value>
</init-param>
```

Developing Struts applications with WebSphere Studio

The Struts application development tools that are provided by WebSphere Studio make it easy for you to build and manage a Struts-based Web application. WebSphere Studio provides the following features:

- ▶ Creation of a Struts project with the tag libraries and related resources correctly referenced and ready to use.
- ▶ Wizards to develop and create Struts elements like form beans and action classes, giving you a head start in Struts development.
- ▶ A specialized editor to create and modify the Struts XML configuration file.
- ▶ Struts support for validation and editing; for example, by helping you to use Struts tag libraries.
- ▶ A visual assembly tool, which provides the following capabilities:
 - Helps architects to design the flow of a Struts-based Web application and to communicate that design to other professionals
 - Embeds Struts code in several components
 - Provides quick access to resource-appropriate editors and wizards
 - Separates team responsibilities for greater productivity and focus

As shown in Figure 5-20, the editor will let you connect and visually include Struts components seamlessly.

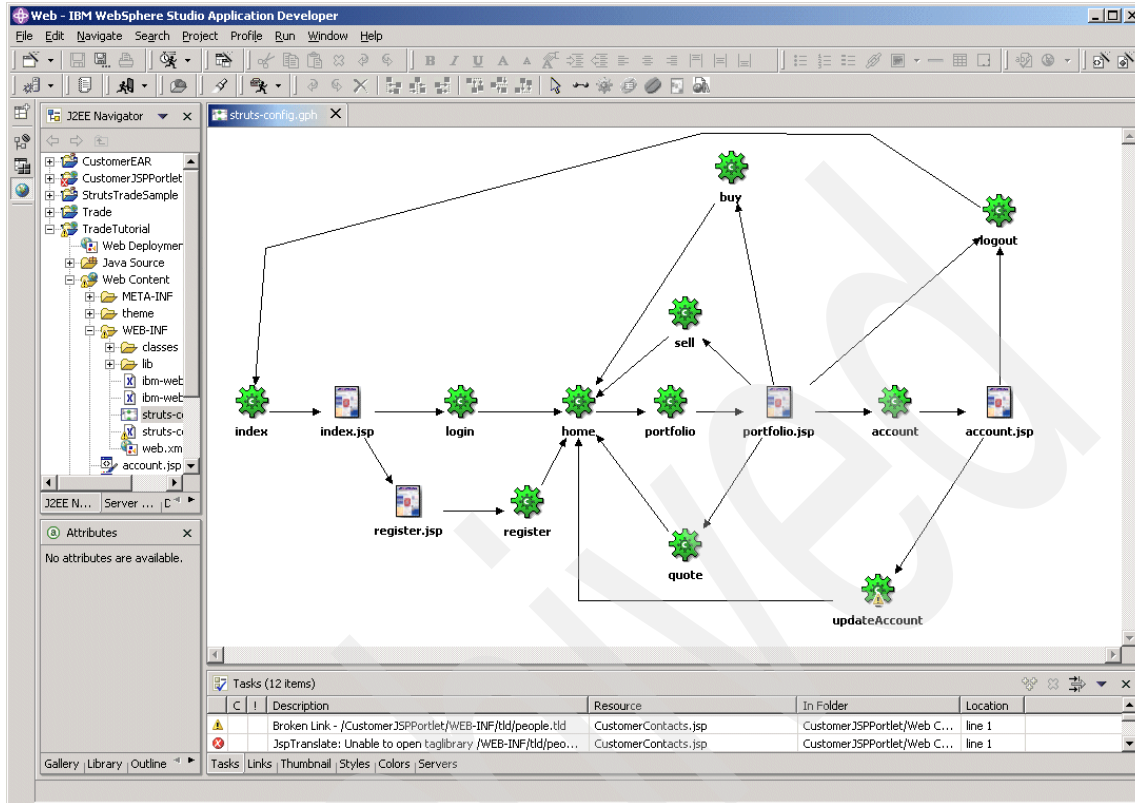


Figure 5-20 WebSphere Studio Struts sample diagram

5.11 General portlet development guidelines

The following general portlet performance thoughts have been taken directly from the Portlet Developer's Best Practice and Coding Guidelines. We consider performance crucial to any production portlet, therefore we have included a copy of this information here.

Portlet coding guidelines

These guidelines are intended to help you produce best-of-breed portlets for the WebSphere Portal environment.

Refrain from using instance variables. Portlets, like servlets, exist as a singleton instance within the server's JVM. Therefore, a single memory image of the portlet services all requests and must be thread-safe. Data stored in instance variables will be accessible across requests and across users and can collide

with other requests. Data must be passed to internal methods as parameters. There are other means of storing data globally.

Pass data to the view (JSP) as a bean in the request object. Use the `PortletRequest` object to pass data to the view for rendering. This way, when the request is complete, the data falls out of scope and is cleaned up. Passing it as a bean allows the JSP to simply refer to the data as properties on the bean using intrinsic functions in the JSP syntax.

Use the portlet logging facility. Using the `PortletTraceLogger` installed with WebSphere Portal for tracing and logging allows your portlet's debug and trace output to be stored with other portlet output in the portal log files. It also takes care of the infrastructure around logging and tracing. The `PortletLog` object that is the interface to the logging mechanism can be obtained from the portlet context:

```
PortletLog log = getPortletConfig().getContext().getLog();
```

You should also verify that logging is enabled for attempting to log data, especially if that requires additional logic or storage to build the log message.

Example 5-11 Using a log from portlet

```
if (log.isDebugEnabled()) {  
    String logMsg = new String("Time to retrieve data: " + elapsedTime);  
    log.debug(logMsg); }  
}
```

Adopt good code documentation habits. While commenting of code is not required for the functionality of the portlet, it is essential for its maintenance. In addition to the fact that the responsibility for a portlet's maintenance can change hands over time, well-written portlets serve as models for other portlets, which means that someone else must understand what you wrote. Well documented code implies more than simply inserting comments, it implies good naming practices for Java resources as well. The following are examples of guidelines to follow:

- ▶ Insert JavaDoc-compliant prologues for all public classes, variables, and methods. Be sure to document inputs and outputs.
- ▶ Include inline comments, but do not include them on the same line as Java source. It is difficult to align and follow comments which are on the same line as Java source.
- ▶ Use meaningful names for variables and methods. Variables `x`, `y`, `z`, while easy to type, are not easy to follow through code. Capitalization conventions and the use of underscores (`'_'`) within resource names is a matter of personal choice, but be consistent once your choice is made.

Use the ContentAccessService to fetch external content, when necessary.

If it is necessary to fetch external content using an HTTP request, use the ContentAccessService as it is intended to perform that function as expediently as possible. It also prevents you from having to rewrite the same function and introduce another maintenance point.

Cache portlet settings or portlet data. If portlet settings or data is complicated and requires complex parsing to digest, it may be a good idea to cache the data for quick retrieval later. Avoid using the PortletSession to store this data as it causes “session bloat”. Alternatively, consider using the Application Server’s Dynacache CommandCache to store this data. PortletSettings can be stored keyed off the portlet’s name. PortletData will have to be stored keyed off the portlet name and user name. When updates to the cache are necessary, based on updates to PortletData or PortletSettings, devise a means of communicating updates to the data, such as through implementing the PortletSettingsAttributesListener interface, or by using a “dirty bit” that can be interrogated to see if updates have been made to the configuration, in which case the cache can be invalidated and updated.

Note that the CommandCache is not currently cluster-aware, meaning that cached content is not shared across horizontal nodes in a cluster. So, portlets must be able to create the cache entry from scratch if the cache entry does not exist.

Follow design guidelines for Struts portlets.

If you are developing portlets based on Struts, use the following additional guidelines to ensure your implementation is the best it can be:

- ▶ Be sure to use the sample portlet applications that accompany the Struts Portal Framework package to ensure Struts is working properly in your environment before testing Struts support in your portlet.
- ▶ Make sure to review the documentation on the Struts Portal Framework for application requirements and any restrictions. For existing Struts applications refer to the section in the same document titled “Migrating an Existing Struts Application”.

Portlet packaging guidelines

Make common functions available externally to portlets. If the portlet contains common functions that are replicated across several portlets, consider isolating them making them externally accessible by the portlets. The easiest way to do this is build another JAR file of the common classes and place the JAR file in a location that is in each portlet’s classpath, such as the AppServer/lib/app directory. Another approach is to build a Portlet Service from the common

functions which can then be accessed using the common PortletService interface, retrieved using the PortletContext.getService() method.

Combine portlets with similar functions into a single portlet with multiple configuration settings. If you find yourself developing several similar portlets, such as a mortgage calculator portlet, car loan calculator portlet, and a home equity loan calculator, it might make sense to actually develop a single portlet that can behave differently based on configuration settings. The common portlet code is specified as part of the portlet application's abstract portlet definition. The application's various concrete portlet definitions have unique configuration parameters (PortletSettings) that can be used to customize the behavior of the common portlet code.

Data management

There are three primary areas where portlet configuration information is stored and can be maintained: portlet settings, portlet data, and servlet configuration.

- ▶ **Use portlet settings to store user-independent configuration data.** This data represents configuration parameters from the portlet's portlet.xml file (<config-param> elements under the concrete portlet definitions). It is only writable in the portlet's configure mode (doConfigure() method), but is readable through the getPortletSettings() method on the portlet.
- ▶ **Use portlet data to store user-dependent configuration data.** This data often represents personalized overrides to the portlet settings and is thus stored according to user and portlet instance. This data is only writable from a portlet's edit mode (doEdit() method), but is readable through the getPortletData() method on the PortletRequest object.
- ▶ **Use servlet configuration for storing static initialization information for a portlet.** Examples of this information include the install directory or path information. This data, taken from the portlet's web.xml file, is read only and accessible using the getServletConfig() method on the portlet.

General session management guidelines

Limit the use of the portlet session for storing portlet state information. The Portlet Session is a convenient place to store global data that is user and portlet specific and that spans portlet requests. However, there is considerable overhead in managing the session, both in CPU cycles as well as heap consumption. Since sessions may not expire for quite some time, the data contained in the sessions will remain resident in active memory even after the user is done with the portlet. Be very judicious about what is stored in the Portlet Session.

Tip: A good rule of thumb to use when determining if data should be stored in the Portlet Session is if the *data is user-specific and cannot be recreated* by any other means, such as portlet state information.

For example, parsed configuration data (from PortletSettings) should not be stored in the Portlet Session since it can be recreated from the database at any time.

Do not rely on portlet sessions if the portlet is to allow anonymous access.

If your portlet is to be used by unauthenticated users, such as on the Welcome page, then the portlet should not rely on the portlet session at all. Portlet sessions are bound to a user context. By default, for unauthenticated users, a portlet session can be requested by a portlet, but it is a temporary session which only lasts for the duration of the current request. Public session support exists, where sessions are issued to unauthenticated users, but because there is no log out action for an unauthenticated user, the session will not be invalidated until it times out, which can lead to severe memory consumption issues.

There are other ways to preserve global data for anonymous access, such as:

- ▶ Storing data in a collection whose key is communicated back to the client using a cookie.
- ▶ Storing data in the WebSphere Application Server's dynacache facility, again using a unique key which is communicated to the browser using a cookie.
- ▶ Storing the state information as a cookie itself.
- ▶ Storing non-sensitive data as hidden field elements within FORMs so that it gets passed back on subsequent form submission. This has the added advantage of binding state-aware data to the page requiring it.

Alternatively, consider limiting the function of the portlet for anonymous access. You can check for the presence of a user object (PortletRequest.getUser()) to see if a user has logged into the portal, in which case a portlet session should be available.

Always request an existing portlet session. Always request a portlet session using either PortletRequest.getPortletSession() or PortletRequest.getPortletSession(false), which will only return a session if one does not already exist. The portal server will establish a portlet session for a portlet before it is invoked. This helps prevent the case where a temporary session is generated for a portlet during anonymous (unauthenticated) access.

Prevent temporary sessions from being generated in the JSP: Add the JSP page directive `<%@ page session="false" %>` to the JSP to prevent temporary sessions from being created by the JSP compiler if none already exist. This will

help guard against attempting to use the session to store global data if the session will not exist past the current request. You will need to be sure the portletSession exists before trying to use it.

Other general development guidelines

There are potentially many different types of browsers, or user agents, which access WebSphere Portal besides a typical desktop browser, such as hand-held devices (PDAs, or personal data assistants) and wireless phones. The capabilities of these devices vary widely, as do their users' attention spans and interaction models. Therefore, the portlet's design needs to take these facts into consideration.

Information priority is different depending on the device it is viewed on. Capabilities of desktop browsers are limitless. Users can download almost anything quickly and randomly navigate through it. For handheld devices, however, screen real estate and device memory is at a premium, not to mention the relatively slow speed of a wireless connection versus a LAN connection. Also, users of mobile and handheld devices need quick access to concise information and cannot afford the time or effort it takes to navigate through several screens to get to the desired information. Here are some more general guidelines for portlet development:

All portlet strings should be fetched from resource bundles. All displayable strings should be stored in resource bundles and fetched using the ResourceBundle Class. The resource bundles should be organized by language under the portlet's WEB-INF/classes directory.

For example, if a portlet supports English and Finnish Language, it would have three resource bundles (default, English, and Finnish) and might be organized under WEB-INF/classes as so:

```
WEB-INF/classes/nls/mystrings.properties  
WEB-INF/classes/nls/mystrings_en.properties  
WEB-INF/classes/nls/mystrings_fi.properties
```

Using the ResourceBundle class, you would refer to the properties file as "nls.mystrings". Java will automatically look for the appropriate properties file based on the current locale

Organize portlet help files by language. As opposed to typical JSPs in the portlet's view, the portlet's help JSP files can, and should be language dependent. Since little else in the JSP will exist but help content, there isn't much need in extracting strings from resource bundles. Translate the JSPs instead, and organize them by language. When using the include() method on the portlet context to dispatch a JSP, the method will search the directory structure for

locale-specific subdirectories first. For example, if you include the JSP “/WEB-INF/helpjsps/html/myHelp.jsp”, and the current request locale is en_US, then the method will look for the myView.jsp in the following directories, in order:

/WEB-INF/helpjsps/html/en_US/myHelp.jsp

/WEB-INF/helpjsps/html/en/myHelp.jsp

/WEB-INF/helpjsps/html/myHelp.jsp

/WEB-INF/helpjsps/myHelp.jsp

Note that the include() method will also look for a default myHelp.jsp if one does not exist in the HTML subdirectory.

Use portlet messaging to communicate (send messages) to other portlets on the same page.

It is possible to send messages to a specific portlet or to all portlets on a page. The message body may contain data that the receiving portlets require. The message is sent from a portlet’s actionPerformed() method and received by other portlet’s messageReceived() methods, before those portlet’s doView() is called as a result of the action, so there is an opportunity to set data or state information before the receiving portlets are invoked. See the section titled Developing portlets → Writing portlets → Portlet messaging in the WebSphere Portal InfoCenter for more details.

Avoid the use the HttpSession to share data with other portlets/servlets.

The HttpSession is not common between nodes in a cluster, so any data store in the HttpSession is only visible by J2EE resources in the same Application Server instance. The PortletSession is derived from the HttpSession, and besides only having visibility of the portlet instance, is also bound by the same restriction. Persistent sessions can be turned on in the application server, which will serialize the session to a database which can be shared across nodes in a cluster, but that can cause performance issues for very large sessions. See Session Management guidelines section for other ideas.

Avoid the use the HttpSession to share data with other portlets/servlets.

The HttpSession is not common between nodes in a cluster, so any data store in the HttpSession is only visible by J2EE resources in the same Application Server instance. The PortletSession is derived from the HttpSession, and besides only having visibility of the portlet instance, is also bound by the same restriction. Persistent sessions can be turned on in the application server, which will serialize the session to a database which can be shared across nodes in a cluster, but that can cause performance issues for very large sessions. See Session Management section for other ideas.

Performance considerations guidelines

Do not spawn threads. Since portlets are multi-threaded to begin with, spawning child threads can create problems with synchronization or multi-threaded access to the spawned threads. Threads should be used with extreme caution, and when necessary, should be used briefly (no long running threads, especially any that outlast a request).

Do not use threads to access J2EE resources. In certain Java Virtual Machine (JVM) implementations, such as on zOS, there are a limited number of threads in the process thread pool which are allocated for accessing J2EE resources, such as JCA connectors. All such resources should be manipulated on the calling thread to minimize the possibility of starving the thread pool.

Limit temporary storage. Temporary storage includes temporary variables created within a block of code which are used then discarded. Temporary variables are typically utilitarian in nature, such as Strings, integers, booleans, Vectors, and such. However simple in nature, temporary variables take CPU cycles to create and destroy and occupy space on the heap which must be maintained by the garbage collector. The following is a list of simple guidelines for reducing or optimizing temporary storage:

- ▶ Reuse temporary variables as much as possible
- ▶ Declare temporary variables outside loops
- ▶ Instead of String, use StringBuffer, which are optimized for parsing and string assembly
- ▶ Declare collection classes (Vectors, Arrays) with an initial size that is the average maximum size, to prevent growth and reallocation of space on the heap.

Avoid synchronized methods. Synchronization causes a bottleneck through your portlet, or a path that only allows one thread at a time to pass through. Besides slowing the entire portal system, the possibility of a deadlock occurring is also high, which could hang the portlet, or portal, for all users.

Avoid long-running loops. Simply put, portlets need to be fast. Long running loops consume a large number of CPU cycles and cause the portal page to wait for this one portlet to finish. If a long-running loop seems necessary, re-inspect the design to see if it can be accomplished by some other means, such as through block/notification logic, or breaking the large loop up into several shorter ones.

Use JSPs instead of XML/XSLT. JSP are more efficient than XML/XSLT in general. Since JSPs are compiled into servlets, they run much faster than having to parse XML and apply XSL stylesheets on every request. Also, the portal infrastructure is optimized around JSPs, allowing for easy expansion into other

markups, languages, and even browser support by simply adding new directory structures of JSPs. In general, XML parsing and XSLT processing is expensive. XSL processing, for example, causes hundreds of transient String objects to be created and destroyed, which is expensive in CPU cycles and heap management. If XML/XSLT is a requirement, then make use of caching as much as possible to reduce the amount of parsing and XSLT processing which must take place.

Use caching as much as possible. If portlet output is static in nature or is valid for some period of time before it is updated, the portlet should enable the “display cache” by setting appropriate values for the <cache> elements in the portlet’s portlet.xml file (note that caching must be enabled in a portlet before it is installed or updated). A portlet’s caching settings can be set such that the output is never cached (<expires> value of 0, the default), cached for some period of time (<expires> value greater than 0, in seconds), or cached indefinitely (<expires> value of -1). The cached entries can also be shared across all users of the same portlet (<shared> value of YES) if the output is valid for any user.

Important: Using the cache vastly improves the performance of the portlet, and thus of the page in general.

The portlet can also implement the `getLastModified()` method to control the invalidation of the cached content, informing the portal which it should call the portlet’s `doView()` method to refresh the output. Refer to the portlet development section of the WebSphere Portal InfoCenter for examples of how to use the `getLastModified()` method.

Archived



Portlet development using Java: Integration examples

In this chapter we show you how to put to use the Java integration techniques introduced in the previous chapter. We do this by providing step-by-step examples of adding functionality to the sample Sales Tracking application that we first introduced in Chapter 2.

First we discuss hybrid techniques used in the two samples where we add functionality for searching and pagination in portlets.

Then we discuss full Java integration techniques, from the simple “HelloWorldFromDominoServer” portlet, to using JavaBeans in a portlet, and creating an ACL browsing portlet for Domino. We also demonstrate how to use logging (log4j) in a portlet and how to get started with Object pooling.

6.1 Software and tools used

We prepared the samples in this chapter using WebSphere Studio Application Developer version 5, and Portal Toolkit version 4.2.5. Both were introduced in earlier chapters. In addition, we used Domino Toolkit for Java version 2.1, which is described in the following section.

6.1.1 Domino Toolkit for Java version 2.1

The Lotus Domino Toolkit for Java/CORBA, commonly referred to as “the Java/CORBA toolkit,” is a comprehensive source of samples, documentation, and software that helps you in your Java development work.

The overall goal of the Java/CORBA toolkit is to help you leverage the power of Java, Domino, CORBA, Notes, and Portlets. This toolkit is suitable if you are developing:

- ▶ Java programs, such as portlets, that use Domino Objects for Java to access local and remote Domino data and services
- ▶ Java programs that access data in relational database management systems
- ▶ Java and C++ programs that use the Common Object Request Broker Architecture (CORBA), the Internet Inter-Operability Protocol (IIOP), and the Domino Interface Definition Language (IDL) to access remote Domino objects

It provides a range of sample code you can use to create your own Java programs. However, it does not provide sample code based on portlets because this toolkit was released before WebSphere Portal server existed. Nevertheless, it is very helpful for portlet development work.

Important: Domino Toolkit for Java includes the Domino Objects for Java libraries (Notes.jar and NCSO.jar) that let you develop local and remote Java programs. NCSO.jar lets you develop remote Portlets without necessarily having Domino or Notes installed.

You can download this toolkit from:

<http://www-10.lotus.com/ldd/toolkits>

6.1.2 Hybrid integration techniques

The hybrid integration techniques will start where the JSP option left off, and introduce some additional functionality with Java components. The functionality

we add will focus on the Customer List portlet, where we incorporate the following features:

- ▶ Search for customers
- ▶ Paging capabilities

We will end up with a portlet that can manage long views since it offers positioning and filtering capabilities, while retaining the previous enhancements like Click to Action and People awareness.

Figure 6-1 shows the different portlet modes in the Portal UI and how the portlets operate. The figure also includes the .java package and actions inside that Java file.

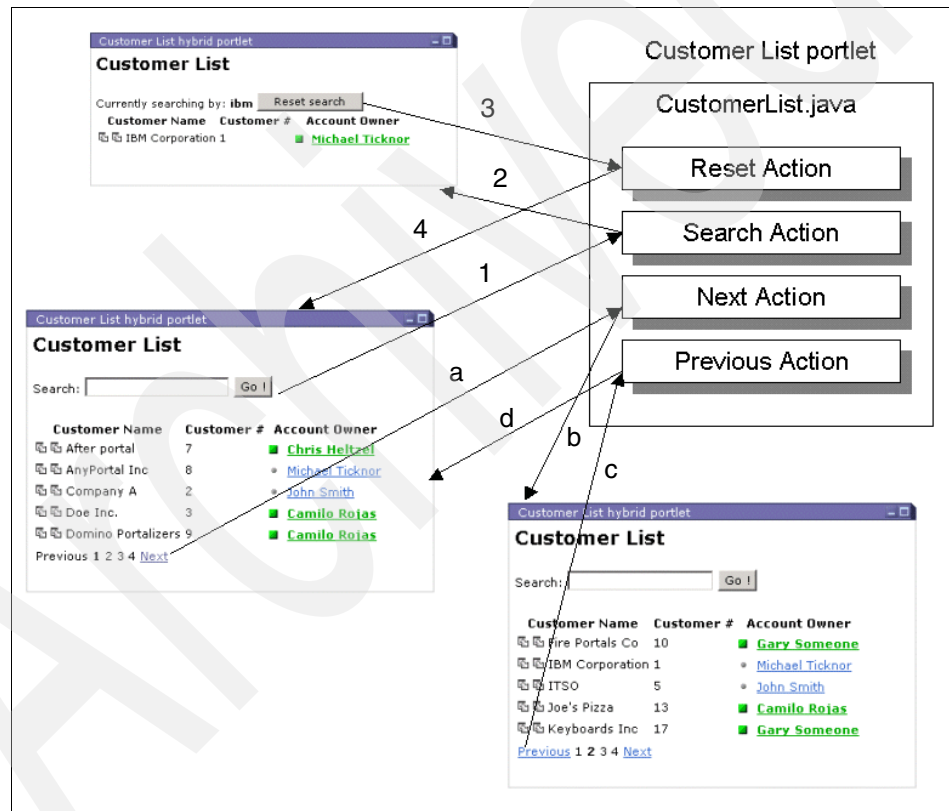


Figure 6-1 Hybrid JSP: Java Customer List portlet

6.2 Search functionality

To add search functionality to your portlet you will rely on Domino's excellent full text search capability. To use this capability, you need to first have a full text index of the Customers database. We assume you have enabled full text index in the Domino database used in this example.

First, let's analyze the life cycle of the portlet. It will support the actions identified in Table 6-1.

Table 6-1 Actions supported by the Customer List portlet with search

Action name	Description
search	This action responds when a user inserts a search string and clicks the button. The <code>actionPerformed()</code> method inserts the search string and adds it to the Session object. When rendering is being performed by the JSP, it checks to find out if the parameter is in Session and introduces an <code>ftsearch</code> attribute to the <code><Domino:view></code> custom tag. And it presents the view filtered.
reset	This action responds when a search statement filtering the view is present and the reset button is pressed. On the <code>actionPerformed()</code> method it removes the search string attribute from the Session, so that when it renders it again, there will be no filter present.

Use the following steps to add search functionality:

1. Open the WebSphere Studio Application Developer and open the CustomerJSPProject created on the previous chapter.

We focus on 2 components: the `CustomerList.java` portlet class and the `/jsp/CustomerList/View.jsp` file.

2. Open the `CustomerList.java` portlet class.
3. Insert the following action URIs in the `doView()` method:

```
PortletURI searchURI = response.createURI();
DefaultPortletAction searchAction = new DefaultPortletAction("search");
searchURI.addAction(searchAction);
request.setAttribute("search", searchURI.toString());
```

```
PortletURI resetURI = response.createURI();
DefaultPortletAction resetAction = new DefaultPortletAction("reset");
resetURI.addAction(resetAction);
request.setAttribute("reset", resetURI.toString());
```

4. Implement an `ActionListener` interface.
5. Add an empty `actionPerformed()` method.

6. Include in the `actionPerformed()` method the following logic to handle each event:

```
public void actionPerformed(ActionEvent event) throws PortletException {
    DefaultPortletAction action=(DefaultPortletAction)event.getAction();
    PortletRequest request=event.getRequest();
    if(action!=null){
        if(action.getName().equals("search")) {
            request.getPortletSession().setAttribute(
                "search",request.getParameter("search"));
        }
        if(action.getName().equals("reset")) {
            request.getPortletSession().removeAttribute("search");
        }
    }
}
```

This code will create the new attribute when the search event is called and remove it when the reset event is called.

7. Open the `/jsp/CustomerList/View.jsp` file.
8. Include the following code just below the Customer List label of the portlet:

```
<% String tempSearchString=null;
    if(request.getSession().getAttribute("search") != null){
        tempSearchString = "FIELD CustomerName contains "
            +request.getSession().getAttribute("search"); %>
    Currently searching by: <b>
        <%=request.getSession().getAttribute("search")%></b>
    <INPUT type="button" name="reset" value="Reset search"
        onClick="window.location.href='
        <%= (String)request.getAttribute("reset")%>' ">
    <br>
    <% } else { %>
    <form name="<portletAPI:encodeNamespace value="search"/>"
        action='<%= (String)request.getAttribute("search")%>'
        method="post">
        Search: <INPUT type="text" name="search" value="">
        <input type="submit" name="Search" value="Go !">
    </form>
    <% }%>
```

This scriptlet, based on the search attribute stored on the session, creates either a form to perform the search (if there is no attribute) or a button to reset the search and display the query.

9. To filter the Domino view add an attribute on the `<Domino:view>` tag with the name `ftsearch`. The tag should be like the following:

```
<Domino:view
viewname="CustomersByName"
```

```

dbserver="<servername example:CN=itsotest-dom/0=itsportal>"
dbname="<database name example :apps/customer.nsf>"
user="*webuser"
host="itsotest-dom"
ftsearch="<%=tempSearchString%>"

```

The portlet after you have enabled the search mechanism should look similar to Figure 6-2.

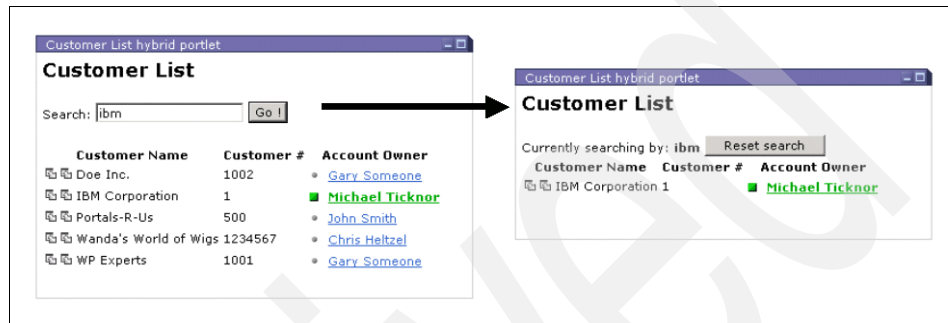


Figure 6-2 Customer List portlet with search capabilities

6.3 Paging through the view

Now let's consider that you have added some extra customers to the CustomerByName view and you can't present them all since the presentation of your portlet will be ruined. You can't include the Domino custom JSP tags that would handle the pagination because of the JavaScript code that gets inserted. Instead, you can write some Java code to manage paging. Again, you will work with the Customer List portlet.

At a high level, the process is: enable the portlet to accept two new actions, called previous and next; define two required variables (number of rows per page and counter for paging); and, on the actionPerformed() method handle the previous and next actions by increasing or decreasing the counter.

Use the following detailed steps to implement this functionality:

1. Open the CustomerList.java portlet file.
2. Insert the following actions in the doView() method:

```

PortletURI previousURI = response.createURI();
DefaultPortletAction previousAction = new DefaultPortletAction("previous");
previousURI.addAction(previousAction);
request.setAttribute("pageNo", previousURI.toString());

```



```

PortletURI nextURI = response.createURI();
DefaultPortletAction nextAction = new DefaultPortletAction("next");
nextURI.addAction(nextAction);
request.setAttribute("pageNo", nextURI.toString());

```

3. Include the following code at the beginning of the `doView()` method:

```

if(request.getPortletSession().getAttribute("pageNo")==null){
    request.getPortletSession().setAttribute("rows", "5");
    request.getPortletSession().setAttribute("pageNo", "1");
}

```

This code will create two attributes on the initial rendering: number of rows and a counter page. Both will be stored on the session.

4. In the `actionPerformed()` method add the handlers of the two newly created actions by inserting the following code:

```

if(action.getName().equals("previous")) {
    request.getPortletSession().setAttribute(
        "pageNo", ""+String.valueOf(Integer.parseInt(
            request.getPortletSession().getAttribute("pageNo").toString())-1));
}
if(action.getName().equals("next")) {
    request.getPortletSession().setAttribute(
        "pageNo", String.valueOf(Integer.parseInt(
            request.getPortletSession().getAttribute("pageNo").toString()+1));
}

```

5. Open the `/jsp/CustomerList/View.jsp` file and insert after the `<Domino:view>` tag a new `<Domino:page>` tag which will be in charge of paging through the view. Don't forget to close the tag just before the `</Domino:view>` tag. The tag inserted should be similar to the following:

```

<Domino:page
id="mypage"
start='<%= (String) request.getSession().getAttribute("pageNo") %>'
rows='<%= (String) request.getSession().getAttribute("rows") %>'>
...
</Domino:page>

```

- Just after you close the table that displays the customer view, insert the following code that will insert the Previous and Next buttons and display the page in which you are currently:

```

<table>
<tr>
<td>
<% if(mypage.getPage() > 1){%>
<a href="#" onClick="window.location.href='
<%= (String) request.getAttribute("previous") %>'">
    Previous
</a>

```

```

        <% } else {%>
        Previous
        <% } %>
    </td>
    <td>
        <% for(int i=1;i<=mypage.getPageCount();i++){
        if(i==mypage.getPage()){
            out.println("<b>"+i+" </b>");
        } else {
            out.println(i+" ");
        }
        } %>
    </td>
    <td>
        <% if(mypage.getPage() < mypage.getPageCount()){%>
        <a href="#" onClick="window.location.href='
            <%= (String) request.getAttribute("next") %>' ">
            Next
        </a>
        <% } else {%>
        Next
        <% } %>
    </td>
</tr>
</table>

```

This code inserts three columns to the table. The first one will display the previous link (which is aware of the position, so there won't be any link on the first page). The next column will display the different pages available and will display in bold the current page. The final column will display the next link (which is also aware of the position, avoiding presenting a link on the last page).

Perform a standard deployment as described in previous examples.

Your Customer List portlet should look similar to the one in Figure 6-3.

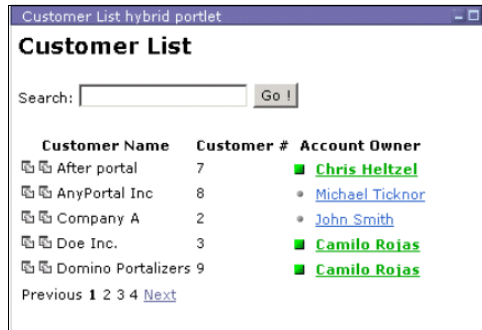


Figure 6-3 Customer List portlet enabled with search and paging

6.4 HelloWorldFromDominoServer portlet

Overview

This section guides you through the process of coding your first Java-based portlet, the “HelloWorldFromDominoServer” portlet. This is the simplest portlet you can write using the Domino Java API in portlet development. It has the following functionalities:

- ▶ Creates a session to Domino server.
- ▶ Saves a session, so that it can be reused.
- ▶ Gets some basic data from the server, like Notes version, platform, server name, and server URL.
- ▶ Once a portlet is refreshed, it uses the same session what was created at first.

Implementation details and example

In order to create Portlet applications suitable for deployment in WebSphere Studio Application Developer, you need to use the wizard. To manually construct the necessary folder structure and deployment descriptors would be tedious and utterly pointless.

1. To start creating a new Portlet Application, launch WebSphere Studio Application Developer and switch to the Portlet Perspective by selecting Perspective → Open → Other → Portlet.
2. Start the Portlet Application wizard in one of the following ways:
 - a. Select File → New → Portlet Application Project.
 - b. Right-click the Portlet Perspective and select New → Portlet Application Project from the context menu.

- c. The wizard can also be started from any perspective by selecting File → New → Other → Portlet development → Portlet application project.

When the wizard has started, you will see the screen shown in Figure 6-4.

Create a Portlet Project

Define the Portlet Project
Create a sample portlet application project.

Project name: HelloWorldFromDomino

Use default

New project location: C:\IBM\Redbook\WSAD\workspace\HelloWorldFromDomino

Enterprise application project: New Existing

New project name: HelloWorldFromDominoEAR

Use default

New project location: C:\IBM\Redbook\WSAD\workspace\HelloWorldFromDominoEAR

Context root: HelloWorldFromDomino

< Back Next > Finish Cancel

Figure 6-4 First screen of the Portlet Application project wizard

3. The options in this dialog are the following:
- **Project Name:** This value will determine the name of the project created by this wizard. The value entered here will be used throughout the remainder of the wizard as the default value for other parameters.
To follow our example, for a Project name field type:
HelloWorldFromDominoServer
 - **Use default location:** This checkbox indicates that you would like the entire contents of the application stored in the workspace. If you would like the contents of the application stored somewhere else on the file system, deselect this box.

If you deselect the Use Default Location check box, this field is enabled and allows you to specify the file path where the application will be saved.
To follow our example, check to use the default location.

- **Enterprise Application project name:** Though the Portal does not recognize EAR files, the Portlet application in WebSphere Studio must be contained in a Enterprise Application. When using the Portal debugging environment, all portlet applications contained in an EAR file are deployed together. You may choose an existing EAR file or enter a new one to be created.

To follow our example, type: HelloWorldFromDominoEAR

- **Context root:** This value will be used in the application.xml and .websettings files. It will not be the context root of the portlet application when deployed. Since the EAR is not used to deploy the portlet application into a full server, this value is only used when the ear is published to debug Portal server connected to WebSphere Studio Application Developer.

To follow our example, type: HelloWorldFromDominoServer

4. Click Next. This will bring up another dialog box, where you can select the type of portlet you would like to create inside the portlet application. The second window is shown in Figure 6-5 on page 315.

If you select Finish in the first screen of the wizard, an empty portlet application will be created. We don't want to do this in our sample.

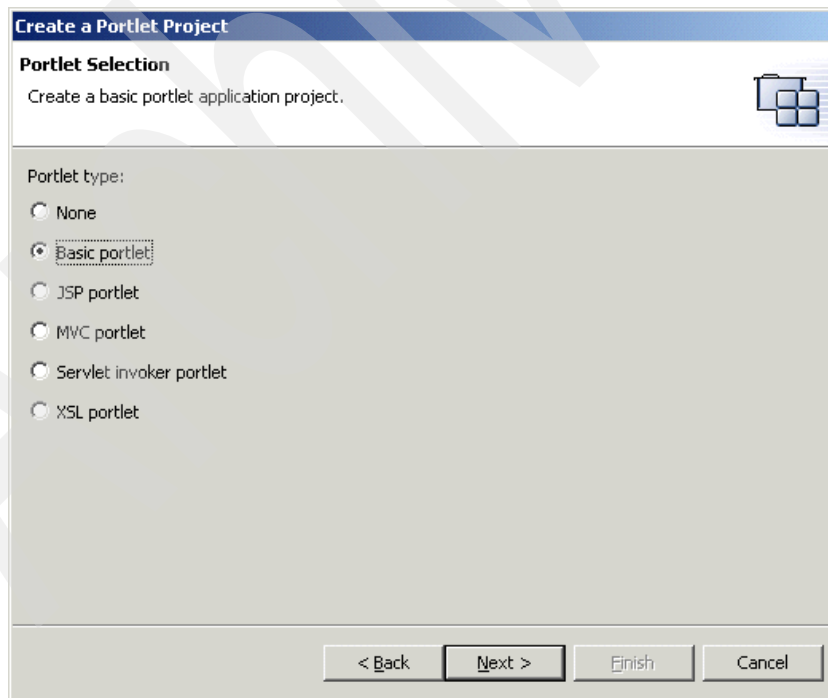


Figure 6-5 Second screen of the Portlet application project wizard

Presently, there are six options when deciding which kind of portlet you want included in the portlet application. You can only create a single portlet in an application via the wizard. For this example, choose **Basic portlet**.

The six types of portlets that can be created are as follows:

- **None:** This is the default option and will not create a portlet in the application. If you choose this option, you will need to manually add portlets to the application, as well as the deployment descriptors.
- **Basic portlet:** This is by far the most common option and will create a simple portlet in the portlet application. The portlet will extend from PortletAdapter and contain meaningful implementations of all four do methods. The implementations will adhere to the MVC approach. A bean will also be created for you to encapsulate your business logic.
- **JSP portlet:** This option assumes the entire application will be contained in a JSP. The application will specify the com.ibm.wps.portlets.JSPPortlet for deployment. This portlet simply forwards calls to a JSP. The JSP that is called is specified in the config-param of the concrete portlet section in the portlet.xml. By default, the wizard will specify and create a view.jsp file for you. You can choose a different name for the JSP in the third screen of the wizard. In the source folder, the wizard will place a dummy.java file that contains no code.
- **MVC portlet:** This choice utilizes the MVCPortlet provided as part of the com.ibm.wps.portlets package. This Portlet relays calls to Controller classes dedicated to servicing a specific markup. As such, if you select this type of portlet, a controller bean will be created for each markup you choose to support. The third screen of the wizard requires that you enter the base name of the controller classes. This name can be any value you like but is defaulted to MyController. This option will also create complete JSP structures for each markup.
- **Servlet invoker portlet:** This option will deploy the com.ibm.wps.portlets.ServletInvokerPortlet portlet and specify a URL as a parameter to the config-param in the portlet deployment descriptor. If you choose this option, the wizard requires you to provide a URL pointing to the servlet you wish to serve. The ServletInvoker portlet simply uses the ContentAccess Service to return unfiltered HTML. You must specify the URL you wish to serve in the final screen of the wizard. The resulting folder structure will contain a dummy.java file containing no code.
- **XSL portlet:** This option makes use of the com.ibm.wps.portlets.xslt.WpsXSLTPortlet. This portlet simply accepts an xml file and a style sheet as parameters and uses these to create a presentation via the XSLT transformation. These parameters must be supplied on the final window of the wizard. If you do not yet have the requisite files, the wizard can use

default sample xml and xsl files. The parameters can be adjusted in the portlet.xml or at run time.

The screenshot shows a wizard window titled "Create a Portlet Project" with a sub-header "Basic Portlet Parameters". Below the sub-header is the instruction "Enter the properties of the basic portlet." and a small icon of a folder with a grid. The form contains several input fields and a checkbox group:

- Portlet application name: HelloWorldFromDomino application
- Portlet name: HelloWorldFromDomino portlet
- Concrete portlet application name: HelloWorldFromDomino application
- Concrete portlet name: HelloWorldFromDomino portlet
- Default locale: en (dropdown) English
- Concrete portlet title: HelloWorldFromDomino portlet
- Portlet class name: HelloWorldFromDomino
- Markups: html chhtml wml VoiceXML

At the bottom of the form are four buttons: "< Back", "Next >", "Finish", and "Cancel".

Figure 6-6 Third screen of the Portlet application project wizard

- There are eight fields required in the final window of the wizard, shown in Figure 6-6. By default, all fields have been completed for you. Most of these fields are used to complete the portlet.xml deployment descriptor. For more information on the portlet.xml and deployment descriptor see the previous chapter. The fields and their meanings are as follows:
 - **Portlet application name:** This name is used in the portlet.xml to specify the abstract application name. This value will never be seen by the administrator of the portlet or the end-user. Generally, there is no need to change this value.
 - **Portlet name:** This value is used to identify the abstract portlet. This name will never be seen by the administrator or the end user. There is typically no need to alter this value.
 - **Concrete portlet application name:** This name is used in the portlet.xml to specify the concrete application. The administrator will see this value in the portal. If you intend to add more concrete applications to this portlet application, you may want to change this value. Otherwise, there is no need to adjust this value.

- **Concrete portlet name:** This name specifies this concrete portlet to be deployed. The end user will see this value when they add the portlet to a page. Generally, there is no need to adjust this value.
 - **Default locale:** This value adds the default locale and the language block to the portlet.xml.
 - **Concrete portlet title:** This will complete the language block with the title. The description, short-title and keywords elements are included in the language block but left empty.
 - **Portlet class name:** This name will be used as the name of the portlet created for you. You should adjust this value to reflect the package name you would like to use. Type `HelloWorldFromDominoServer` in order to follow our sample. If you do not enter a package name, the wizard will place the portlet in the default portlet package.
 - **Markups:** These checkboxes indicate which markup languages you intend to support. By selecting a value, a new folder will be created under the JSP folder containing JSPs specifically for the markup.
6. Finally, click Finish and the wizard will create the necessary folder structure, classes, JSPs, and deployment descriptors.

Developing the portlet

The wizard will create the skeleton of the portlet, which you can then use as a foundation for your portlet development. Figure 6-7 on page 319 shows the result of creating a basic portlet application with the wizard.

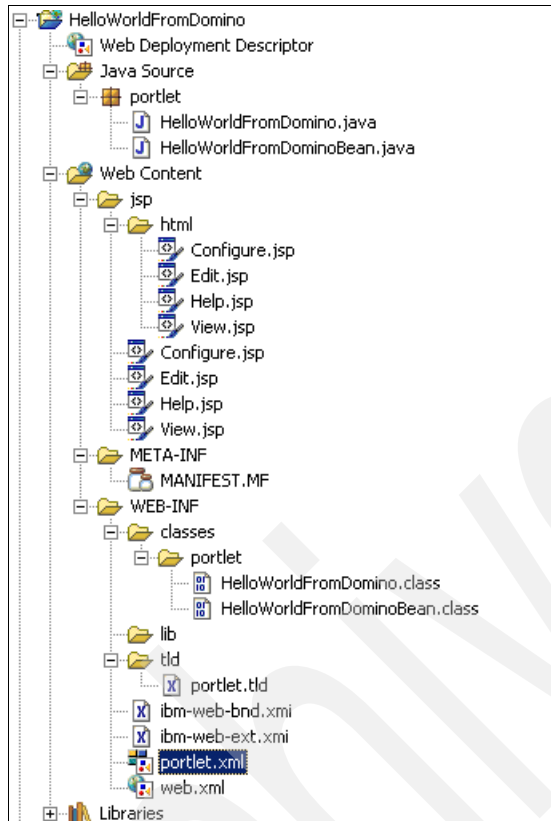


Figure 6-7 Skeleton of the portlet

The generated project contains the following folders and files by default:

- ▶ **Java Source:** This folder contains the Java files that make up the portlet application. By default, the wizard assumes you will follow an MVC approach and creates a simple Java bean for you. Whatever package name was specified in the final screen of the wizard will be created. If a simple class name was specified without a package, the wizard will place the portlet in a package name portlet. In our example we did not build JavaBeans.
- ▶ **Web Content:** This folder contains everything needed to deploy the application to the portal. Essentially, this folder will become the war file. It contains two sub folders: jsp and WEB-INF.
- ▶ **jsp:** This folder will contain all the JSPs used by the application to create the content of the portlet. For each markup you choose to support, a directory will be created containing JSPs for each of the four modes a portlet may support. It will also contain four JSP files for the modes under the root. In the event that the portal is unable to match a client to a markup folder, it will use the default

JSPs contained in the root. To keep development simple and clean, you may choose to delete the default JSPs and work only with the HTML JSPs. In our example we did not build any JSPs.

- ▶ **WEB-INF:** This folder contains the compiled code and deployment descriptors used by the Portal to install the application.
 - **classes:** If your compiled portlet class files are not packaged into a jar file, they are included in this directory. The complete package structure is created in this folder.
 - **lib:** This directory contains any jar files that your application makes use of and which are not normally available in the Portal environment via the classpath. Also, if you have packaged your compiled portlets into a jar, the jar file is placed in this directory. Typically you will import the ncsso.jar file to here.
 - **tld:** This is included to allow JSPs to compile and recognize the custom tags available in the portal environment. This folder and file are not required at deployment time since the tld is installed with Portal. To make maintenance easier and more reliable, you may choose to delete this file upon deployment.
 - **ibm-web-bnd.xmi:** This file is not used by the portal environment but is included with all Web applications created in WebSphere Studio.
 - **ibm-web-ext.xmi:** This file is not used by the portal environment but is included with all Web applications created in WebSphere Studio.
 - **portlet.xml:** This is the deployment descriptor required by the Portal server to install the portlet application. It must be located under the WEB-INF folder or installation will fail.
 - **web.xml:** This deployment descriptor is required by the application server to install the Web application. It must be located under the WEB-INF folder or installation will fail.
- ▶ **.classpath:** This file is used by WebSphere Studio to locate the jar files containing the Portlet APIs. It is required for your portlets to compile but is not included in deployment.
- ▶ **Libraries:** This folder contains Java files needed in portlet development.

Modifying the skeleton

After the portlet wizard builds the skeleton of your portlet, you need to make the following modifications to it.

1. Add the NCSO.jar file to portlet structure.

Choose the lib folder, then select File → Import → File Systems. Select ncsso.jar, and click Finish. Your workspace will look like Figure 6-8 on page 321.

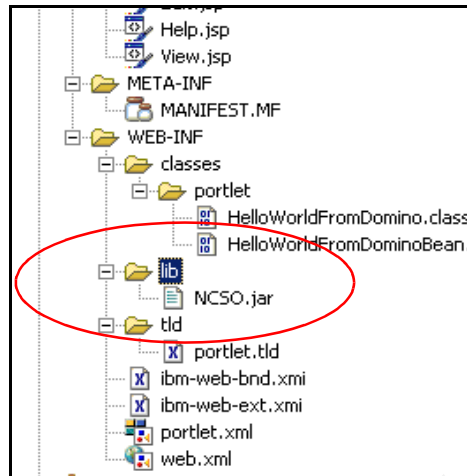


Figure 6-8 Added ncsso.jar

2. Modify Java code in the HelloWorldFromDomino class file.

Select Java Source → portlet → HelloWorldFromDomino. The wizard generated a lot of code for basic use. Make the following modification to the code:

a. Replace doView() method with this:

```
public void doView(PortletRequest request, PortletResponse response) throws
PortletException,IOException{
    // Calling method getSession in viewmode
    getSession(request,response);
}
```

b. Create getSession() method and place it after doView() method. The code has been commented. Replace <servername>, <username> and <password> fields with those suitable for accessing your existing Domino server. The user should have required access rights to the Domino server as described in “Enabling Domino server for DIIOp connection” on page 251.

Example 6-1 getSession() method

```
private Session getSession(PortletRequest request, PortletResponse
response)throws PortletException,IOException {
    // Check first if session already exist
    Session diopsession =
(Session)request.getSession().getAttribute("diopsession");
```

```

// Creating writer for output
PrintWriter writer = response.getWriter();

// Check if diopsession already exists, if it is, use it, if not,
create one
if (diopsession == null)
    try{
        // Creating diop session for Domino
        diopsession =
NotesFactory.createSession("<servername>","<username>","<password>");

        // Set session to attribute for reuse
request.getSession().setAttribute("diopsession",
diopsession);

        // Sending Hellos first time
writer.println("<b>HelloWorldFromDominoServer!</b><br>You have
just created diop session to Domino through
portlet!<br><br><br>ServerNotesVersion:" +
        (String)diopsession.getNotesVersion() + "<br>ServerPlatform: "
+
        (String)diopsession.getPlatform() + "<br>ServerName: " +
        (String)diopsession.getServerName() + "<br>ServerURL: " +
        (String)diopsession.getURL());

    }catch (NotesException e){

        // Sending Hellos from exception
writer.println("HelloFromDominoServer Portlet ran, but with
exception" + e.id + " msg:" + e.text );

    }else{

        // Sending Hellos from reused session
writer.println("<b>HelloFromDominoServer!</b><br><br>You've
successfully reused diopsession, cool!");

    }

return diopsession;
}
}
}

```

c. In the beginning of the code, declare the necessary imports like this:

Example 6-2 Add these imports in the beginning of the ACLBrowser.java file

```
package portlet;  
//Domino Objects for JAVA API  
import lotus.domino.*;  
//Portlet API  
import org.apache.jetspeed.portlet.*;  
import org.apache.jetspeed.portlets.*;  
//Java stuff  
import java.io.*;  
import java.io.PrintWriter;
```

The code is now ready to run at the Portal.

3. Export the portlet file.

To deploy the portlet application, the webApplication directory must be packaged into a WAR file. To create a WAR file for deployment, select the application and right-click it. From the context menu, select Export WAR. You can also get this option from the menu by selecting File → Export and selecting WAR File from the list of options.

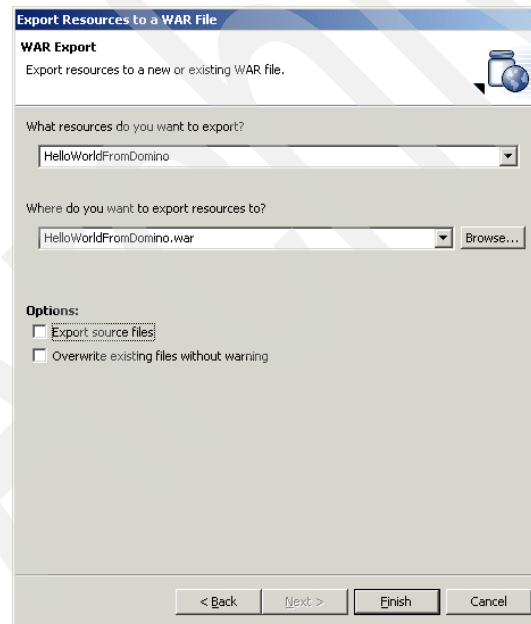


Figure 6-9 Exporting portlet from Application Developer

Export the WAR file. Name the file HelloWorldFromDomino.war. By default it will be stored to root main drive (for instance, in windows c:\) You can specify a more suitable folder if you wish.

4. Deploy the portlet and add it to the page.

HelloWorldFromDominoServer portlet in action

When you open the page for the first time, HelloWorldFromDominoServer portlet should give you information about your server (NotesVersion, Platform, server name, and server URL). Figure 6-10 illustrates this.

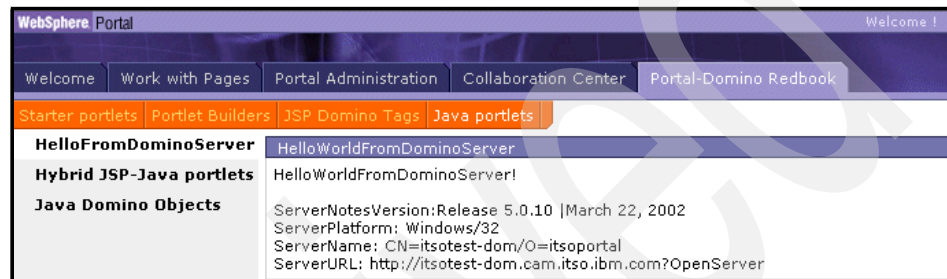


Figure 6-10 HelloWorldFromDominoServer in action

Refresh the browser; the portlet is using session, and you will get a different message from the portlet, as illustrated in Figure 6-11.

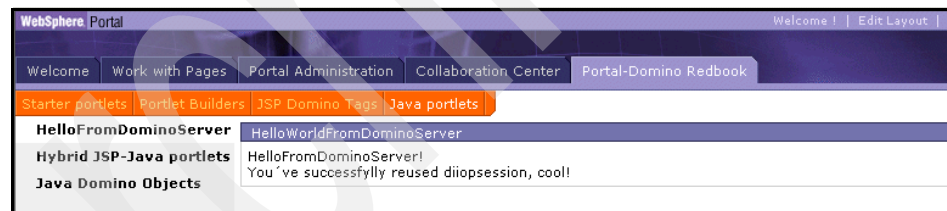


Figure 6-11 HelloWorldFromDominoServer in action reusing a session

Congratulations, you have just finished your first full Java portlet integration with Domino.

6.5 Using JavaBeans in the sample portlet

In the HelloWorldFromDominoServer portlet example, you wrote an output directly to the portlet. A better way to accomplish the same thing is to create a JSP for output and use JavaBeans to pass values to the JSP from Java code.

Overview of JavaBeans

The JavaBeans component architecture extends “write once, run anywhere” capability to reusable component development. In fact, the JavaBeans architecture takes interoperability a major step forward: your code runs on every OS and also within any application environment.

JavaBeans brings component technology to the Java platform. With the JavaBeans API you can create reusable, platform-independent components. Using JavaBeans-compliant application builder tools, you can combine these components into applets, applications, or composite components. JavaBean components are known as “Beans.”

The JavaBeans specification defines a set of standard component software APIs for the Java platform. The specification was developed by Sun Microsystems.

Working through this example gives you the opportunity to extend your Java programming skills from the HelloWorld sample to the use of JavaBeans. This example also reinforces understanding of the MVC model as well: Model is the JavaBean, View is the JSP file, and Controller is the Java file.

Implementation details and example

Starting from the same point as the previous HelloWorldFromDominoServer exercise, open the project and proceed as follows:

1. Create a JavaBean class called PortletDominoBean. Do this by selecting New → Other → Java → Class → Next; for name field type PortletDominoBean, and click finish.

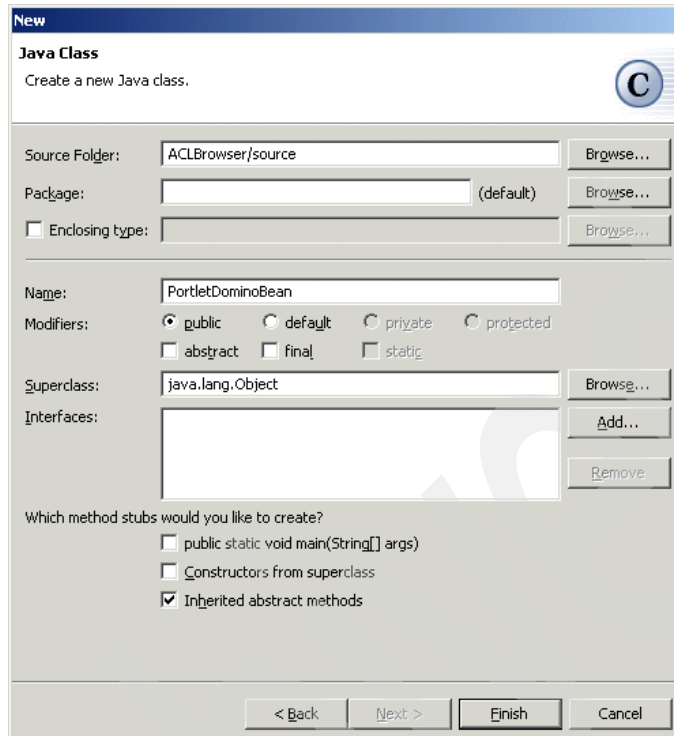


Figure 6-12 Creating a JavaBean

2. Rewrite the PortletDominoBean class as shown in example Example 6-3.

Example 6-3 Example of JavaBean

```
public class PortletDominoBean {
    private String NotesVersion = "";

    public void setNotesVersion(String s) {
        NotesVersion = s;
    }
    public String getNotesVersion() {
        return (NotesVersion);
    }
}
```

3. Modify the portlet code to use the JavaBean. The modified code is shown in Example 6-4 on page 327. Once you are able to getNotesVersion from the session, you will add it to the bean.

Example 6-4 Functionality in doView

```
if (diopsession == null)
    try{
        // Creating diiop session for Domino
        diopsession =
NotesFactory.createSession("<servername>", "<username>", "<password>");

        // Set session to attribute for reuse
        request.getSession().setAttribute("diopsession",
diopsession);
        //Make a bean
        PortletDominoBean bean = new PortletDominoBean();
        //Save name in bean
        bean.setNotesVersion((String)diopsession.getNotesVersion());
        //Save bean in request
        request.setAttribute("PortletDominoBean", bean);
        //Invoke the JSP to render
        getPortletConfig().getContext().include("/jsp/View.jsp
", request, response);
```

4. Create a JSP file to make the final UI for the browser. Select New → JSP; in the name field type View.jsp; and click Finish.

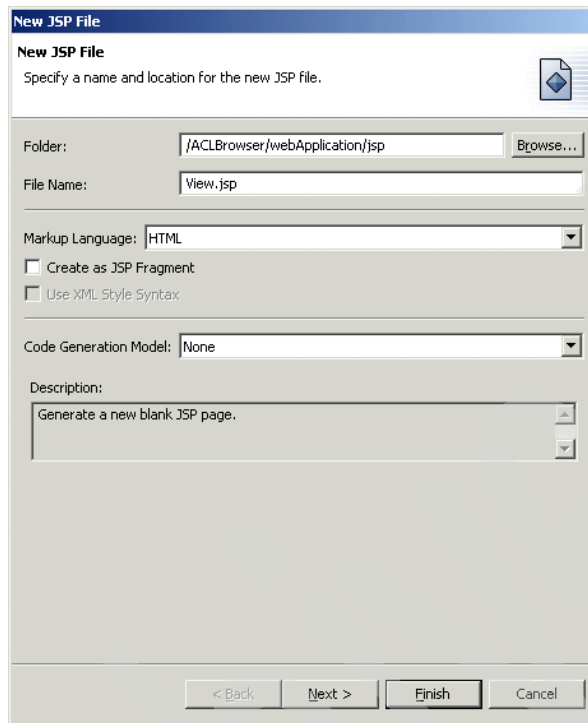


Figure 6-13 Creating a JSP file

5. Click Source from the designer and edit the view.jsp code. The modified code is presented in Example 6-5.

Example 6-5 Using JavaBean in JSP

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML><HEAD>
<%@ page language="java" contentType="text/html; charset=WINDOWS-1252"%>
<META http-equiv="Content-Type"
    content="text/html; charset=WINDOWS-1252">
<META name="GENERATOR" content="IBM WebSphere Studio">
<TITLE>View.jsp</TITLE>
</HEAD><BODY>
<jsp:useBean id="PortletDominoBean" class="portlet.javacode.PortletDominoBean"
scope="request"/>
<h1>Notes version through java bean is
<%=PortletDominoBean.getNotesVersion()%></h1></BODY></HTML>

```

The result of this example is identical to the previous one; therefore, we don't repeat the deployment. The only difference from the previous example is how the data is stored.

6.6 Browsing Domino ACL portlet

The purpose of this example is to build a portlet which:

- ▶ Accesses the Domino server
- ▶ Creates a list of all databases on that server so the user can choose which database ACL to browse
- ▶ Creates a listing of the ACL for the selected database

You will use a variety of techniques to create this portlet: JSPs, JavaBeans, data handling mechanisms with Java (arrays and collections) and iterators to output data. You will also add images and functionality from a JavaScript file to a portlet. Figure 6-14 shows the portlet that is the end result of this example.

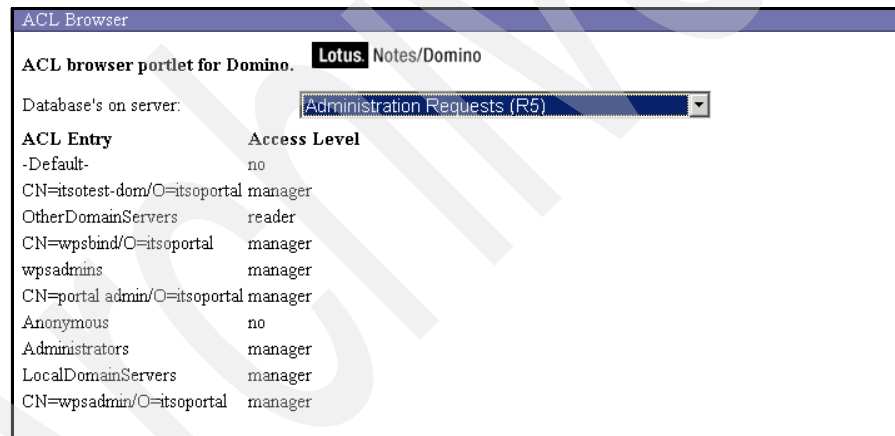


Figure 6-14 ACLBrowser portlet creates list of databases in Domino and lists ACL entries.

Work through this example to extend your skills at Java programming using a mix of Java, JavaBeans, JSP, JavaScript, and image and data handling in a portlet context.

Implementation details and example

1. Start your development work by creating a new portlet; name it ACLBrowser. (Use the same steps presented in 6.4, "HelloWorldFromDominoServer")

portlet” on page 313 to do this.) Create and open the ACLBrowser.java file for editing.

2. Insert/replace the code in the ACLBrowser.java file with that in Example 6-6. (You can copy/paste it.) Modifications you make to the Java code will provide the following functionalities:
 - Initialize the portlet and create a session to Domino.
 - Create an own entry list for holding values for ACLEntry and ACLLevel, put it in ArrayList, and save it according to the “value” request. If “value” is empty by default, load the ACL list for the names.nsf, otherwise, load the list of the ACL entries for the selected database.
 - Create an own entry list holding values for database names in server and database path, put it in ArrayList, and save it to request.
 - Create a JavaBean for sets and gets of value.

Example 6-6 ACLBrowser.java code

```
package portletjavacode;

//Domino Objects for JAVA API
import lotus.domino.*;
import lotus.domino.Database;

//Portlet API
import org.apache.jetspeed.portlet.*;
import org.apache.jetspeed.portlets.*;

//Java stuff
import java.io.*;
import java.io.PrintWriter;
import java.util.*;
import java.lang.String;

public class ACLBrowser extends AbstractPortlet {

    public void init(PortletConfig portletConfig) throws UnavailableException {
        // Initializing
        super.init(portletConfig);
    }

    public void doView(PortletRequest request, PortletResponse response)
        throws PortletException, IOException {
        // Calling method getSessions in viewmode
        getSession(request, response);
    }
}
```

```

private Session getSession(
    PortletRequest request,
    PortletResponse response)
    throws PortletException, IOException {

    // Check first if session already exist
    Session diopsession =
        (Session) request.getSession().getAttribute("diopsession");

    // Creating writer for output
    PrintWriter writer = response.getWriter();

    // Check if diopsession already exists,if it does not, create one
    if (diopsession == null)
        try {
            // Creating diop session for Domino
            diopsession =
NotesFactory.createSession("<servername>","<username>","<password>");
            // Set session to attribute for reuse
            request.getSession().setAttribute("diopsession", diopsession);
            //getACL(request,response,diopsession);
        } catch (NotesException e) {
            writer.println("ACL browser portlet ran, but with exception"+
e.id+ " msg:"+ e.text);
        }

        //creating acllist
        getACL(request, response, diopsession);
        return diopsession;
    }

    //Method for getting ACL entries
    private String getACL(PortletRequest request,PortletResponse
response,Session diopsession)
        throws PortletException, IOException {

        try {
            //Make a bean
            PortletDominoBean bean = new PortletDominoBean();

            //Read from request if database was selected from list
            request.getSession().setAttribute("value",
request.getParameter("value"));
            String dbtoview = request.getParameter("value");
            if(dbtoview == null){
                dbtoview = "names.nsf";
            }

            Database database = diopsession.getDatabase("", dbtoview);

```

```

ACL acl = database.getACL();
ACLEntry entry = acl.getFirstEntry();
entry.getLevel();
//Save name in bean
bean.setACL((String) entry.getName());

////
ArrayList acllist = new ArrayList();

do {
    String ACLEntry = entry.getName();
    int ACLLevel = entry.getLevel();
    String lev = null;
    //transforming the ACL levels in to text
    switch (entry.getLevel()) {
        case ACL.LEVEL_NOACCESS :
            lev = "no";
            break;
        case ACL.LEVEL_DEPOSITOR :
            lev = "depositor";
            break;
        case ACL.LEVEL_READER :
            lev = "reader";
            break;
        case ACL.LEVEL_AUTHOR :
            lev = "author";
            break;
        case ACL.LEVEL_EDITOR :
            lev = "editor";
            break;
        case ACL.LEVEL_DESIGNER :
            lev = "designer";
            break;
        case ACL.LEVEL_MANAGER :
            lev = "manager";
            break;
    }

    //build the list
    AcITestEntry NameAndLevel = new AcITestEntry(ACLEntry, lev);
    acllist.add(NameAndLevel);
} while ((entry = acl.getNextEntry(entry)) != null);

//Save bean in request
request.setAttribute("DATA_ACL", acllist);
getDBList(request, response, diopsession);
} catch (NotesException e) {
    PrintWriter writer2 = response.getWriter();

```

```

        writer2.println("Error making ACL: " + e.id + " msg:" + e.text);

    }

    return null;
}

private class AclEntry implements ACLEntry {
    private String ACLEntry = null, ACLLevel = null;

    public AclEntry(String ACLEntry, String ACLLevel) {
        this.ACLEntry = ACLEntry;
        this.ACLLevel = ACLLevel;
    }

    public String getACLEntry() {
        return ACLEntry;
    }
    public String getACLLevel() {
        return ACLLevel;
    }
}

}

//method getting databaselist
private String getDBList(
    PortletRequest request,
    PortletResponse response,
    Session diopsession)
    throws PortletException, IOException {

    //lets check if databaselist has already created
    Session prevdblist =
    (Session) request.getSession().getAttribute("dblist");
    if (prevdblist == null){

    try {
        //Make a bean
        PortletDominoBean bean = new PortletDominoBean();

        //
        ArrayList dblist = new ArrayList();

        DbDirectory dir = diopsession.getDbDirectory(null);
        String server = dir.getName();
        if (server == "")
            server = "Local";
        Database db = dir.getFirstDatabase(DbDirectory.DATABASE);
    }
}
}

```

```

//building the list of database
while (db != null) {
    String dbfp = db.getFilePath();
    String dbtitle = db.getTitle();
    DBListTestEntry values = new DBListTestEntry(dbfp, dbtitle);
    dblist.add(values);
    db = dir.getNextDatabase();
}

dir.recycle();

request.setAttribute("DATA_DBLIST", dblist);
getPortletConfig().getContext().include("/jsp/View.jsp",request,
response);

} catch (NotesException e) {
    PrintWriter writer2 = response.getWriter();
    writer2.println("Error making DBList: " + e.id + " msg:" + e.text);
}
}

return null;
}

private class DBListEntry implements DBListEntry {
    private String DBFilePath = null, DBTitle = null;
    public DBListEntry(String DBFilePath, String DBTitle) {
        this.DBFilePath = DBFilePath;
        this.DBTitle = DBTitle;
    }

    public String getDBFilePath() {
        return DBFilePath;
    }
    public String getDBTitle() {
        return DBTitle;
    }
}
}
}

```

Remember to replace <servername>, <username> and <password> with the correct values for your environment.

3. Save the file.
4. Create the three Java files in Examples 6-7, 6-8, and 6-9 to take care of needed entries.

Example 6-7 ACLEntry.java

```
package portletjavacode;
public interface ACLEntry
{
    public abstract String getACLEntry();
    public abstract String getACLLevel();
}
```

Example 6-8 DBListEntry.java file

```
package portletjavacode;

public interface DBListEntry
{
    public abstract String getDBFilePath();
    public abstract String getDBTitle();
}
```

Example 6-9 PC.java file

```
package portletjavacode;
public interface PC
{
    public static final String DATA_ACL = "DATA_ACL";
    public static final String DATA_DBLIST = "DATA_DBLIST";
    public static final String PARAM_VALUE = "value";
}
```

5. Create the following JavaBean file.

Example 6-10 PortletDominoBean.java file

```
package portletjavacode;
public class PortletDominoBean {

    private String ACL = "";
    public void setACL(String s) {
        ACL = s;
    }
    public String getACL() {
        return (ACL);
    }
}
```

6. Start to build the view.jsp for gathering output by first creating the view.jsp file.

7. Create a folder called images and place the file named logo.gif (the image you want to add to the portlet) into the folder. Add the following lines to view.jsp to retrieve the image.

```
%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="resources" %>
% String imagePath = portletResponse.encodeURL("/images/"); %>
img src="<%= imagePath %>logo.gif" border="0" align="top">
```

8. Create a folder for JavaScript files called js. Create a JavaScript file, give it the name Redbook.js, and save it to the js folder. Insert the following code in the JavaScript file.

```
function SelectBoxActionSet(objForm, objSelect)
{
    objForm.action = eval('objForm.' + objSelect + '.value')
}
```

9. Add the following lines to view.jsp to add js functionalities:

```
<script language="JavaScript" src="<portletAPI:encodeURI
path="/js"/>/Redbook.js"></script>
<form name="mobileview" id="frmGPResult"
onSubmit="SelectBoxActionSet(this,'gpResult');"
method="post">
```

10. Add the following lines to view.jsp to retrieve a list of databases and ACLs, as well as to get JavaScript SelectBoxActionSet method.

Example 6-11 Retrieving list of database

```
///
<%@ page import="java.util.Collection,
    java.util.Iterator,
    portletjavacode.ACLEntry,
    portletjavacode.DBListEntry,
    portletjavacode.PC"%>
<%@ taglib uri="/WEB-INF/tld/portlet.tld" prefix="portletAPI" %>
<portletAPI:init />
<jsp:useBean id="PortletDominoBean" class="portletjavacode.PortletDominoBean"
scope="request"/>
///
```

```

<select name="gpResult"
onChange="SelectBoxActionSet(document.mobileview,'gpResult');

submit());">

<%
Collection dblast2 = (Collection) request.getAttribute(PC.DATA_DBLIST);
for (Iterator iterator = dblast2.iterator() ; iterator.hasNext() ;)
{
    DBListEntry entry = (DBListEntry) iterator.next();
%>

<option value="<portletAPI:createURI >
<portletAPI:URIParameter name="<%= PC.PARAM_VALUE %>"
value="<%=entry.getDBFilePath()%>"/>

</portletAPI:createURI>"><%= entry.getDBTitle() %></option>

<%
}
%>

</select>

// ACL List generated here:
<table border="0">
<tr><td><b>ACL Entry</b></td><td><b>Access Level</b></td></tr>
<%
Collection ACLs = (Collection) request.getAttribute(PC.DATA_ACL);
for (Iterator iterator = ACLs.iterator() ; iterator.hasNext() ;)
{
    ACLEntry entry = (ACLEntry) iterator.next();
%>
    <tr><td>
entry.getACLLevel()
%></td></tr>
<%
}
%>

```

11. Save all the open files, deploy the portlet, and add it to a page.

6.7 How to use log4j

Here is a simple example of how log4j can be used with the portlet.

Before working through the example, use the following steps to obtain log4j functionalities:

1. Download the log4j distribution from:
<http://jakarta.apache.org/log4j/docs/download.html>
2. Extract the archived files to some suitable directory.
3. Import the file `dist/lib/log4j-1.2.6.jar` to your development environment. Select `File → Import → File System`, browse to the log4j jar file, and import it to the portlet `WEB-INF/lib` folder.

Example 6-12 is a very simple example of a program implementing a `SimpleLayout` and `FileAppender`. It will write a log to the `outputfile.txt` file.

Example 6-12 Adding log4j to HelloWorldFromDomino portlet

```
///Import
import org.apache.log4j.Level;
import org.apache.log4j.Logger;
import org.apache.log4j.SimpleLayout;
import org.apache.log4j.FileAppender;

public class HelloWorldFromDomino extends AbstractPortlet {

    static Logger logger = Logger.getLogger(HelloWorldFromDomino.class);

    public void init(PortletConfig portletConfig) throws UnavailableException {

        SimpleLayout layout = new SimpleLayout();
        FileAppender appender = null;
        try {
            appender = new FileAppender(layout,"outputfile.txt",false);
        } catch(Exception e) {}

        logger.addAppender(appender);
        logger.setLevel((Level) Level.DEBUG);

        logger.debug("Here is some DEBUG");
        logger.info("Here is some INFO");
        logger.warn("Here is some WARN");
        logger.error("Here is some ERROR");
        logger.fatal("Here is some FATAL");
    }
}
```

```

        // Initializing
        super.init(portletConfig);

    }

    public void doView(PortletRequest request, PortletResponse response) throws
PortletException,IOException{
        // Calling method getSession in viewmode
        logger.debug("We are now in doView!");
        getSession(request,response);
    }

    private Session getSession(PortletRequest request, PortletResponse
response)throws PortletException,IOException {

        // Check first if session already exist
        logger.debug("Trying to create diopsession!");
        Session diopsession =
(Session)request.getSession().getAttribute("diopsession");

    ...

```

A more sophisticated way to do this is to put all logging settings in the ResourceBundle property file, and also implement class name ahead of every log. That way you are able to configure the settings in an external file (switch logging off and on, set log level, and so forth etc.) and get more informative and useful logs.

To do this, perform the following steps:

1. Create a properties file for log4j. Select File → New → Simple → File; name the log4j.properties. This is illustrated in Figure 6-15.

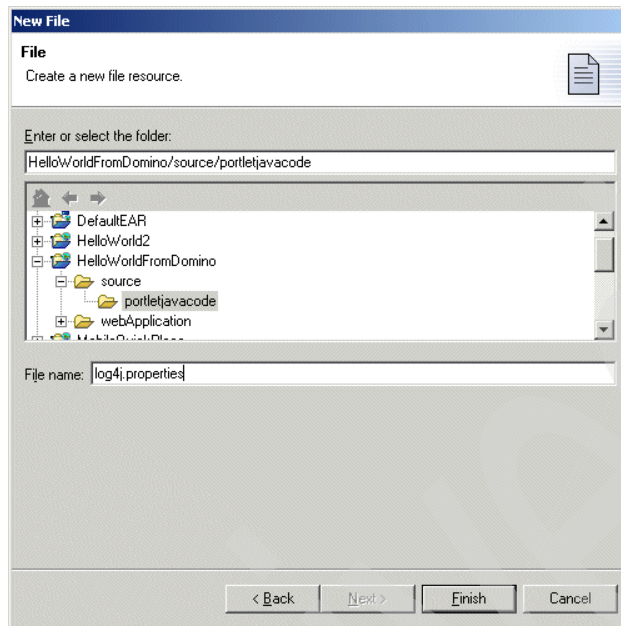


Figure 6-15 Creating log4j.properties file

2. Open your log4j.properties file and copy the following code into the newly created file.

Example 6-13 Configuration file for logging

```
# Initialise root logger level DEBUG and call it A1.
log4j.rootLogger=DEBUG, A1

# A1 is set to be a FileAppender.
log4j.appender.A1=org.apache.log4j.FileAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d [%t] %-5p %c - %m%n
log4j.appender.A1.File=C:/OurOwnPortletLogFile.txt
log4j.appender.A1.append=true
```

3. Save the configuration file. Logging will be written to the c:/OurOwnPortletLogFile.txt file. More possible configuration settings are discussed on the Apache Web site's log4j documents page.
4. Import the logging.jar file to your portlet. Logging.jar is a simple logging utility with support for logging filters.

The structure of your project should now resemble the one shown in Figure 6-16.

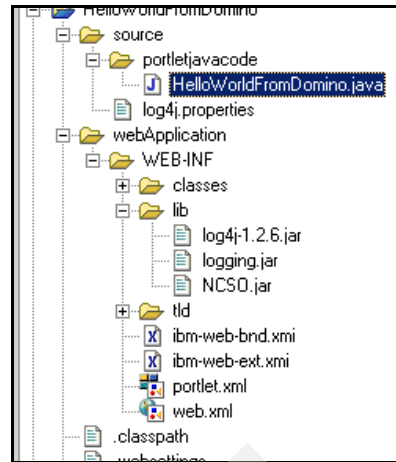


Figure 6-16 Structure of portlet in second logging sample

5. You can now start to put your log calls anywhere in the portlet code you want to. Example 5-25 illustrates this.

Example 6-14 Another example of sophisticated portlet logging

```
..
//Import
import com.ibm.logging.Log;

public class HelloWorldFromDomino extends AbstractPortlet {

    public void init(PortletConfig portletConfig) throws UnavailableException {
        // Initializing
        Log.debug(getClass(), "Initializing portlet");
        super.init(portletConfig);
    }

    public void doView(PortletRequest request, PortletResponse response) throws
PortletException,IOException{
        // Calling method getSession in viewmode
        Log.debug(getClass(), "Trying to get session..");
        getSession(request,response);
        Log.debug(getClass(), "Get session method ran..");
    }
}
```

```
private Session getSession(PortletRequest request, PortletResponse
response)throws PortletException,IOException {

    // Check first if session already exist
    Log.debug(getClass(), "Trying to get session from session");
    Session diiosession =
(Session)request.getSession().getAttribute("diiosession");
```

6.8 Session pooling

In this exercise you will create a very simple example of a Domino session pool, based on the open source Jakarta Commons pool implementations.

The Jakarta Commons product has a pool component ready to extend. It is very easy to use and will give the portlet developer excellent results. To obtain more information on the Jakarta Commons pool, see the following Web site:

<http://jakarta.apache.org/commons/pool/>

To work through this example, download the Commons Pool code from the Web page and unzip the file to a folder that we will reference as COMMONS_POOL_HOME.

Note: The example of this pool can be downloaded from the on-line resources of this redbook. Refer to Appendix B, “Additional material” on page 421 for instructions on downloading the example code.

At a high level, this example will:

- ▶ Create a Domino Session Pool factory
- ▶ Create a sample Java client which tests your pool
- ▶ Analyze the performance of your pool compared with a create-destroy approach for Session Management

The detailed steps are in the following sections.

Domino Session Pool Factory Java class

Start by creating the Domino Session Pool Factory. You will extend the BasePoolableObjectFactory class that comes with the Jakarta Commons pool, and define in your factory how a Session is created and destroyed.

1. Open WebSphere Studio 5.
2. Switch to the Java perspective.

3. Select the File → New → Project menu.
4. Select Next.
5. Name the project DominoSessionPool and click Next.
6. In the Java Settings dialog in the Source tab, change the “Use the project as source folder” to “Use source folders contained in the project,” then click Create New Folder, enter src, and click OK. This is shown in Figure 6-17.

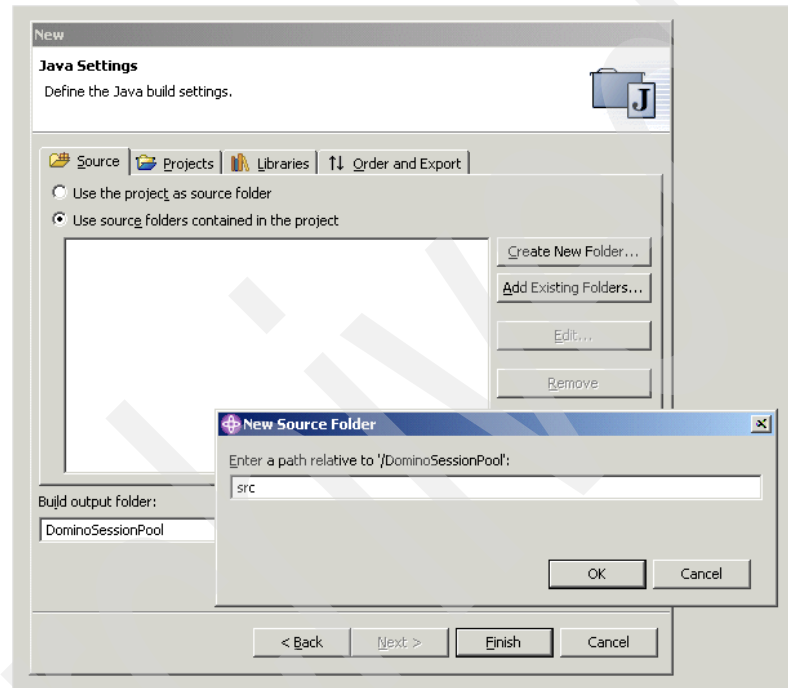


Figure 6-17 Creating DominoSessionPool project

7. A new dialog confirms the creation of a new build output directory to DominoSessionPool/bin. Click Yes.
8. Switch to the Libraries tab, and add the following external JARs:
 - a. Add NSCO.jar from the DOMINO_HOME\data\Domino\java directory.
 - b. Add Notes.jar from the DOMINO_HOME directory
 - c. Add commons-pool.jar from the COMMONS_POOL_HOME\commons-pool-1.0.1 directory

The libraries tab should look like Figure 6-18 on page 344.

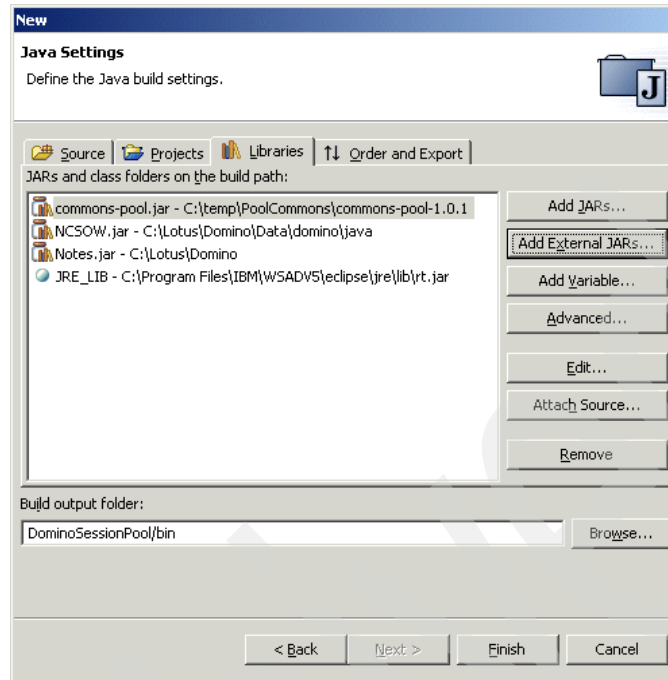


Figure 6-18 Creating DominoSessionPool project: Adding Libraries

9. Click Finish.
10. Select the newly created DominoSessionPool project, right-click the src folder, and select New → Package.
11. Name the package `com.ibm.itso.sg247004.DominoSessionPool`; click Finish.
12. Right-click the newly created package and select New → Class.
13. Name the class `DominoSessionFactory` and extend the `BasePoolableObjectFactory` as shown in Figure 6-19. Click Finish.

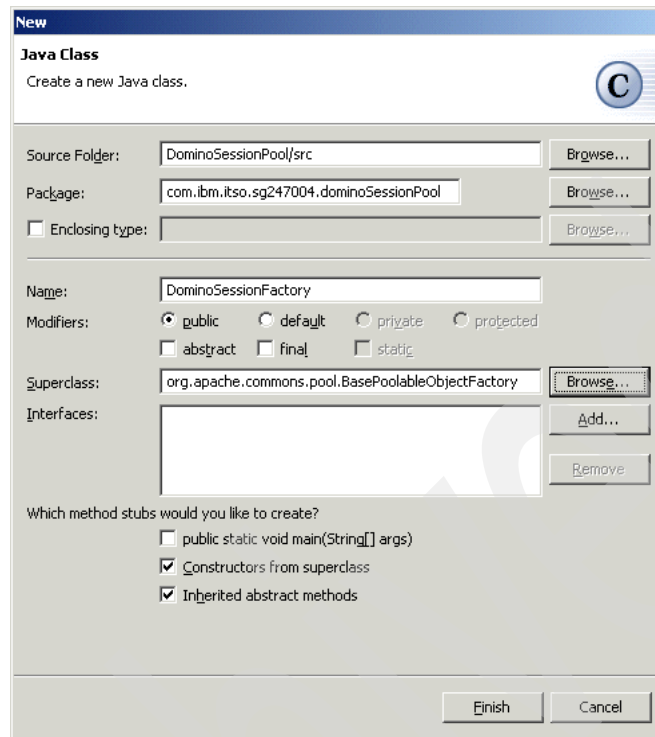


Figure 6-19 Creating the DominoSessionFactory

14. The DominoSessionFactory should pop up. Add the following import statement at the beginning of the class:

```
import lotus.domino.*;
```

15. Replace the make() method with the following code:

```
public Object makeObject() throws Exception {
    System.out.println("\t DomSessFact: Creating Domino Session...");
    Session s = NotesFactory.createSession(
        "<servername>", "<username>", "<password>");
    System.out.println("\t DomSessFact:
        Domino Session successfully created !");
    return s;
}
```

16. Add a destroyObject() method with the following code:

```
public void destroyObject(Object arg0) throws Exception {
    System.out.println("\t DomSessFact: Destroying Domino Session...");
    Session s=(Session)arg0;
    s.recycle();
    super.destroyObject(arg0);
}
```

```
        System.out.println("\t DomSessFact: Domino Session destroyed  
                           successfully !");  
    }  
}
```

17. Save and close the DominoSessionFactory class.

Client Java class

You have now built a Domino session factory that will take care of basically creating and destroying the code. For the client example you will create a simple executable class that will use this factory and reuse an existing session adequately.

Do this by performing the following steps:

1. Right click the com.ibm.itso.sg247004.DominoSessionPool package and select New → Class.
2. Name this class ClientPoolTester. Be sure you select the checkbox that creates the main() method stub. Implement the Runnable interface as shown in Figure 6-20.

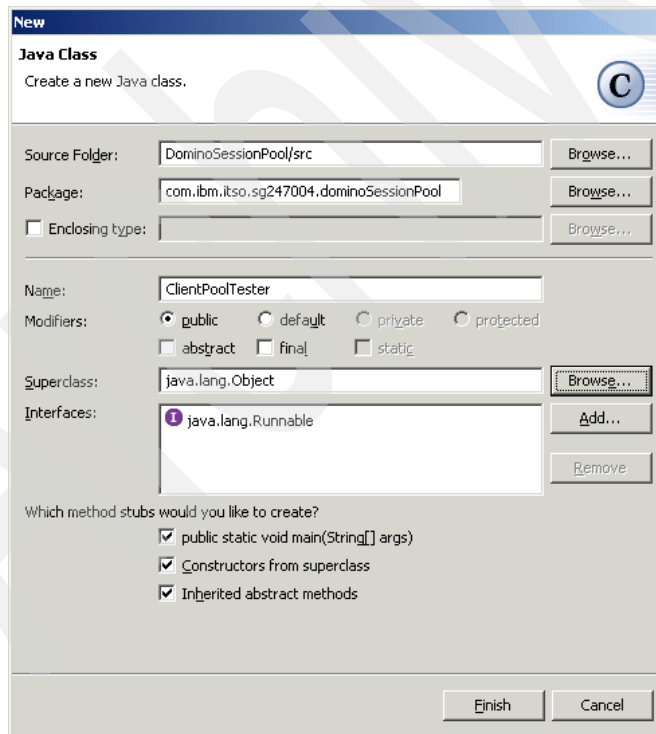


Figure 6-20 Creating the ClientPoolTester class

3. The ClientPoolTester class should open. Insert the following import statements:

```
import org.apache.commons.pool.*;
import org.apache.commons.pool.impl.*;
import lotus.domino.*;
```

4. Add the following class variables to the class:

```
public ObjectPool pool; // The pool that will manage our sessions
public static String mode = "pooled"; //Can be pooled or not pooled
public static long totalTime = 0; // This will keep the timer
```

5. Modify the default constructor to the following code:

```
public ClientPoolTester() {
    super();
    PoolableObjectFactory factory = new DominoSessionFactory();
    pool = new StackObjectPool(factory, 5);
}
```

As you can see, you are creating a factory based on your previously created class and then you are creating an ObjectPool that will stack 5 Domino Sessions.

6. Modify the run() method to contain the following code:

```
public void run() {
    Session mySession = null;
    try {
        long timer = System.currentTimeMillis();
        if (mode.equals("pooled")) {
            // The session is borrowed from the pool
            mySession = (Session) pool.borrowObject();
        } else {
            // A new session is created from scratch
            mySession =
                NotesFactory.createSession(
                    "<servername>", "<username>", "<password>");
        }
        // Now we will print some session information
        System.out.println("Domino Server: " + mySession.getServerName());
        Database db = mySession.getDatabase(
            mySession.getServerName(), "names.nsf");
        View view=db.getView("People");
        // We will full text search the People view and print results
        System.out.println("Found "+view.FTSearch(
            "FIELD LastName contains Admin")+ " number of persons
            with an Admin Lastname...");
        if (mode.equals("pooled")) {
            // Now we will return the connection to the pool
            pool.returnObject(mySession);
        }
    }
}
```

```

    } else {
        // Here we are recycling the session
        mySession.recycle();
    }
    System.out.println("Running thread took: "
        + (System.currentTimeMillis() - timer)
        + " millisecs\n");
    // Keeping the timer on every run...
    totalTime += System.currentTimeMillis() - timer;
} catch (NotesException e) {
    Exception ex = e;
    System.err.println(e.getClass().getName() + ": " + e.text);
    ex.printStackTrace();
} catch (Exception e) {
    System.err.println(e);
    e.printStackTrace();
}
}
}

```

As you can see, this `run()` method basically can run in two modes using pooled or non-pooled requests. If the mode equals pooled, the class will borrow a Domino Session from the pool and connect it to the `names.nsf` database and perform a full text index on the People view.

If the mode equals non pooled, it will create a session and after performing the same tasks will recycle the session.

Important: You should create a Full Text Index for the `names.nsf` database, if there isn't one already, to run this example.

Also note that the `totalTime` variable is incremented by the execution of each `run()` method.

7. Have the `ClientPoolTester` class implement the interface `Runnable` so that you can start several threads of this class and simulate a sample serialized access load into the pool.

Modify the `main()` method to the following code:

```

public static void main(String[] args) {
    try {
        ClientPoolTester t = new ClientPoolTester();
        // We will create 5 threads of this class
        Thread nt = new Thread((Runnable) t);
        Thread nt2 = new Thread((Runnable) t);
        Thread nt3 = new Thread((Runnable) t);
        Thread nt4 = new Thread((Runnable) t);
        Thread nt5 = new Thread((Runnable) t);
        // Now we will start each thread and put each thread to sleep
    }
}

```

```

        // this will help stabilize the pool.
        nt.start();
        nt.sleep(400);
        // This sleep emulates work and stabilizes pool for access
        nt2.start();
        nt2.sleep(100); //This sleep emulates work
        nt3.start();
        nt3.sleep(100); //This sleep emulates work
        nt4.start();
        nt4.sleep(100); //This sleep emulates work
        nt5.start();
        nt5.sleep(100); //This sleep emulates work
        // Finally wait for the thread to die
        nt.join();
        nt2.join();
        nt3.join();
        nt4.join();
        nt5.join();
        // Now we will close the pool and the session inside
        t.pool.close();
        // Print the total time to do the job.
        System.out.println("*****");
        System.out.println(" TOTAL TIME: " + totalTime);
        System.out.println("*****");
    } catch (Exception e) {
        System.err.println(e);
        e.printStackTrace();
    }
}

```

The `main()` method basically creates five threads and starts each one of them. Every thread will start the `run()` method, and finally, the total time it takes to perform the five tasks is printed.

This approach to the Domino Session pool is extremely basic. There are other important considerations, not covered by this example, that should be included in an enterprise implementation. Among these considerations are:

- ▶ **Managing the DIIOP time out.**

Domino Sessions should be pinged frequently to avoid timing out of the session and losing the reference. In addition, you should recycle the session after some time to avoid possible memory leaks.

- ▶ **User management.**

The credentials used to authenticate to the Domino server are fixed, depending on each portlet use case. You could group different types of sessions based on user profiles.

- ▶ Pooling other objects, like Domino Views.
The Domino Session can be an expensive object to instantiate, so the pool will help make your portlet application perform better. There also are other objects that can be expensive to instantiate, like the View object, so you might consider pooling this object as well.
- ▶ Handling of orphan sessions.
- ▶ How the pool can grow in a controlled manner.
- ▶ Other standard pool management characteristics. Consult the Jakarta Commons pool documentation for more information.

Analysis of the two approaches

In the lab, We ran the client in an Ethernet 100mbps network with the pool and with the create-destroy approach.

Note: This isn't a statistically valid analysis since we ran only 10 tests, and the testing environment wasn't a dedicated one. However, it does serve to illustrate, anecdotally, the fact that pools improve the performance of your application.

The results of the pooled approach are shown in Figure 6-21.

```

Java - IBM WebSphere Studio Application Developer
File Edit Source Refactor Navigate Search Project Profile Run Window Help
Console [ <terminated> C:\Program Files\IBM\WSADV5\eclipse\re\bin\javaw.exe (5/27/03 8:52 AM
DomSessFact: Creating Domino Session...
DomSessFact: Domino Session successfully created !
Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 491 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 170 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 170 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 171 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 170 miliseconds

DomSessFact: Destroying Domino Session...
DomSessFact: Domino Session destroyed successfully !
*****
TOTAL TIME: 1172
*****

```

Figure 6-21 Results from the pooled approach

The results of running the same client without using the pool are shown in Figure 6-22.

```

Java - IBM WebSphere Studio Application Developer
File Edit Source Refactor Navigate Search Project Profile Run Window Help
Console [<terminated> C:\Program Files\IBM\WSADV5\eclipse\jre\bin\javaw.exe (5/27/03)
Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Domino Server: CN=itsotest-dom/O=itsoportall
Running thread took: 1182 miliseconds

Found 1 number of persons with an Admin Lastname...
Running thread took: 1022 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Found 1 number of persons with an Admin Lastname...
Running thread took: 1021 miliseconds

Domino Server: CN=itsotest-dom/O=itsoportall
Running thread took: 1021 miliseconds

Found 1 number of persons with an Admin Lastname...
Running thread took: 1032 miliseconds

*****
TOTAL TIME: 5288
*****

```

Figure 6-22 Results from the non-pooled approach

We ran the test 10 times, and recorded the results shown in Table 6-2.

Table 6-2 Performance between pooled and non-pooled approaches

Run	Pooled approach (msecs)	Non-pooled approach (msecs)
Test 1	1172	5288
Test 2	1262	5298
Test 3	1302	5318
Test 4	1833	5356
Test 5	1232	5347
Test 6	1234	5368
Test 7	1241	5347
Test 8	1222	5307
Test 9	1202	5306
Test 10	1272	5317
AVERAGE TIME	1297	5325

The performance improvements were impressive in our tests: we had a 411% improvement when using the pooled approach. The test results are presented in Figure 6-23.

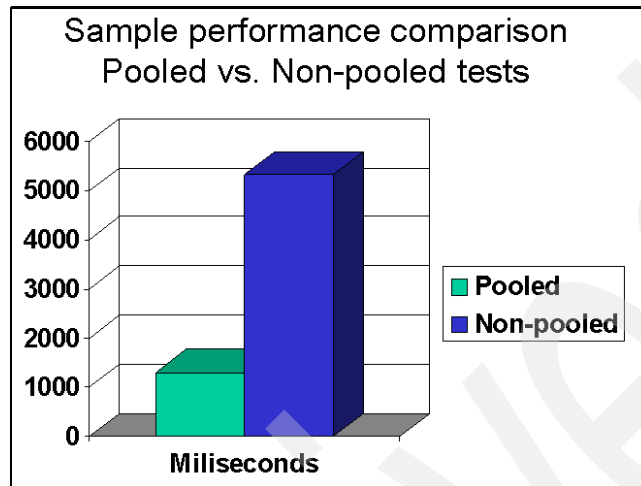


Figure 6-23 Performance comparison: Pooled and non-pooled

6.9 Reference material, links, Redbooks

- ▶ The Lotus Domino Toolkit for Java/CORBA / Domino Java API
<http://www-10.lotus.com/ldd/toolkits>
- ▶ CORBA
<http://www.omg.orgb>
- ▶ WebSphere Portal InfoCenter
<http://publib.boulder.ibm.com/pvc/wp/current/index.html>
- ▶ WebSphere EveryPlace Access InfoCenter
http://www-3.ibm.com/software/pervasive/products/library/ws_everyplace_access.shtml
- ▶ *Portlet Development Guide*
<ftp://ftp.software.ibm.com/software/webserver/portal/V41PortletDevelopmentGuide.pdf>
- ▶ *Portlet Best Practices Guide*
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/portletcodingguidelines.html>

- ▶ Portlet API
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/4.1api/>
- ▶ Portlet JSP Tag Library Syntax
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/V41jsptaglib.html>
- ▶ Portal Zone
<http://www7b.software.ibm.com/wsdd/zones/portal/>
- ▶ Portal Struts support
OpenSource: <http://jakarta.apache.org/struts/index.html>
Enablement package:
<http://www-3.ibm.com/software/webservers/portal/portlet/catalog/action/ChangePage/.pg/74/.reqid/3?viewPage=detail&NAVCODE=1WP10003N>
Struts in WebSphere Portal 4.1:
<http://www.ibm.com/support/docview.wss?rs=0&org=SW&doc=7002247>
- ▶ Log4j
<http://jakarta.apache.org/log4j/docs/>
- ▶ Object Pooling
<http://jakarta.apache.org/commons/pools>
- ▶ Click to Action
 - Enablement package:
<http://www-3.ibm.com/software/webservers/portal/portlet/catalog/action/ChangePage/.pg/74/.reqid/1?viewPage=detail&NAVCODE=1WP10003L>
 - *Using Click to Action to Provide User-Controlled Integration of Portlets*
http://www7b.software.ibm.com/wsdd/library/techarticles/0212_roy/roy.html
- ▶ People Awareness
 - IBM Redpaper: *WebSphere Portal 4.12 Collaboration Services*
<http://w3.itso.ibm.com/redpieces/abstracts/redp0319.html>
 - Lotus Collaborative Services JavaDoc
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/4.2api/collaborative/>

- ▶ *Building a Portlet within the Model-View-Controller Paradigm using WebSphere Portal*
http://www7b.software.ibm.com/wsdd/library/techarticles/0210_kwong/kwong.html
- ▶ *WebSphere Portal Programming: Portal Aggregation for Pervasive Devices*
http://www7b.software.ibm.com/wsdd/techjournal/0210_godwin/godwin.html
- ▶ *Hello World Portlet Revisited: Adding Globalization Support for Multi-languages Using WebSphere Portal 4.1.2*
http://www7b.software.ibm.com/wsdd/library/techarticles/0210_xu/xu.html
- ▶ *WebSphere Portal V4 programming, Part 1: Portlet application programming*
<http://www-106.ibm.com/developerworks/library/i-portalv4/?n-dd-8222>
- ▶ *WebSphere Portal V4 programming, Part 2: Portlet application programming*
<http://www-106.ibm.com/developerworks/library/i-portal2v4/?n-dd-8222>
- ▶ *WebSphere Portal Programming: Pervasive Portlet Development*
http://www7b.software.ibm.com/wsdd/techjournal/0207_wanderski/wanderski.html

Portlet builders

Portlet building technologies, including the IBM Application Portlet Builder Version 4.2 and those of various IBM Business Partners, promise to provide the capability to rapidly create portlets that can access and manipulate Domino applications without requiring in-depth J2EE programming knowledge. In this chapter we take a look at five portlet building tools that are currently available. We present detailed information on IBM Portlet Builder for Domino, Bowstreet Portlet Factory for WebSphere, and Conet Portlet Factory for Domino. In addition we briefly present the capabilities of Sofor Interactive Portlet Builder for Domino and Aprix Portlet Connector.

7.1 Overview of the portlet builders option

This chapter outlines the technologies available in the marketplace that bridge the gap between the use of existing portlets and custom Java development options. It focuses on the technology offerings from five vendors including IBM, Bowstreet, Conet, Sofor, and Aprix. All offer solutions that simplify the building of custom portlets to expose Domino applications on the WebSphere Portal platform. These builders utilize your Domino applications by querying and aggregating Domino data and displaying that content within a portlet context.

This chapter differentiates these various offerings, as well as providing key information about each portlet builder including features, pricing information, and advantages as well as disadvantages.

Why choose this option

While using existing portlets to expose Domino data is by far the most expedient option, it is only feasible when your portlet requirements match the functionality provided by an existing portlet.

Custom portlet development using Java and JSPs provides a much richer set of development options. There is no limit to what can be developed using the tools and techniques available from these options. If your team has the right development skills and enough time, custom portlet development is the way to go.

However, there are often situations where the skill set of the development team or the time allotted for development can prevent custom portlet development using Java and JSP tags from being a viable option.

Portlet builder technologies offer a middle-ground approach to portlet development. They offer significantly more development capabilities than the use of existing portlets. In addition, they promise a shorter development time and require less in-depth knowledge about custom portlet development than the more advanced Java- and JSP-based options.

7.2 IBM Portlet Builder for Domino

Overview

IBM WebSphere Portal Application Integrator is available for use with WebSphere Portal for no additional charge. The IBM WebSphere Portal Application Integrator package includes IBM Portlet Builder for Domino and is available for download from the WebSphere Portlet Catalog.

IBM Portlet Builder for Domino provides developers, administrators, and power users with the ability to easily create a portlet-based interface into an existing Domino application. No Java or other development skills are required.

Note: In addition to Domino, IBM WebSphere Portal Application Integrator provides configurable portlets used to access many other back-end systems, including:

- ▶ PeopleSoft
- ▶ SAP
- ▶ Siebel
- ▶ JDBC to relational databases
- ▶ WebSphere Portal Content Publishing

IBM Portlet Builder for Domino provides a highly configurable interface with a much richer set of capabilities than the Domino portlets available out-of-the-box with WebSphere Portal. The capabilities of IBM Portlet Builder for Domino include:

- ▶ Connects to any Domino database.
- ▶ No changes to existing Domino databases required.
- ▶ Click to Action enabled.
- ▶ Presence awareness (using Sametime).
- ▶ Offline browsing support.
- ▶ Mobile support for browsers and devices capable of rendering HTML and WML markup.
- ▶ Attachment support.
- ▶ Sortable columns.
- ▶ View search.
- ▶ Output WAR files are immediately available for re-deployment on another WebSphere Portal server.
- ▶ Select multiple views and forms in a Domino database to customize via portlet builder.
- ▶ Domino views are customized by selecting what columns to display.
- ▶ Domino forms are customized by selecting what fields to display.
- ▶ Domino views are selected by the users from a list at runtime.
- ▶ View columns are resizable.

- ▶ Documents selected from a view are either rendered within the same portlet (take over portlet, or render beside view) or through a separate document viewer portlet.
- ▶ The Document viewer portlet can render the document either as customized (through its associated form) or through an IFRAME (as the original document from Domino).
- ▶ Documents can be updated and new ones created through either the iFRAME viewer or the “data” viewer.

The advantages and disadvantages of using IBM Portlet Builder for Domino are similar to those of our first option, using existing portlets. Specifically, if this tool meets your application’s needs, it is much cheaper and faster than a custom development effort. However, if your application has requirements not available from IBM Portlet Builder for Domino, there is no way to extend this tool with customized functionality.

IBM Portlet Builder for Domino uses IIOP over the HTTP transport to communicate with each Domino server.

Implementation issues

- ▶ **Applicable Portlet patterns**
IBM Portlet Builder for Domino is well suited for the Display pattern. It is also capable of implementing a simple Integrated pattern.
- ▶ **Development time**
Very low development time is required.
- ▶ **Developer skill set**
Basic portal configuration skills. No familiarity with Java or Portlet development is required.
- ▶ **Range of applications**
The IBM Portlet Builder portlet is highly configurable, and provides many more options than the Domino portlets currently shipping with WebSphere portal. However, its range of applications is limited by the fixed set of functionality and interface options. If these options do not meet your application’s needs, there is currently no way to extend this interface.
- ▶ **Rich text handling**
If the form is integrated into the portlet content, rich text fields are rendered as plain text.

Note that it is possible to configure an IBM Portal Builder portlet to open documents using an IFRAME. Using this option, documents will be rendered

using Domino server's HTTP engine and can therefore leverage all of the Web-based rich text functionality provided by Domino.

Performance

- ▶ Session management
IBM Portlet Builder does not currently support session pooling.
- ▶ Clustering
IBM Portlet Builder does not natively support server fail-over in a clustered Domino environment.
- ▶ Scalability
No data is currently available for IBM Portlet Builder scalability.
- ▶ Single sign-on support
Single sign-on is supported, but not required for IBM Portlet Builder for Domino. Many authentication options are available, including:
 - LTPA
 - Credentials Vault
 - Prompt for a user ID and password
 - Use an administrator-specified ID and password

Required software versions

- ▶ IBM/Lotus Domino R5 or 6
- ▶ WebSphere Portal v4.1, 4.2x
- ▶ WebSphere Portal Express v4.1, 4.2x

7.2.1 Implementation details

Installing IBM Portlet Builder for Domino

Note: Installation instructions may change from version to version. For the most accurate installation instructions, use the documentation provided with your version of Portlet Builder.

Download the installation package from the WebSphere Portlet Catalog and expand the zip file to a temporary folder. The folder now contains the files needed for installation. Use the following steps to install the Portlet Builder.

1. To install Portlet Builder for Domino, simply install the two WAR product files:
 - BOBuilderPortlet.WAR
 - DominoStruts.WAR

Figure 7-1 and Figure 7-2 illustrate this. For detailed instructions on installing WAR files, see 2.4.1, “Install portlets” on page 54.

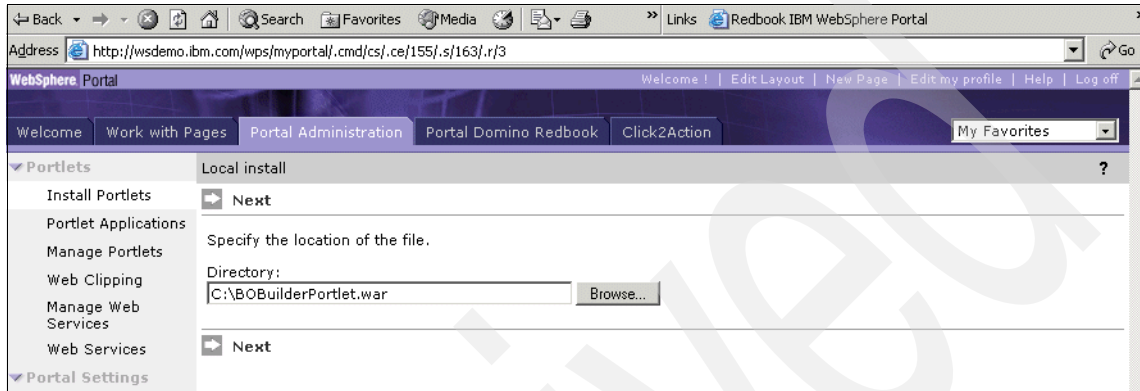


Figure 7-1 Installing the BOBuilderPortlet.WAR for IBM Portlet Builder

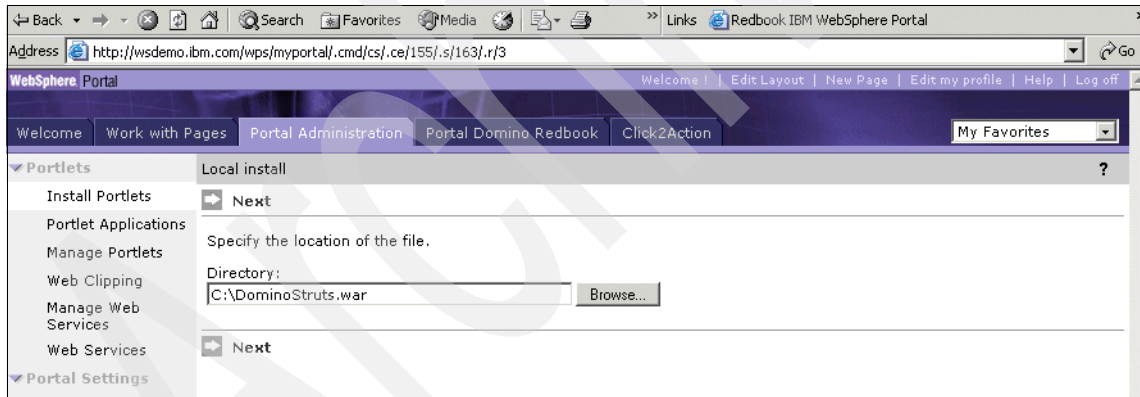


Figure 7-2 Installing the DominoStruts.WAR for IBM Portlet Builder

2. After installing the WAR files, create a new page and add the Portlet Builder for Domino Portlet to this page.

Figure 7-3 illustrates this. For detailed instructions on creating a page and adding a portlet to the page, see 2.4.4, “Adding portlets to a page” on page 58.

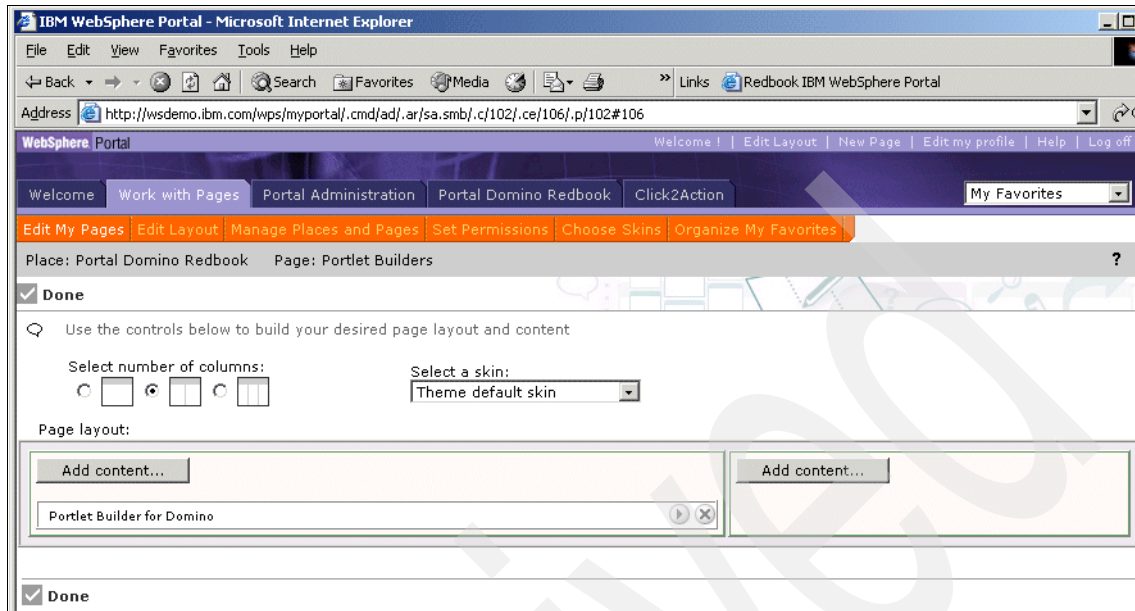


Figure 7-3 Adding the Portlet Builder for Domino portlet to a page

Creating a new Portlet with Portlet Builder for Domino

Follow these steps to create and configure a portlet using IBM Portlet Builder for Domino.

1. Open the page containing the Portlet Builder for Domino portlet, as shown in Figure 7-4. Click the Create new portlet button.

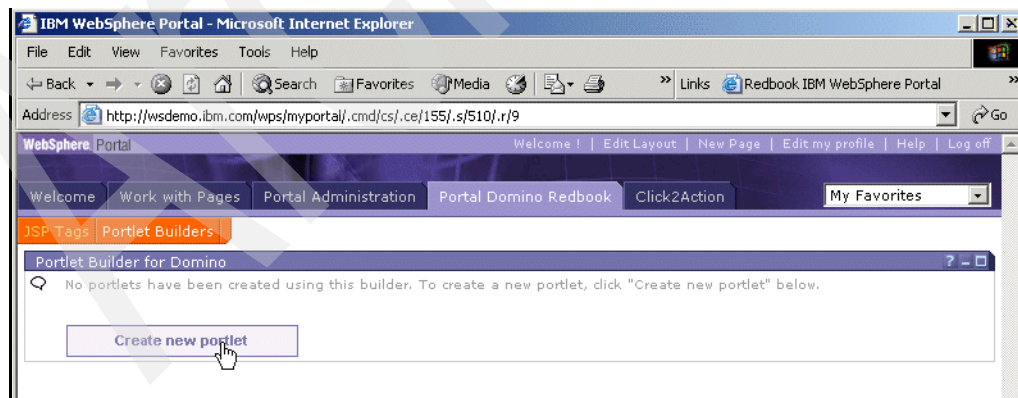


Figure 7-4 Accessing the Portlet Builder for Domino portlet

2. Enter a Portlet name and specify your Domino server name, as shown in Figure 7-5. When finished, click the Connect to server button.

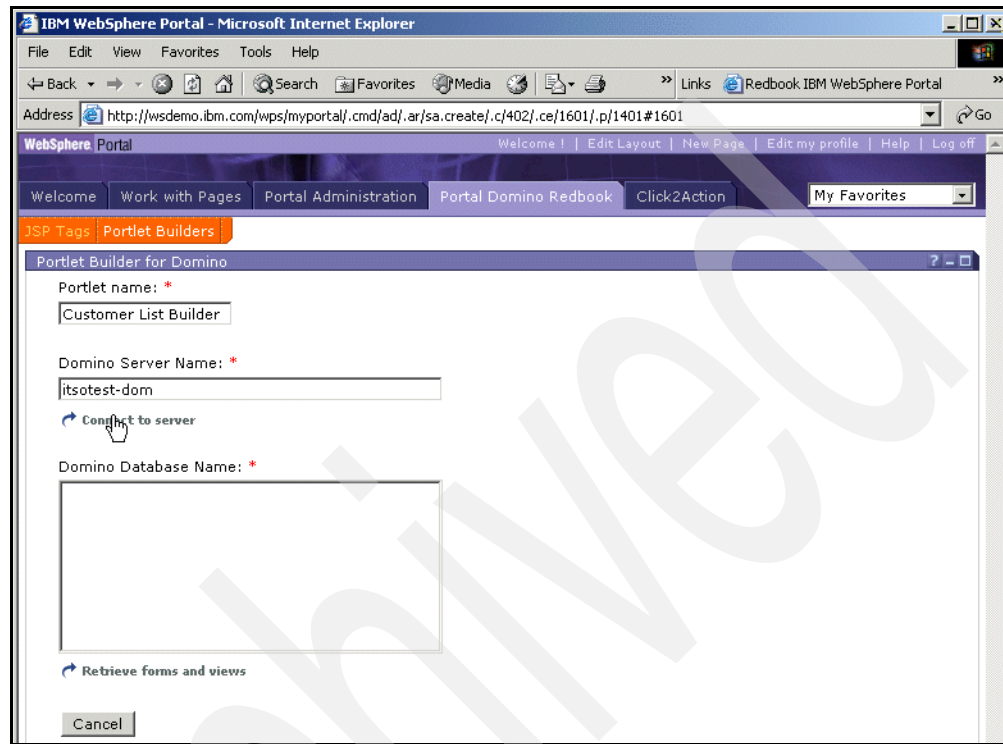


Figure 7-5 Configuring the Portlet Builder for Domino: Enter portlet name and server name

3. Enter a valid Domino username and password with access rights to the server and database you wish to access, as shown in Figure 7-6. When finished, click OK.

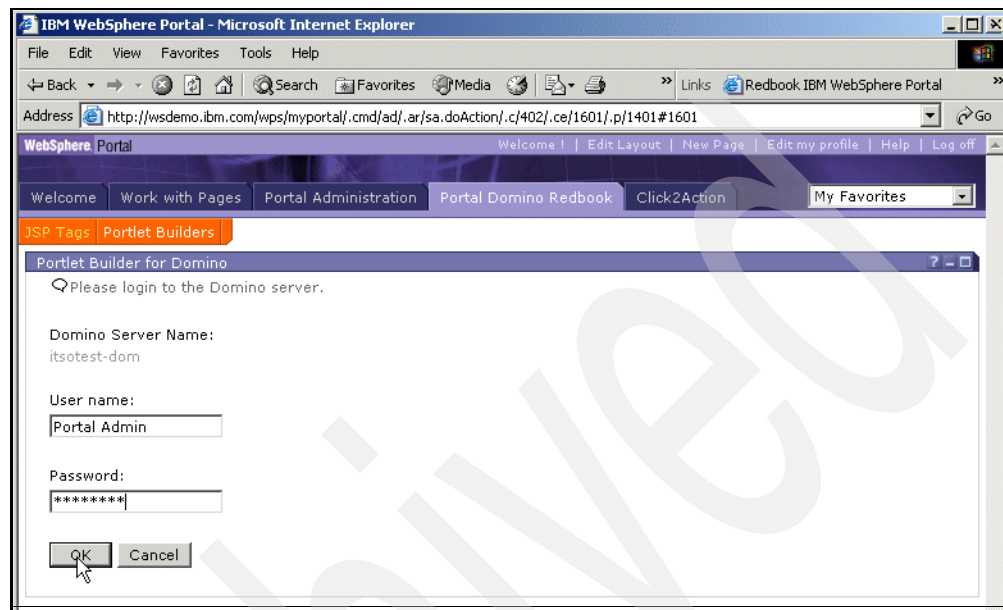


Figure 7-6 Configuring the Portlet Builder for Domino: Enter username and password

4. Select the filename of the Domino database you wish to access, as shown in Figure 7-7. When finished, click Retrieve forms and views.

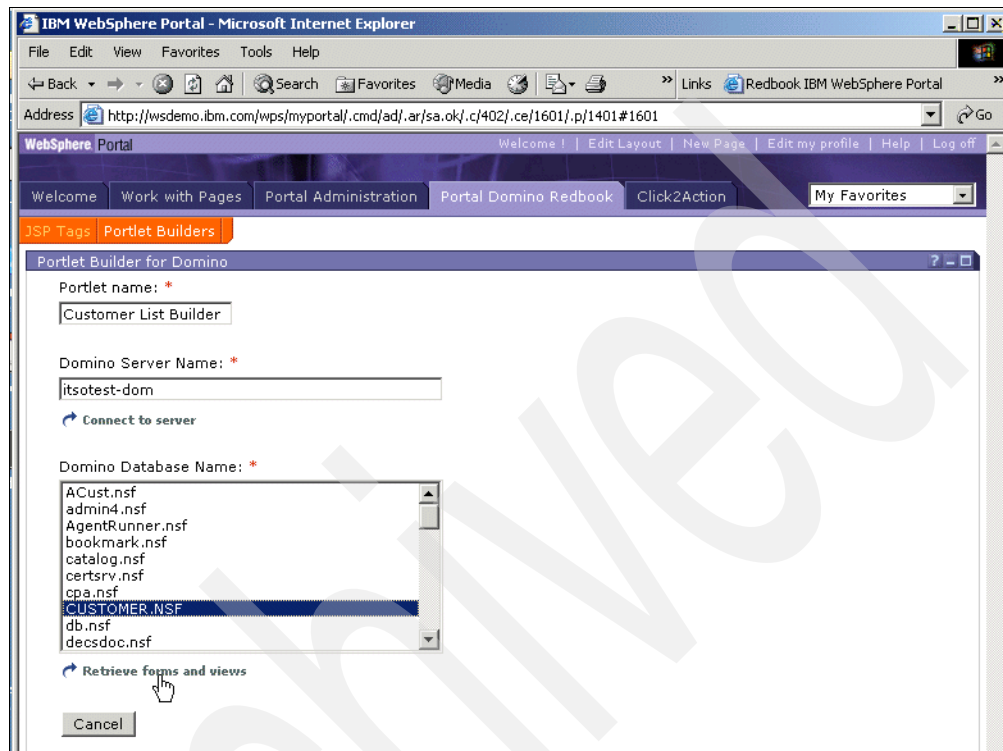


Figure 7-7 Configuring the Portlet Builder for Domino: Select a database

5. Select the forms and views you would like to make available in this portlet, as shown in Figure 7-8. If there are a large number of views and forms, you can browse through the list, page by page, using the navigation buttons. When finished, click Next.

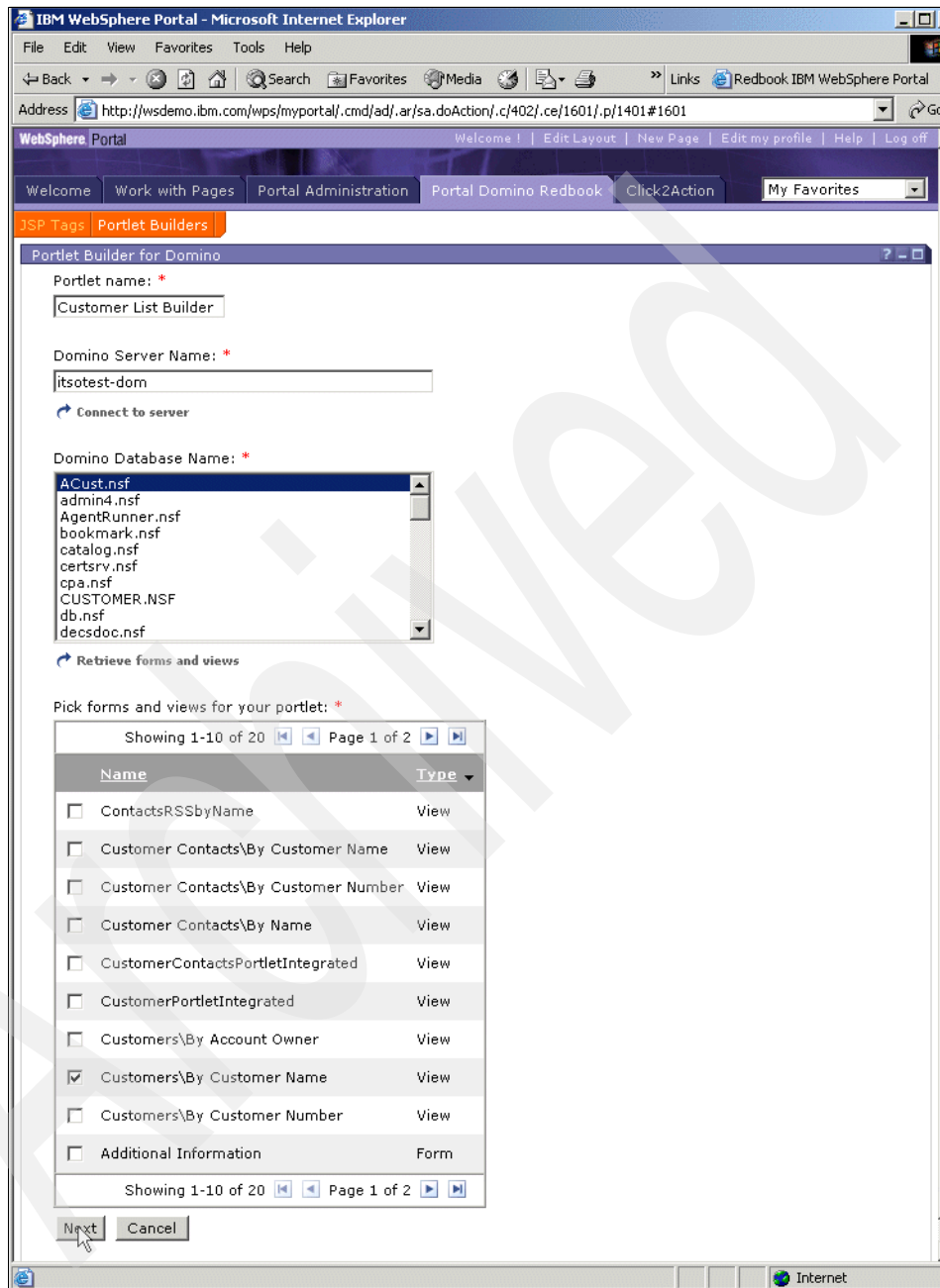


Figure 7-8 Configuring the Portlet Builder for Domino: Select the forms and views

6. You will be taken to the form and view options configuration page shown in Figure 7-9.

From here you are able to:

- Configure the portlet settings for each form and view to be displayed.
- Specify the portlet authentication options.
- Specify the form display options.

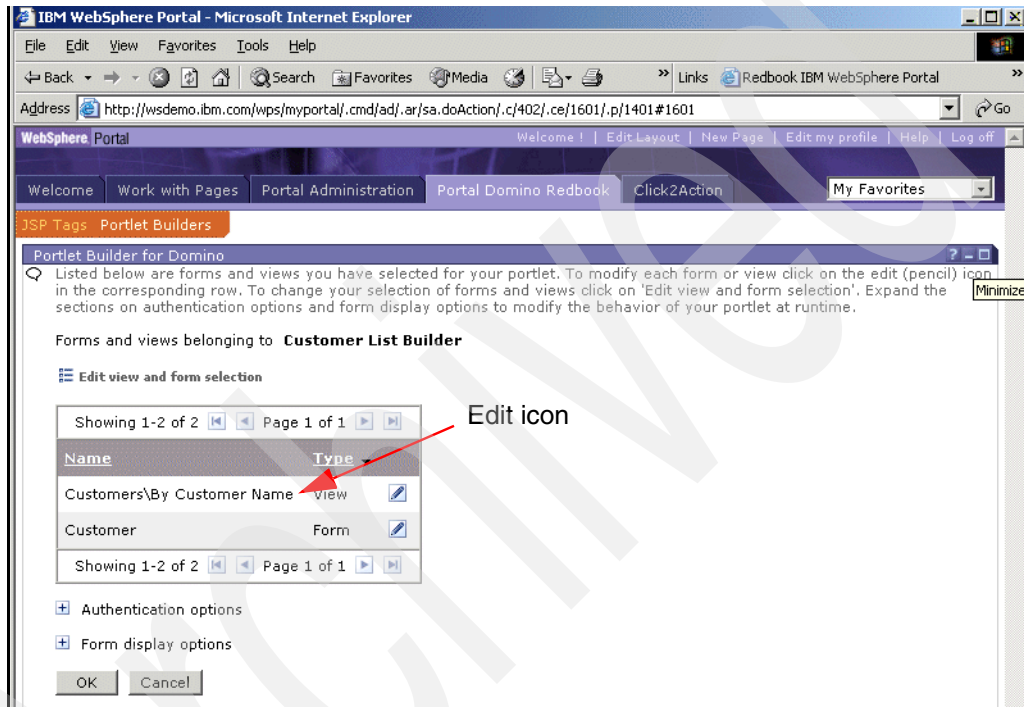


Figure 7-9 Configuring the Portlet Builder for Domino: Form and view options configuration page

7. From the form and view options configuration page, click the Customers\By Customer Name view edit icon. You will be taken to the view configuration page, as shown in Figure 7-10. Here, you are able to control what columns are displayed, the column labels, if a column is searchable, and many other options. On the configuration page you are also able to specify Click to Action parameters and specify which columns are enabled for people awareness.

Now, configure the view to your liking and click Next.

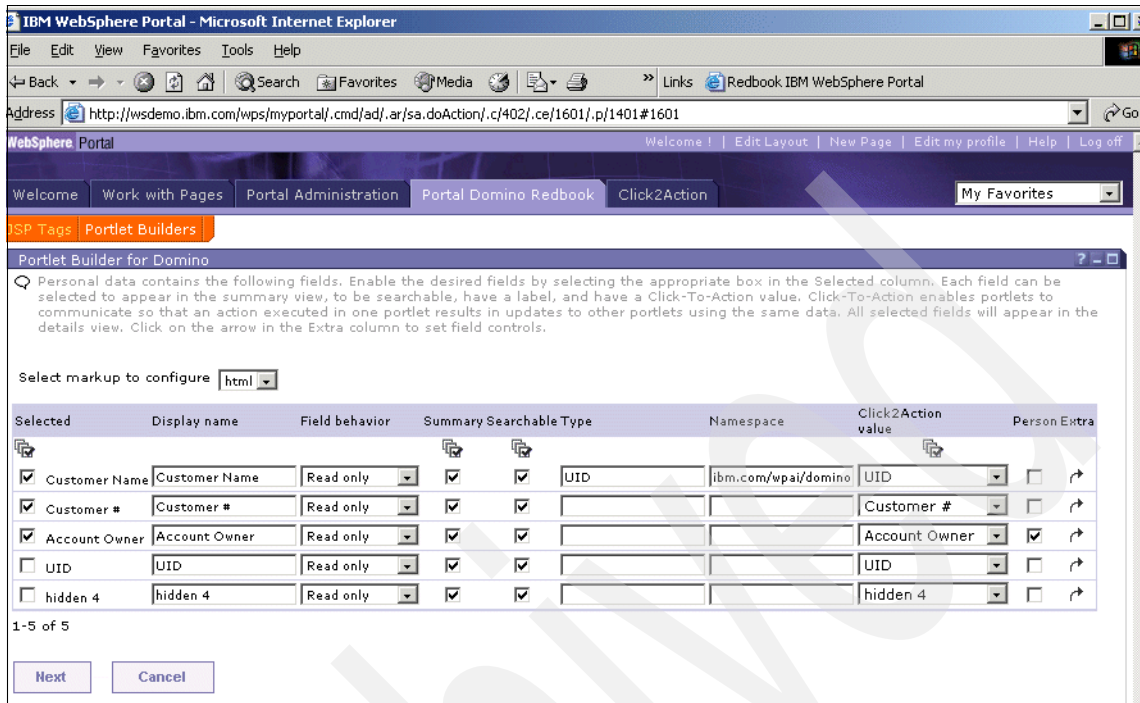


Figure 7-10 Configuring the Portlet Builder for Domino: Configuring the CustomersBy Name view

- You can now control the order in which the columns are displayed within the portlet, using the interface shown in Figure 7-11. The buttons with up or down triangles change the column order accordingly. Adjust the column order to your liking and click Finish.

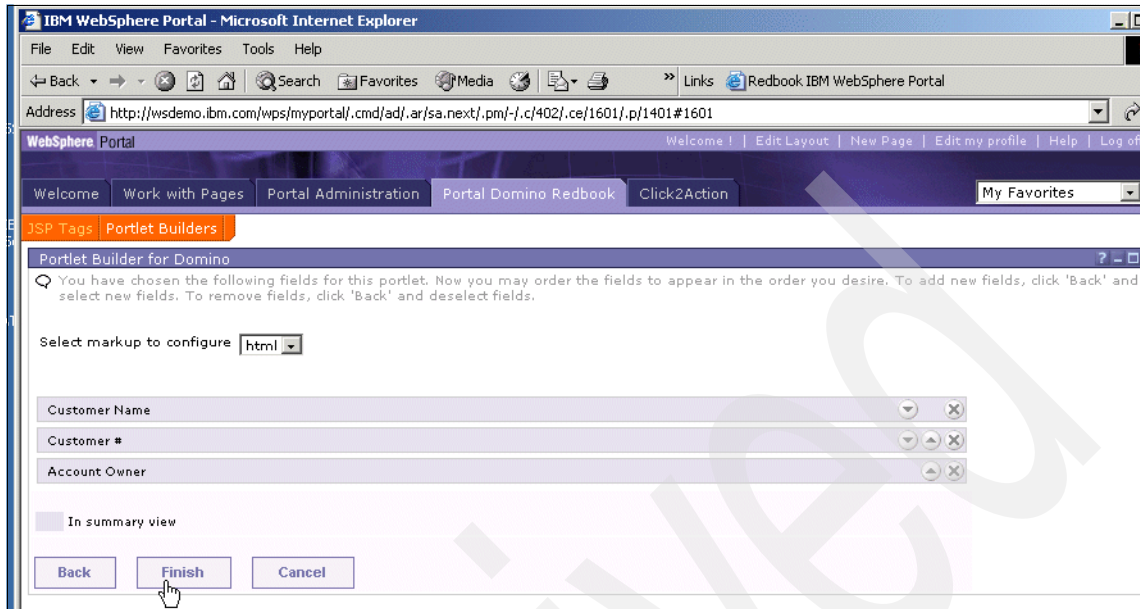


Figure 7-11 Configuring the Portlet Builder for Domino: Ordering the Customers\By Name columns

9. You are taken back to the form and view options configuration page. You can configure another form or view by repeating steps seven and eight. You can also expand the Authentication Options and Form Display Options sections, as shown in Figure 7-12. The options available are as follows:

Authentication options

- Use single sign-on: The portlet will use the user's LTPA token to authenticate with Domino.
- Prompt for user ID and password: The portlet will prompt the user for their Domino ID and password.
- Use this user ID and password: The portlet will always use the ID and password specified here.
- Use existing credential vault slot: The portlet will authenticate using the ID and password in the specified slot of the authenticated user's credential vault.

Form display options

- Use data form: The portlet will display forms using the Portlet Builder's simple data form interface, as configured for each form using steps seven and eight. While these forms will be embedded seamless inside the portlet, they have very limited functionality and display capabilities.

- Use Inline frame: The portlet will display forms within an IFrame, using Domino server's native HTML rendering capabilities. If IFrames are acceptable within your portlet, this will provide the quickest and easiest way to implement full form functionality within your portlet. For some functionality, such as for displaying rich text, using IFrames is the only option.

Once you have finished configuring the portlet options, click OK.

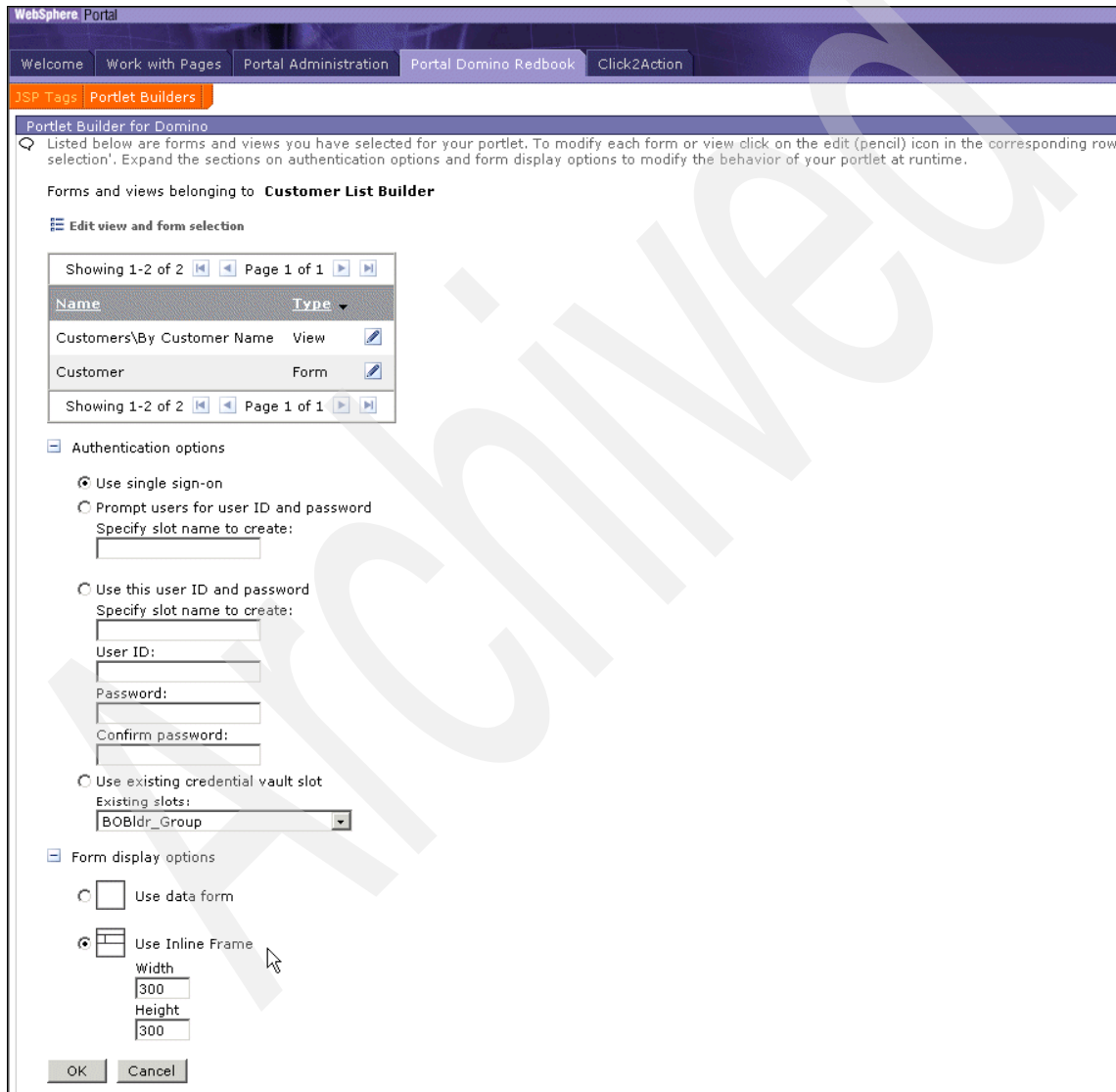


Figure 7-12 Configuring the Portlet Builder for Domino: Authentication and form display options

10. You should now see your newly created portlet listed within the Portlet Builder for Domino portlet, as shown in Figure 7-13. You can reconfigure this portlet at any time.

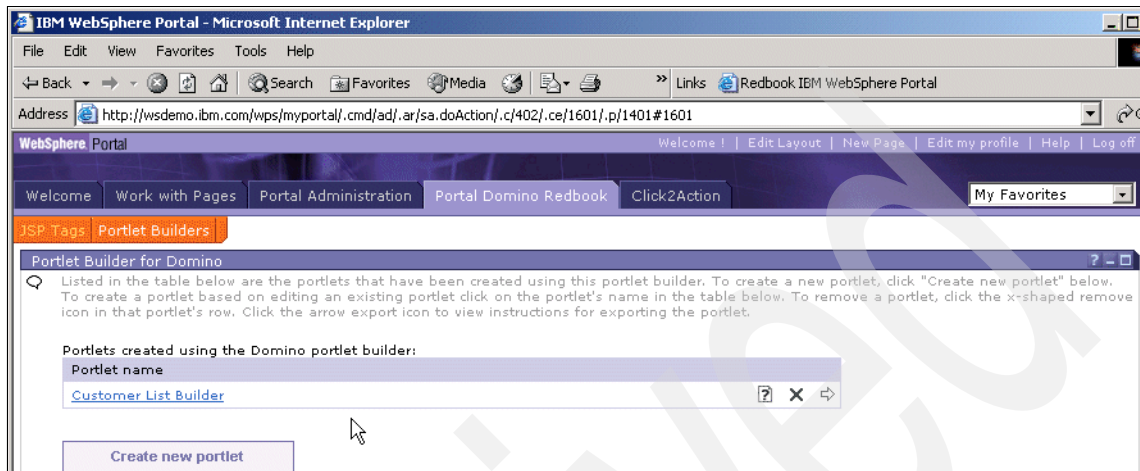


Figure 7-13 Configuring the Portlet Builder for Domino: Completed portlet configuration

11. Add the newly created portlet to one of your pages, as shown in Figure 7-14. Steps describing how to add a portlet into a page are detailed in 2.4.4, "Adding portlets to a page" on page 58.

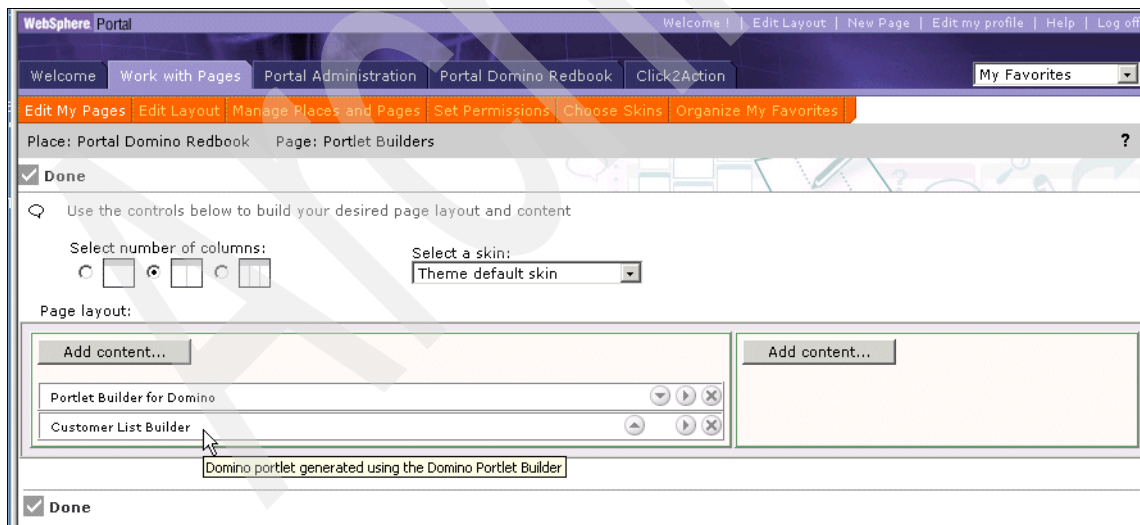
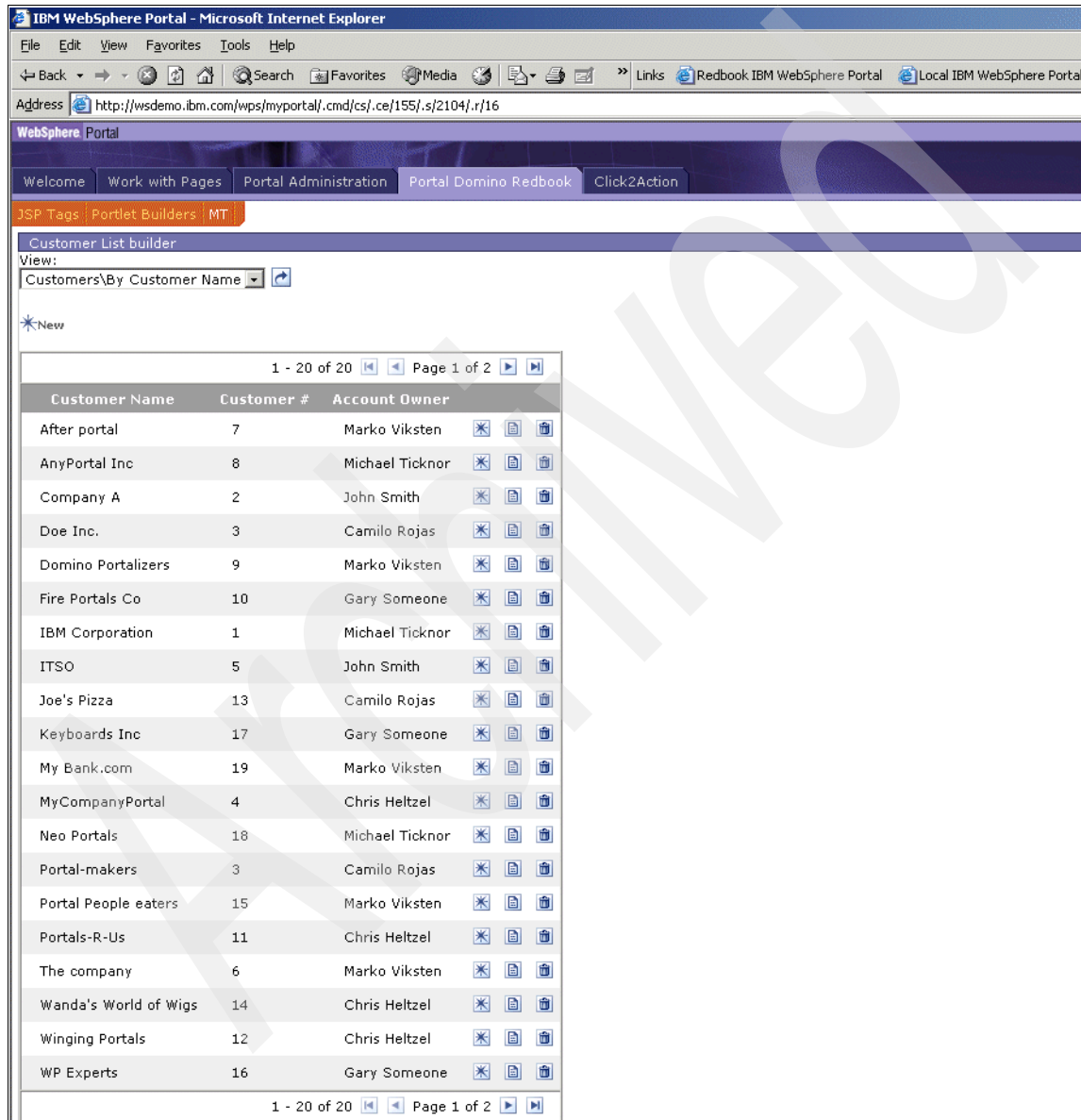


Figure 7-14 Configuring the Portlet Builder for Domino: Adding the newly created portlet to a page

7.2.2 Implementation example

Following steps 1 through 11 in the previous section, Figure 7-15 show the Customers\By Name view from our Case Study application, as rendered by IBM Portlet builder for Domino.



The screenshot shows a web browser window titled "IBM WebSphere Portal - Microsoft Internet Explorer". The address bar shows the URL: <http://wsdemo.ibm.com/wps/myportal/.cmd/cs/.ce/155/.s/2104/.r/16>. The page content includes a navigation bar with tabs: "Welcome", "Work with Pages", "Portal Administration", "Portal Domino Redbook", and "Click2Action". Below this is a sub-navigation bar with "JSP Tags" and "Portlet Builders - MT". The main content area is titled "Customer List builder" and shows a "View:" dropdown menu set to "Customers\By Customer Name". A "New" link is visible. The main content is a table with the following data:

Customer Name	Customer #	Account Owner			
After portal	7	Marko Viksten			
AnyPortal Inc	8	Michael Ticknor			
Company A	2	John Smith			
Doe Inc.	3	Camilo Rojas			
Domino Portalizers	9	Marko Viksten			
Fire Portals Co	10	Gary Someone			
IBM Corporation	1	Michael Ticknor			
ITSO	5	John Smith			
Joe's Pizza	13	Camilo Rojas			
Keyboards Inc	17	Gary Someone			
My Bank.com	19	Marko Viksten			
MyCompanyPortal	4	Chris Heltzel			
Neo Portals	18	Michael Ticknor			
Portal-makers	3	Camilo Rojas			
Portal People eaters	15	Marko Viksten			
Portals-R-Us	11	Chris Heltzel			
The company	6	Marko Viksten			
Wanda's World of Wigs	14	Chris Heltzel			
Winging Portals	12	Chris Heltzel			
WP Experts	16	Gary Someone			

Figure 7-15 The Customers/By Name view as rendered by IBM Portlet Builder for Domino

7.3 Bowstreet Portlet Factory for WebSphere

Overview

Bowstreet Portlet Factory for WebSphere is a framework and set of tools for rapidly creating and maintaining customized portlets for the WebSphere Portal environment. With Portlet Factory, developers build portlets by pulling together a sequence of reusable software components called *Builders*. Developers assemble Builders into *models*. These models are then executed at runtime to dynamically generate application code. The code generated includes JSPs, Java classes, XML documents, and all of the low-level artifacts required to create a portlet application. Thus, developers can capture and automate the process of building portlets instead of explicitly coding each portlet.

Bowstreet Portlet Factory for WebSphere comes with the Lotus Collaboration Extension. This is a set of builders (that is, tools) that assist the developer in building portlets to access Domino applications.

Bowstreet Portlet Factory is not targeted exclusively to Domino developers. In addition to the Lotus Collaboration extension, it includes a large number of tools for accessing relational databases, building stand-alone applications, and so forth.

A key strength of this tool is that it does not completely insulate a developer from the J2EE framework. If a specific task is not possible with the tools provided, a Java developer can code a method or class in Java and include this code in the Bowstreet development environment.

A detailed knowledge of Java and J2EE is not a requirement. However, Bowstreet Portlet Factory is very J2EE-centric. A basic understanding of Java and J2EE technologies will greatly increase a developer's productivity with these tools.

Bowstreet provides developers with:

- ▶ Ability to leverage existing Domino applications
- ▶ Ability to more easily create of custom portlets
- ▶ Robust personalization and customization capabilities
- ▶ Simplified portlet-to-portlet communication
- ▶ Categorization and search
- ▶ Many single sign-on options
- ▶ People awareness

Bowstreet development tool

Bowstreet Portlet Factory has a WebSphere Studio plug-in, Bowstreet Designer, used for creating, viewing, and running portlets. Bowstreet Designer plugs into the Eclipse and WebSphere Workbench IDEs.

Portlets created with the Bowstreet Portlet Factory follow a standard J2EE model-view-controller design. In order to simplify the task of writing custom portlet Java and JSP code, Bowstreet introduces the developer to a few key objects used to create a portlet application.

Builder

A *builder* is the core building block that automates design and development tasks performed by developers. Simply put, a builder is a collection of Java classes and XML documents that represent a specific pattern or high-level component of a Web application. A builder provides a wizard-like UI for gathering configuration information from the developer, as well as the code for rendering the pattern or context-aware elements within the Web application.

A simple builder might add a button to a JSP page, while another might render a Domino view. Builders can analyze the application and perform tasks in the context of what previous builders have created. For example, a “page navigation control” builder could reference a set of JSP pages and create a set of navigational controls relevant to the context of those pages. If a page changes, then the navigational controls update automatically in a ripple effect that can cascade through the entire application.

Model

A *model* is a sequenced collection of builders that generate the application components representing the behavior, structure, data, and presentation of the portlet application. Underneath the covers, a model is simply an XML file containing a series of calls to builders.

Each model can be turned into a portlet, or can be run as a stand-alone J2EE application.

Profile

A *profile* contains a set of inputs that vary the way a portlet behaves. Profile settings can be edited after a portlet is deployed by clicking the configuration icon for the portlet. A profile feeds values into builders based on user identity or other contextual information (such as day of the week). Using profiles, you can automatically generate different variations of a generic portlet (from the same model) for different users or situations.

Configurable profiles are very easy to implement with Bowstreet Portlet Factory.

Regeneration

When a model is regenerated, each builder in the model executes in sequence and creates pieces of your portlet, such as JSP pages or Java methods. During regeneration, profiles can feed different inputs to builders based on the user or situation, automatically creating custom portlets “on the fly.”

There is a negligible performance hit associated with regeneration; less than 1% of the total processing resources in a typical “regen-enabled” execution environment are spent performing regeneration. This is because generated objects are cached at optimal levels of granularity, as are sessions. Furthermore, it's possible to break a model into a hierarchy of components, whereby models are selectively regenerated.

WebApp

The *WebApp* is a profile-specific instance of a portlet application that is dynamically created by the factory regeneration engine. Each builder, when called during regeneration, creates objects that get woven into the portlet application, such as pages, buttons, variables, or methods. The regeneration engine creates the *WebApp* by combining the regeneration of a model with a unique instance of profile data. The *WebApp* objects are then processed by the factory's execution engine to instantiate the executable J2EE application sessions.

Bowstreet server components

Portlets created with Bowstreet Designer plug in to the WebSphere Portal and application server. The Portlet Factory's Automation Engine leverages WebSphere's HTTP stack as well as all of the services from WebSphere Application Server and WebSphere Portal, such as clustering, failover, J2EE security, and session management.

Bowstreet Portlet Factory integrates with the WebSphere Portal via the Bowstreet Portlet Adapter WAR. This is a standard WPS portlet WAR that includes the portlet factory classes (JAR files). When a request comes in from the portal, the Bowstreet Adapter Portlet class calls in to the portlet factory code at the layer below the servlet layer (the *WebAppRunner* class). Note that this WAR file is deployed into WebSphere Portal only once, eliminating the need to re-deploy a WAR with every iteration or variation.

Figure 7-16 on page 375 illustrates the code execution architecture.

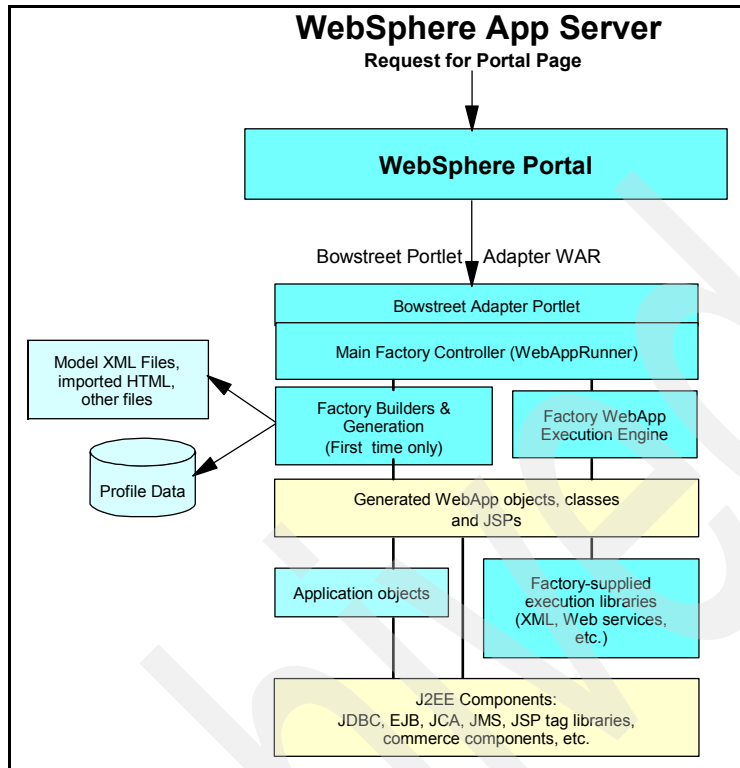


Figure 7-16 Bowstreet Portlet Factory for WebSphere code execution architecture

Implementation issues

► Applicable portlet patterns

Bowstreet Portlet Factory is capable of implementing all four portlet patterns:

- Link
- Display
- Integrated
- Migrated

► Development time

Moderate.

Development time can be significantly less than developing an application from scratch using the JSP and Java options, especially once developers become experienced with the Bowstreet development tools at their disposal.

Certain types of portlets require less time to create than others. For example, view navigation, document display (without rich text), and click to action functionality are significantly easier to develop with Bowstreet Portlet Factory.

Other functionality, such as document creation and editing, can be more challenging, especially when input validation, input translation, computed fields, or dynamic keyword values are involved.

- ▶ Developer skill set

In addition to learning the Bowstreet development tools and techniques, a developer should have good understanding of Domino, HTML, and JavaScript development. While not required, a solid understanding of Java, JSPs, and portal technologies is very beneficial when using some of the more advanced builder configuration options, or when implementing a task not easily performed by an available builder.

- ▶ Range of applications

Bowstreet offers a powerful set of builders for developing a rich portal interface with a high level of functionality. A great number of builders are available and more are being made available with each release. High-level builder functionality can be customized using a wide variety of configuration settings and by adding lower-level builders to your model.

In the event that a builder is not available to provide specific functionality, it is possible to write a custom builder, or add a builder that supports custom Java code. Both of these options, of course, require Java development skills and significantly more development time.

- ▶ Rich text handling

At the time of this writing, rich text fields cannot be viewed or edited using the forms generated by Bowstreet Portlet Factory.

Note that it is possible to configure the Bowstreet Domino view builder to open documents in a new browser window. Using this option, documents will be rendered using Domino's HTTP engine and can therefore leverage all of the Web-based rich text functionality provided by Domino.

Performance

Bowstreet has done extensive performance testing on portlets created with the Bowstreet Portlet Factory for WebSphere. The benchmark they made concludes that the Bowstreet Portlet Factory provides all the benefits of rapid creation of portlets without sacrificing performance. The portlets that the portlet factory generates exhibit performance characteristics (throughput, response time, and CPU utilization) that are highly comparable to traditional hand-coded portlets. Contact Bowstreet to obtain detailed information about performance.

Currently no metrics are available on performance related to portlets accessing Domino data. As with all integration techniques, performance testing is recommended before deploying a portal application using Bowstreet Portlet Factory.

- ▶ Session management

Bowstreet Portlet Factory can implement session pooling for connecting to Domino servers. This will significantly increase the scalability of Domino portal applications.

Note that the session pooling option can be used only when “Use regen credentials” is specified as the runtime credentials for connecting to Domino.

- ▶ Clustering

Bowstreet Portlet Factory does not natively support server fail-over in a clustered Domino environment.

- ▶ Requires single sign-on

Bowstreet supports a wide range of single sign-on options. It can be configured to use LPTA or the Credentials Vault. Also, by configuring a Bowstreet portlet with a valid Domino username and password, it can allow users to connect with Domino servers that do not support SSO.

Required software versions

- ▶ Lotus Domino 5.0.10 or later or 6.0 or later
- ▶ WebSphere Portal v4.1, 4.2 or WebSphere Portal Express 4.1
- ▶ Bowstreet Portlet Factory for WebSphere v5.6+
- ▶ Bowstreet Portlet Factory Lotus Collaboration Extension

Note: People Awareness requires WebSphere Portal Extend.

7.3.1 Implementation details

At a high level, development consists of three main steps:

1. Develop the model.
2. Test the model.
3. Deploy and test the model as a portlet.

Developing the model primarily involves adding and configuring builders. It may also involve the development of custom builders, or the insertion of custom Java code into a builder supporting this action.

Launching the Bowstreet Designer client

After installing and configuring a Bowstreet development environment, all development is performed within the Bowstreet Designer. The Bowstreet Designer client is contained within the WebSphere Studio Application Developer client.

To launch the Bowstreet Designer, open the WebSphere Studio Application Developer and select Bowstreet → New Model or Bowstreet → Open Model.

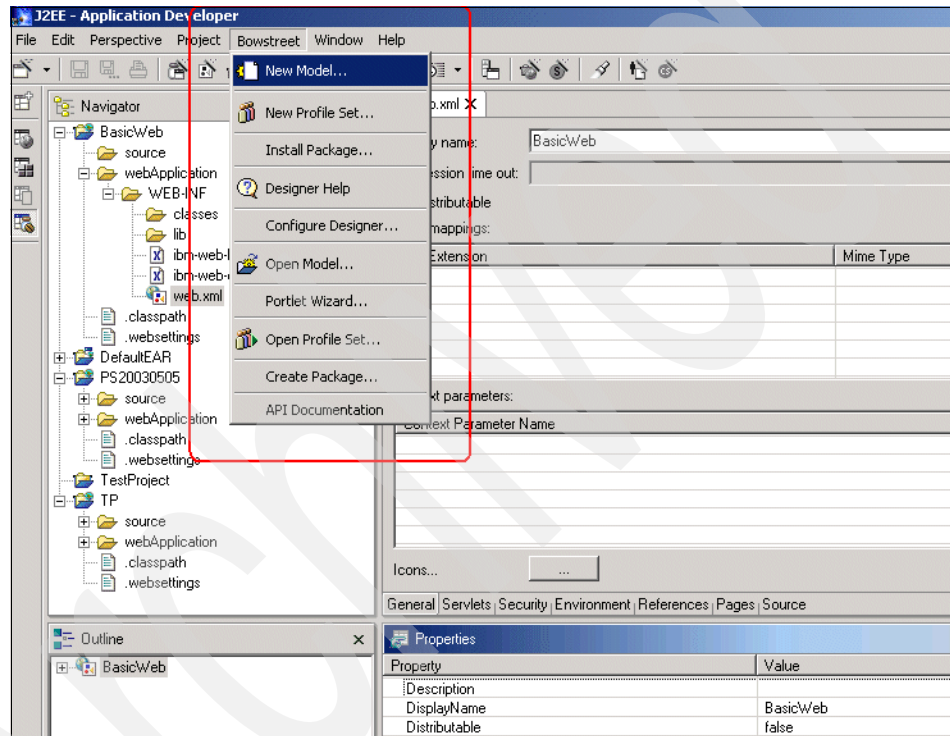


Figure 7-17 Launching the Bowstreet Designer from WebSphere Studio

Creating a new model

Use the following steps to create a new model:

1. After selecting Bowstreet → New Model, you are prompted to select the type of model to create. For Domino Portlets, it is generally best to start with an Empty model (Figure 7-18).

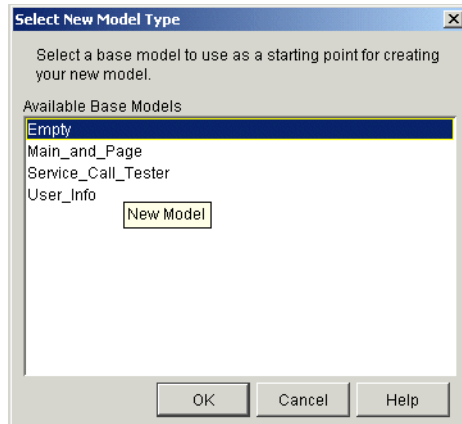


Figure 7-18 Creating a new Bowstreet model: Selecting model type

2. When prompted, enter a model file name and file path. It is best to keep related models in a common subdirectory.

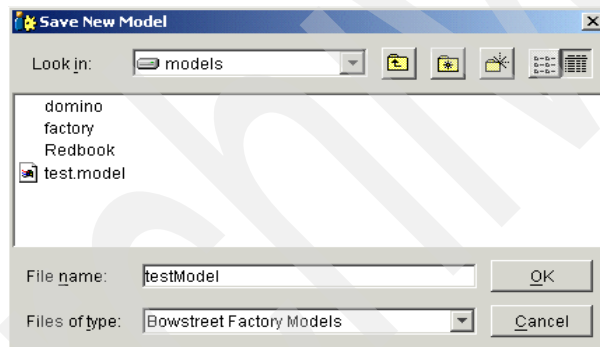


Figure 7-19 Creating a new Bowstreet model: Saving .model file

Once you have created your model file, you are ready to begin adding and configuring builders.

Adding the Domino View & Form builder

Bowstreet provides a great number of builders, each with a very wide range of available configuration options. When building a Domino portlet, you will generally start by adding the Domino View & Form builder and then refine this builder's functionality with other builders.

1. To add the Domino View & Form builder (or any other builder) select Model → Add Builder Call.

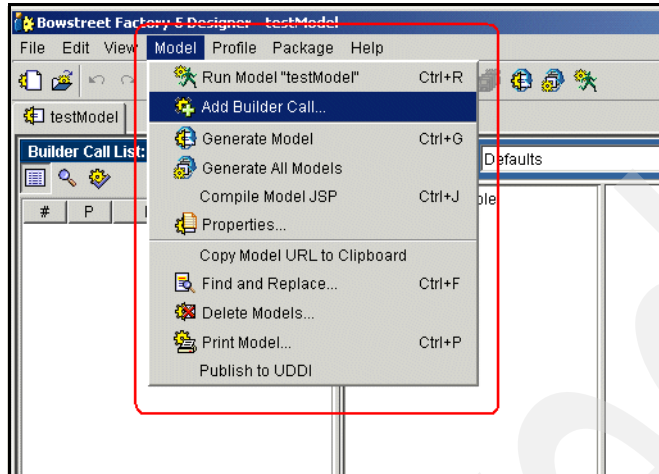


Figure 7-20 Adding a Bowstreet builder to a model file

- From the Bowstreet Builder Palette window, select All → Domino View & Form.

Note: The Domino View & Form builder type will only be available after you have successfully installed the Lotus Collaboration Extension pack.

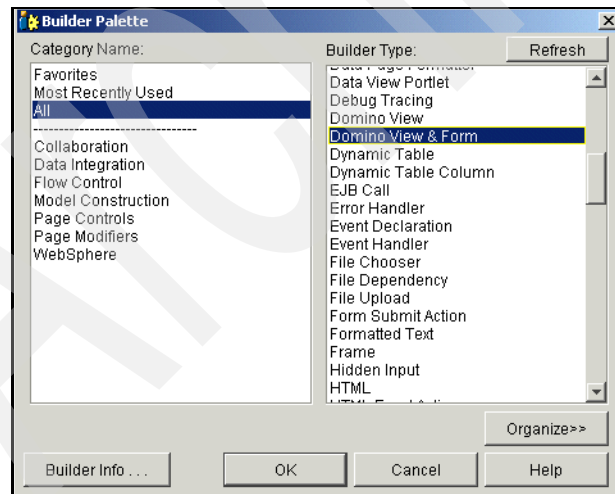


Figure 7-21 Bowstreet Builder Palette window

- After clicking OK, you will be given the Domino View & Form builder configuration interface (Figures 7-18 and 7-19).

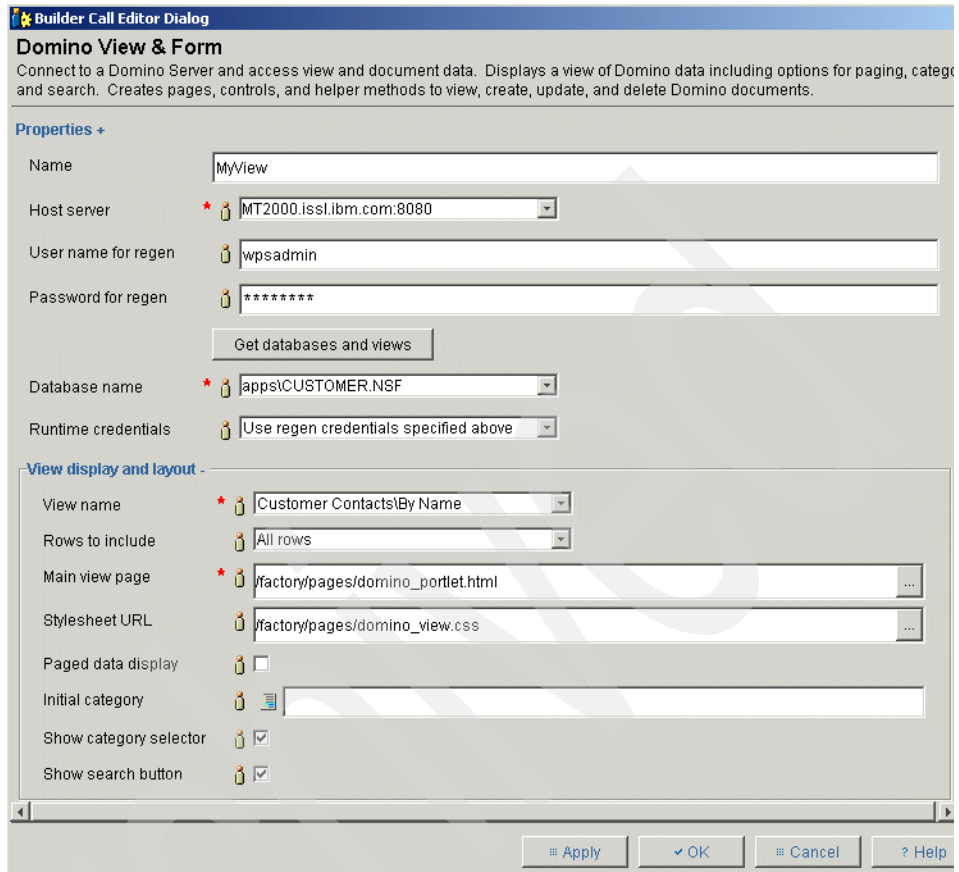


Figure 7-22 Bowstreet Domino View & Form Builder: Configuration window part 1

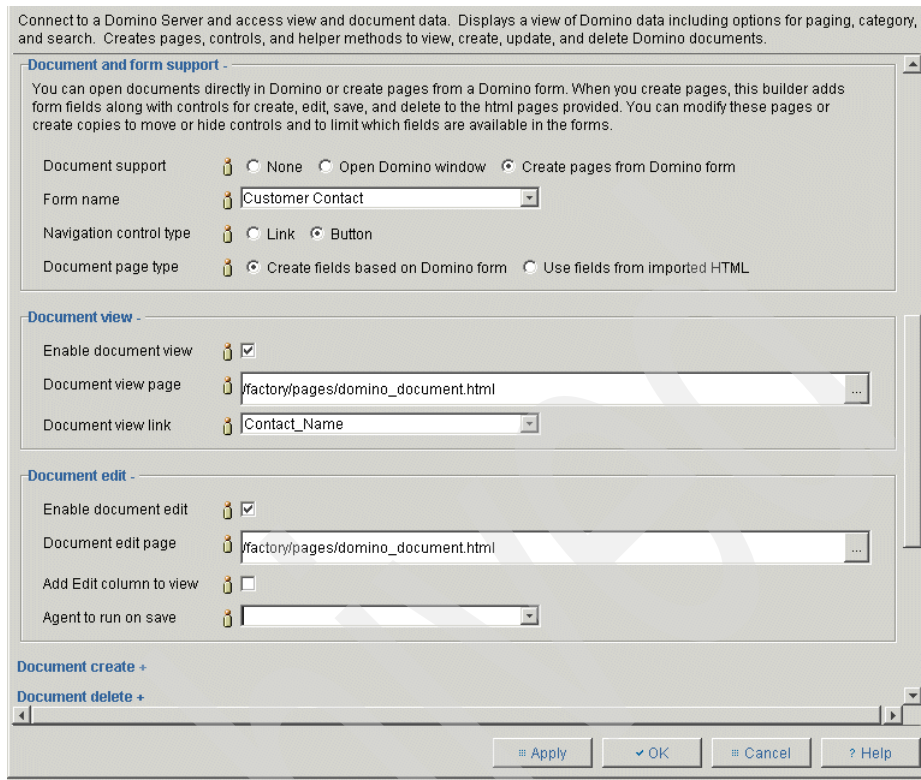


Figure 7-23 Bowstreet Domino View & Form Builder: Configuration window part 2

Note: It is important to verify that the HTTP and DIIOP tasks are running on the targeted Domino Server. You should also verify that the user id & password used has permissions to run Java agents on this server (see the server document in the server's Domino Directory (names.nsf)), and that the id has the appropriate access level to the database being accessed.

4. All settings for the Domino View & Form builder are controlled from this window. The configuration options available include:
 - Domino server, database, view, and form to use with this model
 - Authentication method (passed ID and password, LTPA, credentials vault)
 - Interface options:
 - View options (search, category selection, and so forth)
 - Document link method (no link, new window, create integrated form)
 - Document viewing, creation, edit, and deletion options

- Many other options
5. Once configuration is complete, click OK. You will be taken back to the model and should notice that Java code has been generated based on the builder you have configured.

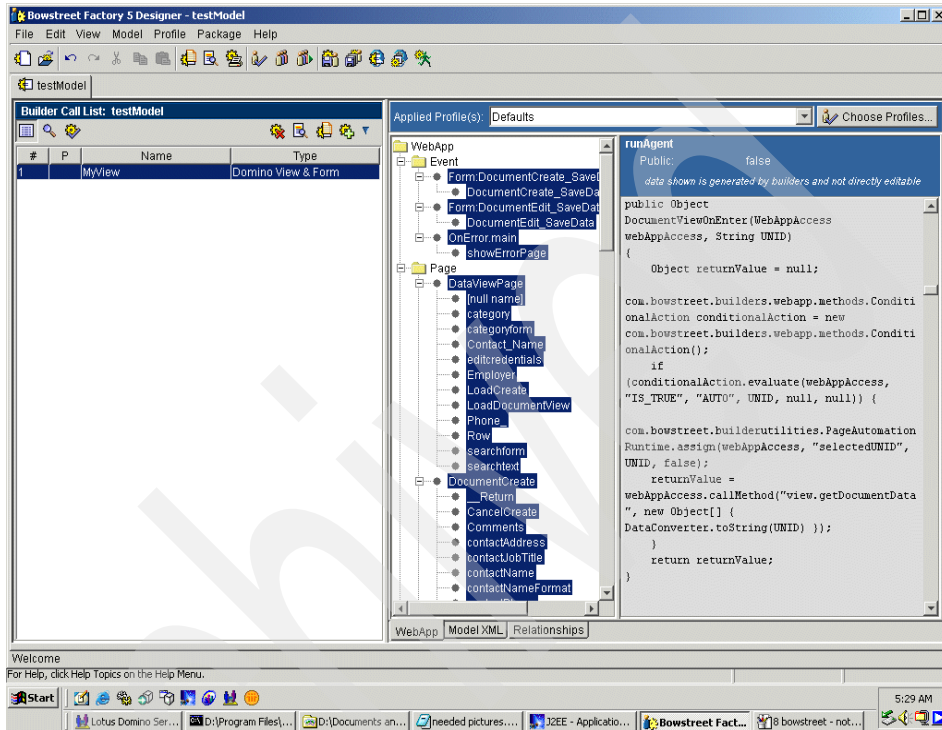


Figure 7-24 Java code generated from the Bowstreet Domino View & Form builder

It is possible to change the settings in the Domino View & Form builder configuration window (or any other builder's configuration window) at any time simply by double-clicking the builder name listed in the left-most window of the model interface.

Testing a Bowstreet Model

It is possible to easily test this model directly from the Bowstreet Designer client. To test a model simply click the Run Model icon in the top left corner of the Bowstreet model interface.

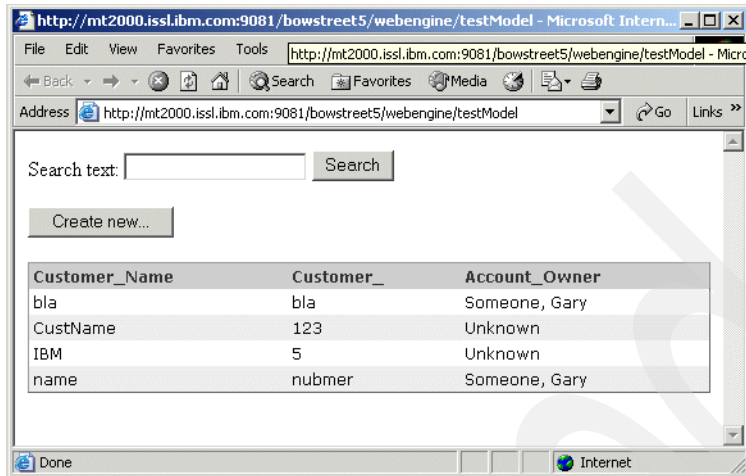


Figure 7-25 Test run of a model based on the Bowstreet Domino View & Form builder

Running a model from the Bowstreet Designer makes it very easy iteratively to add or configure a builder, and then quickly test the changes to confirm the change had the intended effect.

This action will test most aspects of the application with a few exceptions, such as Click to Action, which must be tested from within WebSphere Portal.

Deploying a Bowstreet Model as a Portlet

Turning a Bowstreet Model into a portlet is a fairly straightforward task. Simply add and configure the WPS Portlet Adapter builder to your model. For instructions on adding a builder to a model, see “Adding the Domino View & Form builder” on page 379.

Figure 7-26 lists the configuration options for the WPS portlet adapter builder.

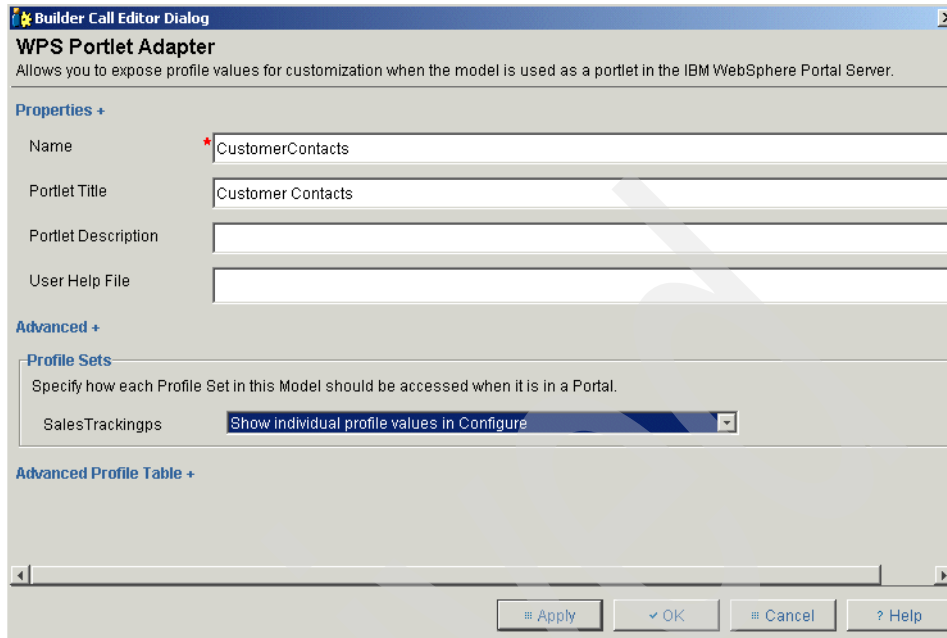


Figure 7-26 Bowstreet WPS Portlet Adapter builder configuration window

Once you have added this builder, you are ready to deploy your model as a WebSphere Portlet. You should find this to be quite simple. All Bowstreet portlets are actually concrete portlets referencing a single abstract portlet. The difference between portlets is simply the model the concrete portlet references.

The easiest way to create a new concrete instance of a Bowstreet portlet is to use the Bowstreet Portlet Creator portlet. This portlet works in much the same way as the Web Clipping portlet configuration tool. It provides a simple interface to create and configure each concrete Bowstreet portlet.

To use the Bowstreet Portlet Creator, log in to the WebSphere Portal where Bowstreet Portlet Factory is installed. Open a page containing the Bowstreet Portlet Creator portlet. If no page contains this portlet, simply create a new page and add this portlet.

From within the Bowstreet Portlet Creator, you can create a portlet based on your newly created model.

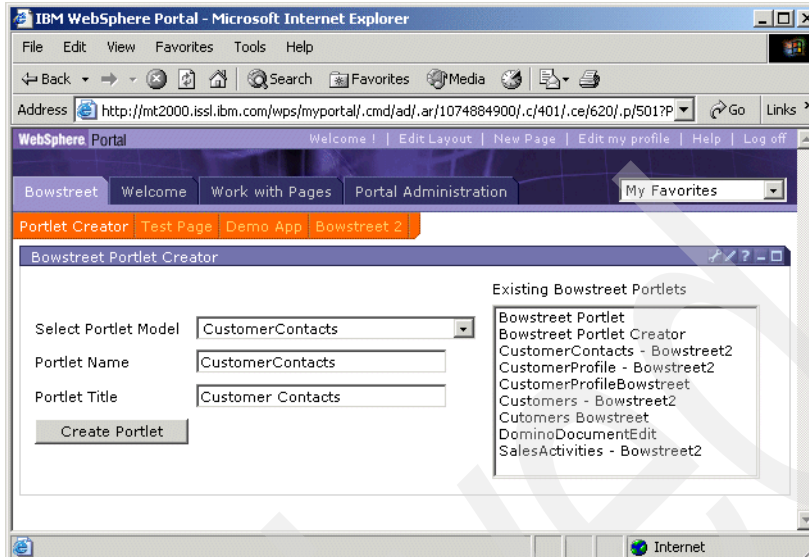


Figure 7-27 Bowstreet Portlet Creator portlet

After you have created your new portlet, simply add it to a page. It will look and function in much the same way as it did in your test environment.

It is also possible to package a Bowstreet portlet as a WAR which can run on any WebSphere Portal server, even if Bowstreet Portlet factory has not been installed. This process requires that you be familiar with the Jakarta ANT package, an open-source application used for building WAR files.

Implementing communication between Bowstreet models

When implementing Click to Action, one portlet (for example, model) is the event listener and another portlet triggers this event.

Note: Because Bowstreet uses the standard WebSphere Portal Messaging API (like Click to Action), it is possible to implement a non-Bowstreet portlet that triggers a Bowstreet event listener. It is also possible to trigger an event in a non-Bowstreet portlet.

Implementing a communication listener

In the Bowstreet model that will be an event listener, add and configure the WPS Event Declaration and Event Handler builders.

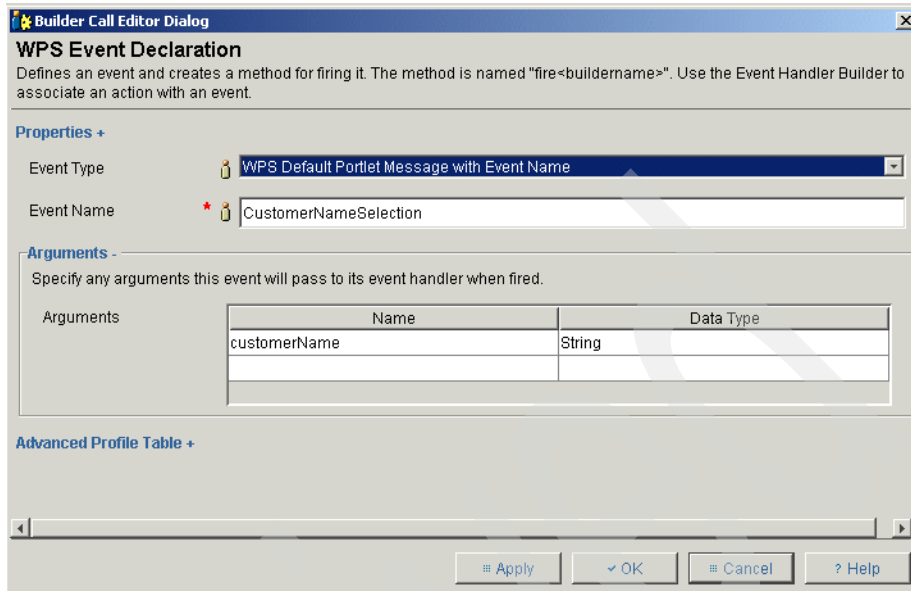


Figure 7-28 Bowstreet WPS Event Declaration builder configuration window

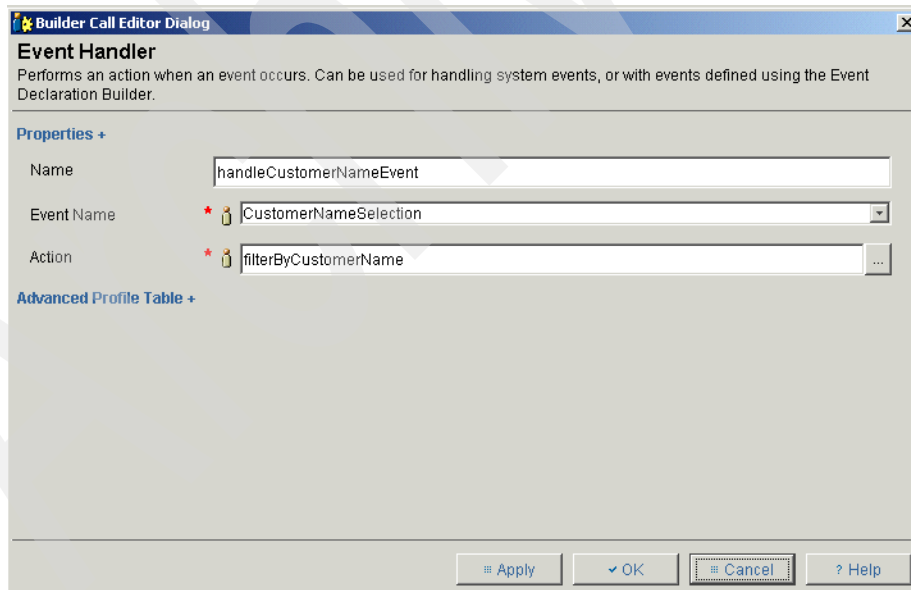


Figure 7-29 Bowstreet Event Handler builder configuration window

In the Event Handler builder, you must specify an Action to be called when this event is triggered. You can select an Action generated by the Domino View &

Form builder, or you can use parameters passed by the event trigger and chain a series of Actions together by adding and configuring an Action List builder.

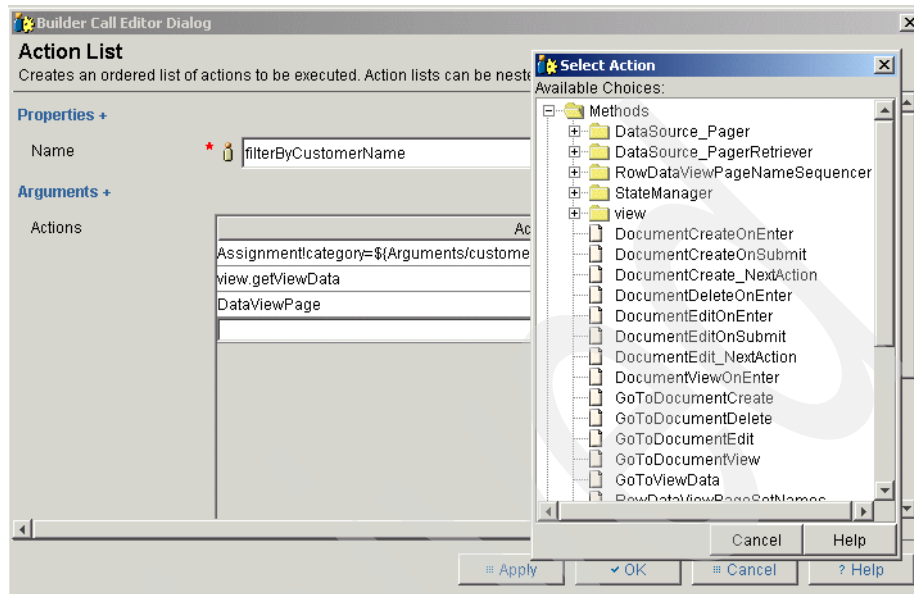


Figure 7-30 Bowstreet Action List builder configuration window

Implementing a Click to Action trigger

To trigger a Click to Action event from a Bowstreet portlet, add and configure the Link builder. Make sure the Action Type setting is set to “Link to an Action.”

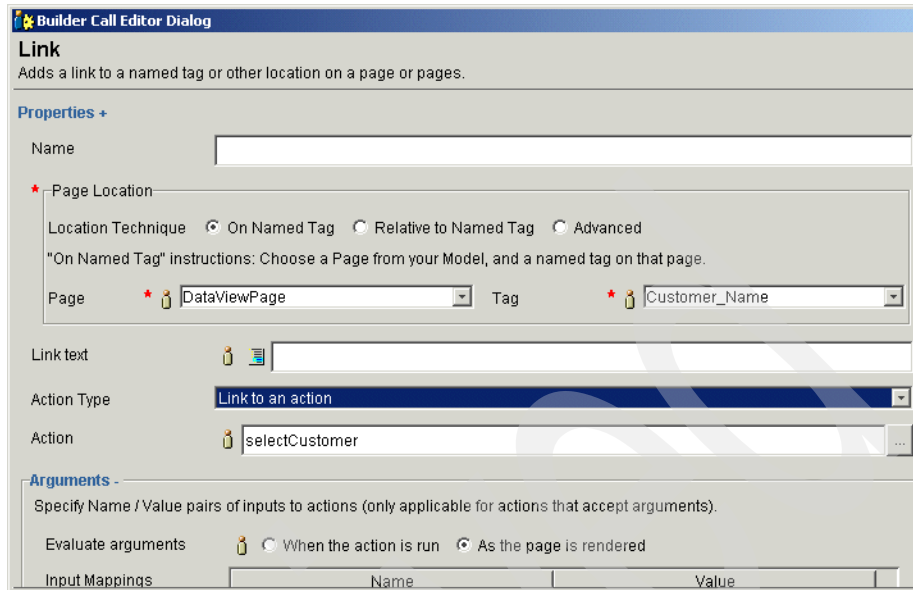


Figure 7-31 Bowstreet Link builder configuration window

You will also need to add and configure an Action List builder to define the action called by the Link builder. The Action List builder can trigger one or more events, and pass any number of required parameters, based on which link was selected.

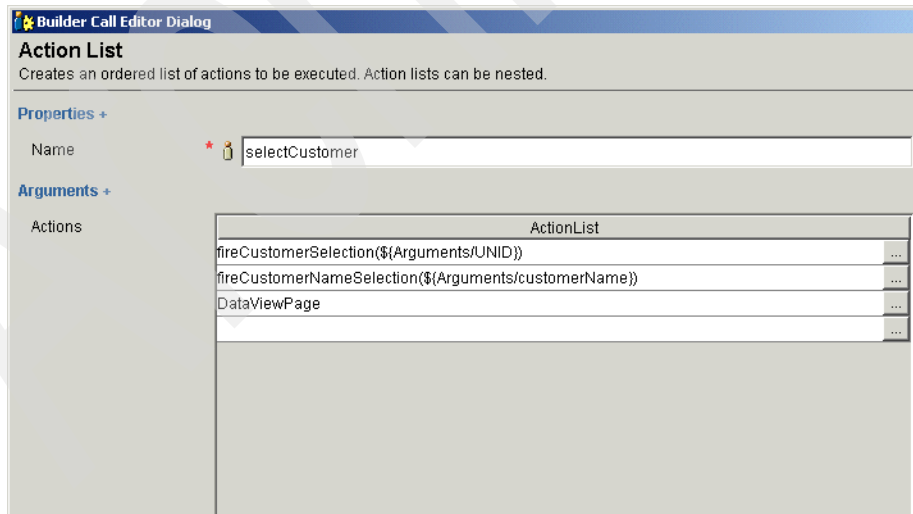


Figure 7-32 Bowstreet Action List builder configuration window

7.3.2 Implementation example

With the assistance of a Bowstreet expert, we implemented the Sales Tracking application case study introduced earlier. Within a day we had developed a fully functional portal workspace, complete with a Click to Action interface between portlets.

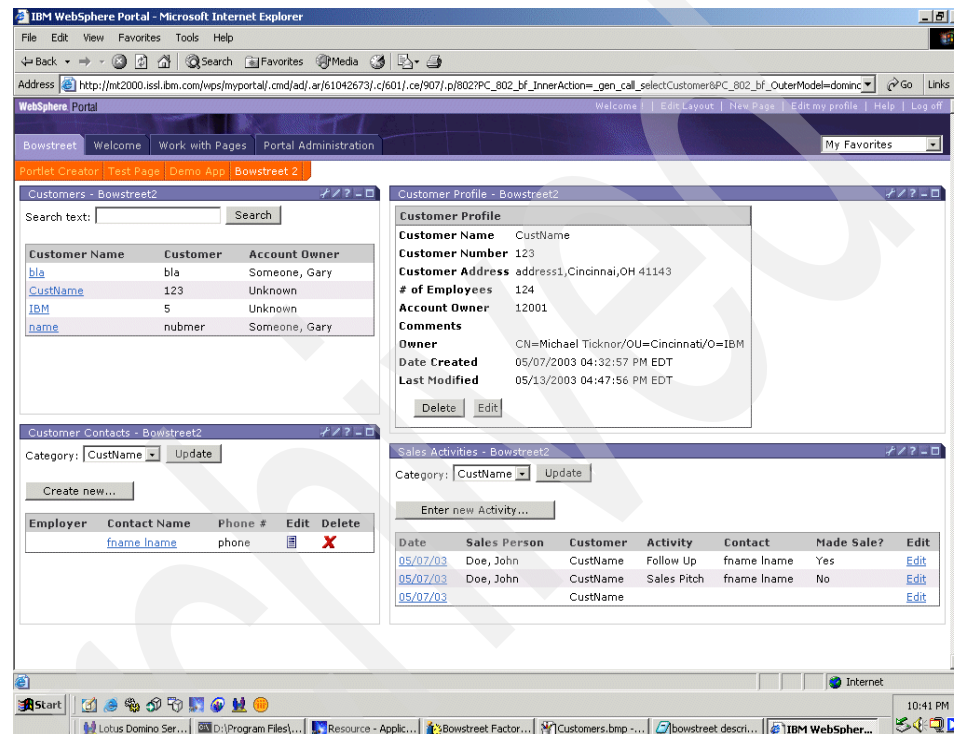


Figure 7-33 Bowstreet implementation of Sales Tracking case study

We created four models to implement this portal workspace, one for each portlet:

1. Customers view
2. Customer profile
3. Customer contacts
4. Sales activities

With each model, we started with the Domino View and Form builder, which generated a simple Notes view and form interface. We then configured the Domino View and Form builder and added additional builders to refine the interface and add functionality such as Click to Action.

Bowstreet's model-builder development technique requires some time to get used to. However, with a little bit of effort it ultimately proved very effective in developing our desired portal application. The ease with which we implemented the challenging functionality of Click to Action across multiple portlets was impressive.

There were a few spots where we added straight Java code. For example, we used a custom Java method to grab values from the product view for use in selection fields.

The following four tables summarize the builders used to generate our portal application. Some of these builders required detailed configuration that has not been captured in these tables.

Table 7-1 Customer profile

Builder Used	Description
Domino View & Form	Creates the base Domino form & view model: Customers\by Customer number, Customer form
WPS Event Declaration	Defines a Click to Action event within this portlet for triggering the action to open a customer profile document
Event Handler	Maps the event to the action, opens a customer profile document based on UNID
WPS Portlet Adapter	Turns this model into a portlet
Action List	Opens the "blank" startup page
Data Field Modifier	Hides the \$\$return field
Variable	Declares a "customer Selected" variable and sets to false
Visibility Setter	Hides the Domino view for this portlet
Action List	Creates the action of opening a page by UNID
Page	Declares the "blank" startup page
Visibility Setter	Hides the "back" button from customer documents, preventing the view from opening in this portlet
Domino View	Declares the sales people\by employee number view
Data Field Modifier	Modified employee_number field behavior in edit mode

Table 7-2 Customers

Builder used	Description
Domino View & Form	Creates the base Domino form & view model: Customers by Name view, Customer form
WPS Event Declaration	Defines a Click to Action event within this portlet for triggering the action to open a profile document by UNID
Link	Called when user selects a customer, triggers the action list, which triggers three Click to Action events
WPS Portlet Adapter	Turns this model into a portlet
Data Column Modifier	Sales Activity Name view, Adjust the tables, alignment, order, ordering and visibility of the view columns created by initial port of the Domino view
WPS Event Declaration	Declares a Click to Action event to handle when a customer name is selected
Action List	Triggers three Click to Action events, one in each of the other three portlets

Table 7-3 Customer contacts

Builder used	Description
Domino View & Form	Creates the base Domino form & view model: Customer Contacts\by Customer Name, Customer Contact form
WPS Portlet Adapter	Turns this model into a portlet
Data Column Modifier	Customer Contact form page: Adjust the labels, alignment, order, sorting, and visibility of fields created by the initial port of the form
Visibility Setter	Hide Employer column in view when employer category is selected
Visibility Setter	Hide system-generated edit button in the view interface
Visibility Setter	Hide system-generated delete Button in the view interface.
Data Column Modifier	Customer Contacts\by Customer Name view, Adjust the labels, alignment, order, ordering and visibility of the view columns crated by initial port of the Domino view
Action List	Set up an action instructing the portlet how to filter documents by a customer number

Builder used	Description
WPS Event Declaration	Defines a Click to Action event within this portlet for triggering the action to filter documents by customer name
Event Handler	Maps the event to the action, the event is triggered, the action is executed.

Table 7-4 Sales activities

Builder used	Description
Domino View & Form	Creates the base Domino form & view model: Sales Activity\by Customer view, Sales Activity form
WPS Portlet Adapter	Turns this model into a portlet
Data Column Modifier	Sales Activity Name view, Adjust the labels, alignment, order, ordering, and visibility of the view columns created by initial port of the Domino view
Data Column Modifier	Sales Activity form Adjust the labels, alignment, order, ordering and visibility of the view columns created by initial port of the form
Domino View	Products by Name View, used to look up product name values from the products database
Method	Write a custom Java method to access the product name values
Data Field Modifier	Creates drop-down list entries product selection fields
Variable	Creates a Java variable
Action List	Set up an action instructing the portlet how to filter documents by a customer number
WPS Event Declaration	Defines an Click to Action event within this portlet for triggering the action to filter documents by customer name
Event Handler	Maps the event to the action; when the event is triggered, the action is executed
Data Page Formatter	Formats the date created field

7.4 CONET Portlet Factory for Domino

Overview

CONET's Portlet Factory is a portlet generator for WebSphere Portal. It allows you to easily portalize existing Domino applications. The development model follows the Domino rapid application development approach, meaning that the portlets are created similar to the way that Domino applications are built, enabling Domino developers to build portlets without the need to know either Java or the portal server APIs.

The key differentiator between CONET Portlet factory and all other Domino integration techniques is Portlet Factory's performance and scalability. This is achieved through its multi-tier, distributed and server-centric architecture. It implements advanced data caching strategies using DB2 as its back end. This guarantees high performance and scalability that is independent from the Domino back-end servers.

The Portlet Factory tool is especially interesting if you have:

- ▶ Many custom Domino applications that you need to portalize
- ▶ Domino development expertise on staff
- ▶ The need for highly scalable Portlets
- ▶ The desire to use a rapid application development model that allows prototyping

CONET components

CONET's Portlet Factory is a true J2EE application implemented in a multi-tier architecture. The components that make up the system are:

- ▶ Portlet Factory Repository

The Portlet Factory Repository is the development environment and storage for the portlet definitions. The developers develop the portlets inside of the portlet definition repository. This is a Domino database with a Notes client front end.

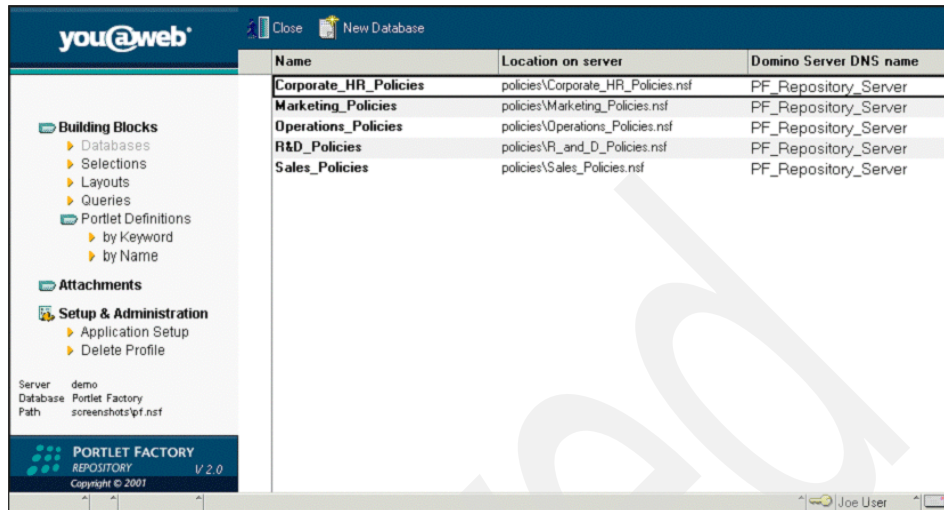


Figure 7-34 Portlet Factory definition repository

Portlet creation is done in a five step process that involves the definition of elements similar to the ones used to build Domino applications. Elements used are, for example, selection formulas, views, and form layouts. All these elements are defined by using the Notes @formula language in combination with HTML. This is explained in more detail later in this section.

- ▶ Portlet Factory portlets

The Portlet Factory portlet provides a framework for the portlets. The PF portlets are generic portlets which use the portlet definitions from the PF Definition Repository.

- ▶ Portlet Factory service

The PF Service is the core component of the Portlet Factory system. It serves incoming requests from Portlet Factory portlets.

- ▶ Portlet Factory connector

The PF Connector is a distributed component that accesses Domino databases on remote Domino servers. PF Connectors are deployed on every Domino node where databases reside that should be served through the portal.

Figure 7-35 gives an overview of the components.

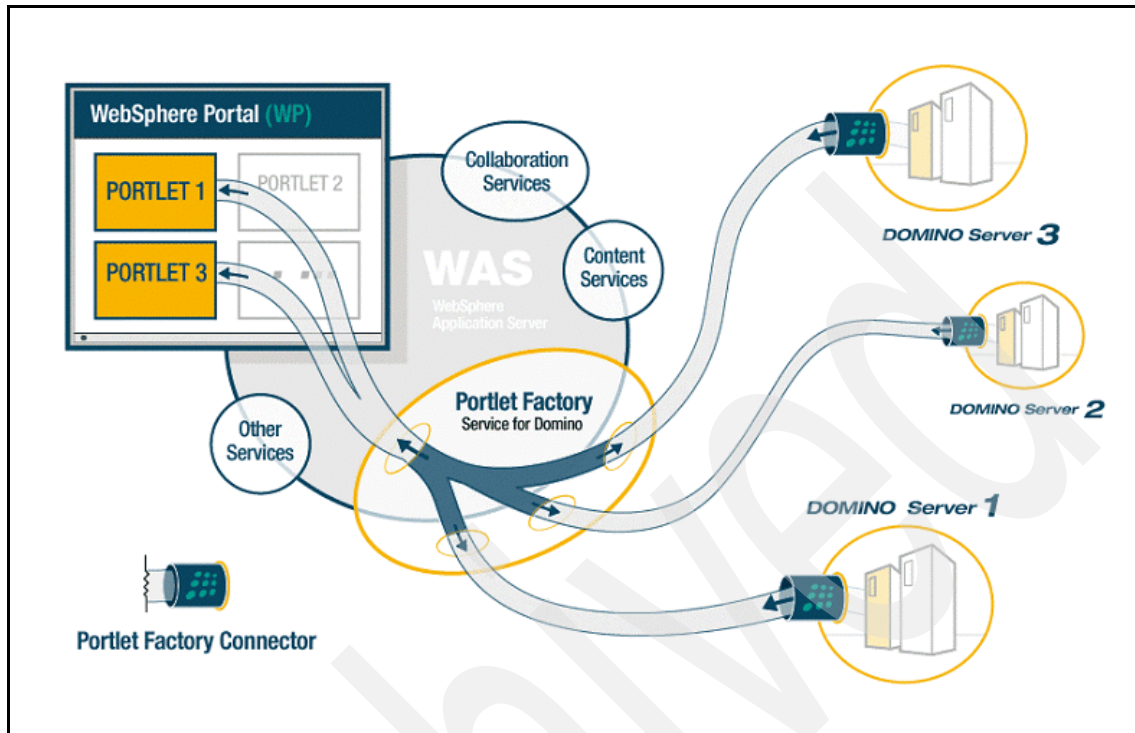


Figure 7-35 CONET Portlet Factory architecture

The Portlet Factory service is the main component of the system. It is implemented using Enterprise Java Beans. The Portlet Factory Service is responsible for handling all Portlet requests for data and application transactions. The Portlet Factory Service gathers information from the Portlet Definition repository and then utilizes one or more Domino Connectors to interact with a Domino application.

There are many tasks associated with fulfilling a request from a Portlet. The Portlet Factory Service facilitates the entire process of gathering, organizing, caching, and rendering the output for the Portlet, as well as the transactions with the underlying Domino application. Transactions can involve write access to Domino databases, Portlet cooperation, and agent execution on the back-end Domino server.

To improve performance, the Portlet Factory Service implements caching. Caching provides a significant performance increase in most situations. The Portlet Factory Service can cache results in memory or persistent to a DB/2 database.

As already mentioned, the Portlet Factory Service is a Java Web application that runs within WebSphere Application Server. The Portlet Factory Service receives requests directly from a Portlet running on a WebSphere Portal server.

The Portlet communicates with the Portlet Factory Service via RMI over IIOP.

Figure 7-36 gives an overview of the interaction of the different Portlet Factory components.

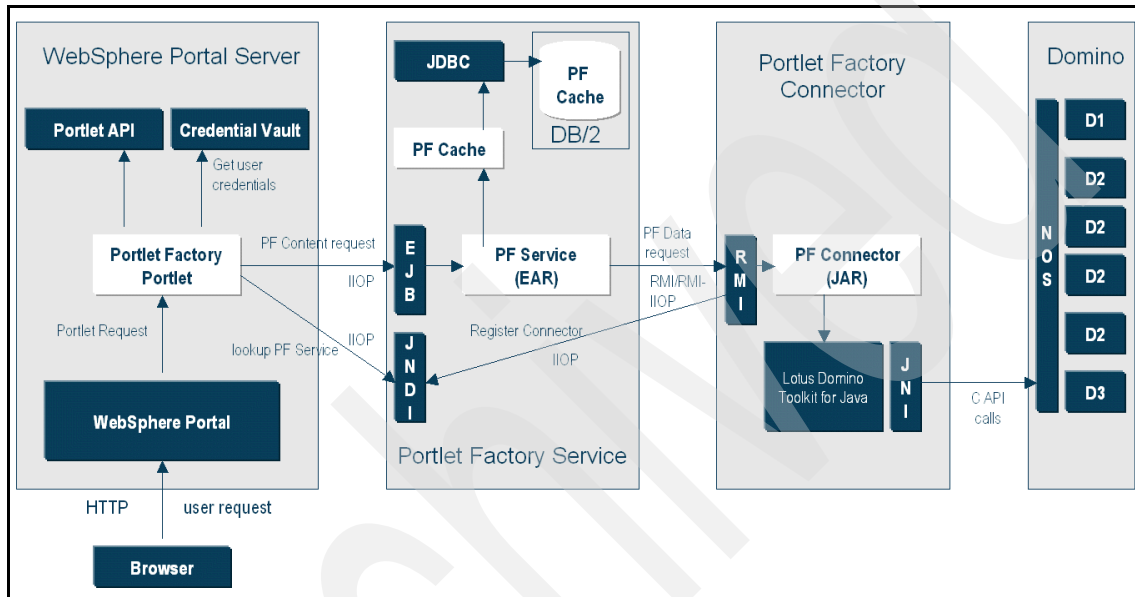


Figure 7-36 CONET Portlet Factory component architecture

In this figure, you see that Portlet Factory is *not* using HTTP or IIOP to remotely access Domino. CONET implemented a Domino Connector written in Java, which uses the Notes Object Services locally to perform the requests on behalf of the Factory Service.

The architecture was chosen to minimize the number of calls necessary to transact with a Domino server. Portlet Factory's remote connectors provide a coarse-grained interface. It receives a detailed task description, performs this task on the local server, and returns a result. Instead of dozens or hundreds of remote IIOP calls, there is just *one*. This improves performance and minimizes network traffic.

It is important to point out that the Domino connector is based on WebSphere and Domino standards. It builds on the Domino Java Toolkit and implements a high-level bridge for the specific transaction needs of the Portlet Factory service.

The modular architecture of Portlet Factory allows it to be deployed in different architectures, as shown in Figure 7-37.

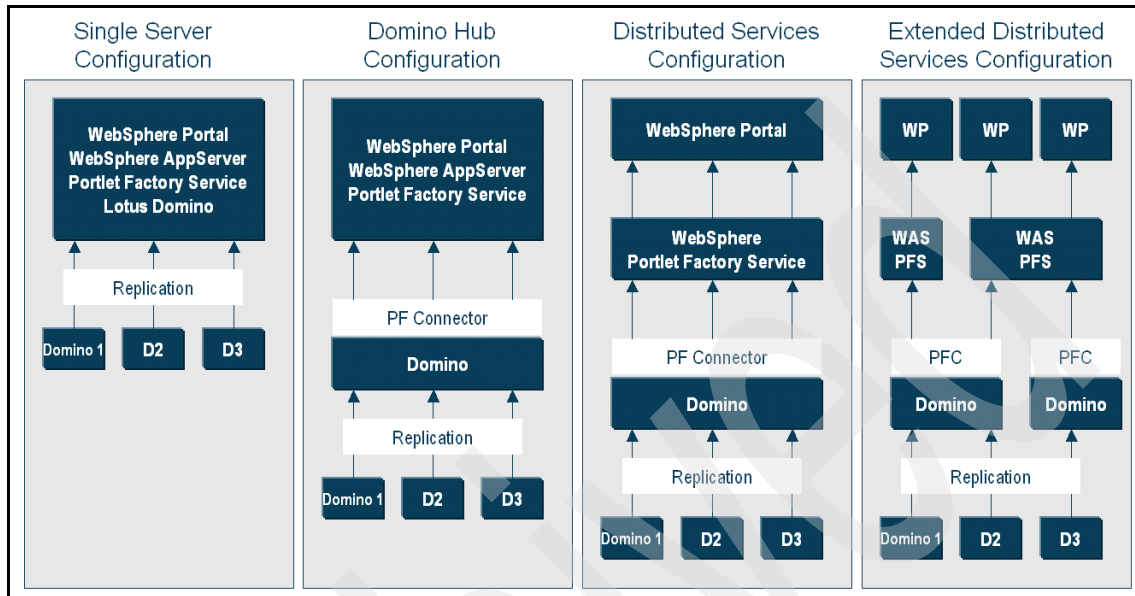


Figure 7-37 Portlet Factory deployment options

Implementation issues

This section summarizes the functionalities of the CONET Portlet Factory.

► Applicable portlet patterns

Portlet Factory Portlets can be used to implement the entire range of Portlet patterns that were outlined in Chapter 1.

► Development time

Portlet development with Portlet Factory is a very efficient process. The ease of use of the @formula language makes Portlet development as fast as building Domino applications. Portlet Factory implements a true rapid application development model.

► Developer skill set

Portlet development can be done with basic Domino development skills. Even junior Domino developers who know the @formula language and little HTML can build Portlets.

► Range of applications

Portlet Factory can be used to build any kind of Domino Portlets. It supports read and write access to Domino applications as well as the execution of agents. Portlet messaging and Click to Action are easy to implement.

The options for advanced database aggregation create a completely new way to work with Domino databases.

Portlet Factory allows you to:

- Combine data from multiple Domino databases
 - Append databases
 - Merge databases
 - Mix data sets
- Relate queries through primary/foreign key relationships:
 - One to One: One row in Query X matches one row in Query Y
 - One to Many: One row in Query X matches 0+ rows in Query Y
 - Many to Many: 1+ rows in Query X matches 1+ rows in Query Y
- Data views can be n-level categorized

One drawback is that in the development process you are limited to the functionalities provided by the @formula language. Currently the use of JSP tag libraries or customized Java code is not supported. CONET has announced that future versions of Portlet Factory will ship with a JSP Tag library that will open the Portlet Factory functionality to JSP development.

► Rich text handling

Portlet Factory Portlets can display and edit Domino rich text. Rich text can be embedded in Portlets with all embedded elements, for example images and links, still working correctly. Therefore the Portlet Factory service implements a reverse proxy and handles URL rewriting. We found that Portlet Factory is currently the only tool that handles Domino rich text properly.

Performance

Portlet Factory performance is very good. Through its multi-tier architecture and the use of memory and DB2 caching it can handle high volume Domino databases with high concurrent user access with good response times.

► Session management

Portlet Factory implements session management. Portlets can use either the active user or a broker that acts on behalf of a group of users to authenticate against the Domino backend.

► Clustering

The Portlet Factory connector can access clustered Domino servers.

In high availability and performance scenarios the Portlet Factory service can be clustered as well.

- ▶ Scalability

The Portlet Factory has been tested for databases with more than 100,000 documents and 1000 concurrent users.

- ▶ Requires single sign-on

Portlet Factory supports the following SSO configurations:

- Active and passive credential vault

Required software versions

Portlet Factory is currently available in Version 2.0 and requires the following software:

- ▶ WebSphere Portal 4.1/4.2
- ▶ WebSphere Application Server 4.0.4
- ▶ Domino 5.0.10 or later or Domino 6

Portlet creation methodology

CONET's Portlet creation methodology is completely Domino-centric. Portlets are created in a way similar to that used to build a Domino application.

The elements that define a Portlet are called "Building Blocks."

To better understand the building block concept, let's draw an analogy and imagine a Notes view, for example your "All Documents" view in your mailbox. You can ask yourself the question: What elements define this Notes view?

The answer is that the Notes view:

- ▶ Lives in a Domino database
- ▶ Has a selection formula
- ▶ Has layout information

Usually, a Notes view is created in the Domino Designer by defining these three elements inside of a view design element.

Let's compare this to a Portlet definition in Portlet Factory. Portlet Factory uses Portlet building blocks. Portlet building blocks deal with information, for example, about the database, selection formula, and the layout. So it uses the same three basic elements that we would use to build a Notes view.

Figure 7-38 on page 401 shows a comparison of a Portlet and Notes view.

Organizational Policies Edit | ? | □ | □

Topic	Department	Responsible	Date
Marketing New Personnel Policy <small>new</small> Complete information for screening new hires, training refreshment and background stats requirements, and CookieCutter Equal Opportunity policies.	Marketing	Fratelli, Juliet	05/23/02
HR Corporate Guidelines Corporate policies for interdepartmental communication, cross-department hiring and transfers, and blanket and procedures which affect all departments.	Human Resource	Miller, Erwin	05/07/02
Sales Partnering Procedures <small>updated</small> Defines corporate requirements for executing sales partnership agreements with third-party providers jointly with the Legal and HR departments.			
R&D Overtime Policies Interdepartmental brief outlining R&D overtime policies which deviate from corporate-defined standards, including holiday benefit transfers and special requests.			
Holiday Benefits – New Sales Hires Updated holiday listing includes MLK holiday, and now conforms to corporate standards. Additional sales-benefits and policy amendments for new hires.			

1 2 3 4 5 6 7 8 Next

Who	Date	Size	Subject
United Airlines	06/07/2002	17,304	Denver savings.
nilsonreyes	06/07/2002	931,512	Re: Folien
Julio-Fernando Soria	06/07/2002	14,133	Raiffeisen Bank
James Deters	06/07/2002	799	cell phone is not
Anders Larsson	06/07/2002	7,918	Re: Incident#14
James Deters	06/07/2002	2,456	Conference call

Figure 7-38 Portlet to Notes view comparison

To build a Portlet out of building blocks, Portlet factory allows you to combine base elements like databases and selections into higher level elements called “Queries” and “Portlet Definitions.”

Figure 7-39 on page 402 shows the containment hierarchy of the different building block elements.

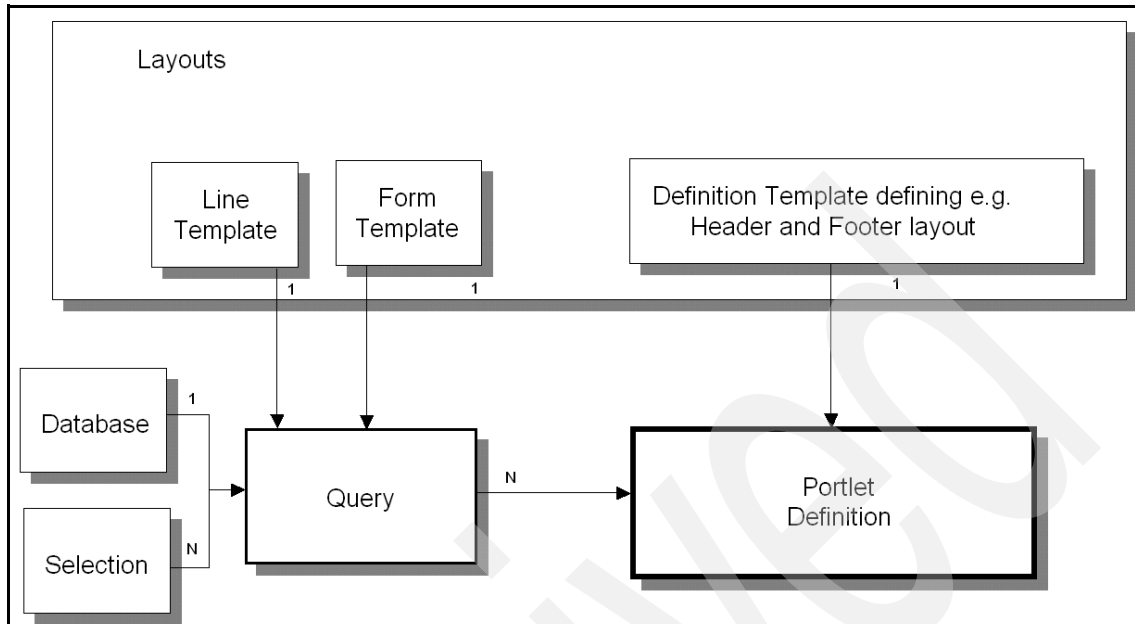


Figure 7-39 Element containment hierarchy

The Query element is essentially used to build views by summarizing one database, one selection, and layout information. The top-most element is the Portlet Definition. It can contain one or multiple Queries. This way, data from multiple Domino databases can be mixed, merged or categorized.

Once you have created a Portlet definition, Portlet Factory is able to automatically create a WebSphere Portal Portlet for you. With one click in the Portlet Factory UI, the tool creates a Portlet war file for you that can be deployed in WebSphere Portal.

In addition to being able to define views, Portlet Factory also allows you to create forms with read and write access, implement Portlet cooperation, and a lot more.

The key concept

The key concept to implement layouts and functionality with Portlet Factory is the use of the @formula language. The Portlet Factory @formula language is a superset of the Notes @formula language we all know from Domino development.

The Portlet layout is defined by using HTML with embedded @formula language, as illustrated in Example 7-1.

Example 7-1 HTML with embedded @formula language

```
<table border=0 cellspacing=0 cellpadding=0 width=100%>
<tr><td> <@ fldTitle @> <@ @if(@Date(@modified)=@today; '<img
src=/images/newicon.gif>&nbsp;'; '' ) @> <br>
<@ fldAbstract @> </td>
<td><@ @Date(@Modified) @><td>
</tr></table>
```

You see that Portlet Factory uses the @formula language in the same way you would use Java on a JSP page. As opening and closing tags for @formulas, you have to use <@ and @>. @Formulas can be as simple as single fields or can be more complex, for example @if conditional constructs.

Figure 7-40 is the Portlet output for the previous code example.



Figure 7-40 The output result for the previous code example

Portlet Factory allows you to use the entire Notes @formula language to build Portlets. It also gives you additional @formulas that make sense in the Portlet context.

7.5 Implementing the Sales Workplace example

In this section we walk you through the five step Portlet creation process to build a Portlet with Portlet Factory. We highlight a few functionalities that we used to implement searching and Portlet cooperation for our Sales Workplace Portlet.

1. Start the process by identifying your data source: the customer database.

Database History	
Basic Data	
Internal Name:	Customer
Comment:	
Database:	CUSTOMER.NSF
Title:	Customers
Internet Server Name:	
Port:	
Connector Server Name:	itsotest-dom
Access by:	<input checked="" type="radio"/> Replica ID <input type="radio"/> Path
Replica-ID:	85256D1F:00479A27

Figure 7-41 Database definition

This is done by creating a database document that contains a reference to the database that you want to portalize. Portlet Factory allows us to specify either the path or the replica ID to a Domino database. In most cases the replica ID will be used.

2. You need to select the information that is relevant from this database, so do a select statement. Portlet Factory allows you to build selections based on @formula statements or full-text selections. Remember: you are building a definition similar to a Notes view. If you have access to the design of the Domino database you can go ahead and copy and paste a view selection formula from a view.

Selection History	
Basic Data	
Internal Name:	Customer_Profiles_Only
Comment:	shows customer profiles only
Selection Type:	<input type="radio"/> FullText Selection <input checked="" type="radio"/> Formula Selection
Selection Data	
Formula:	Form="Customer"

Figure 7-42 Selection definition

3. This step is used to define a layout. As mentioned previously, the layout is a mixture of HTML and the @formula language. Portlet Factory layouts are made up of different layout components. You find layouts for the overall Portlet

layout—this is called Definition Template—as well as layout components for the individual result set row. In this example you are going to focus on the layout of the individual result set row. Portlet Factory names this a Layout Template.

Layout | History

Basic Data

Internal Name:

Comment:

Layout Type:

Sorting

new edit delete Default sorting

Sort By:

HTML

Template

Figure 7-43 Layout definition

The layout also includes an option for sorting. As in a Domino view, you can specify multiple sorted columns. Columns in the Portlet context refer to columns of the table layout.

4. In this step you summarize all the previous steps. You take your database, selection, and layout and put them into a query—the analogy of a Domino view.

Basics Layout Queries History	
Name	
Internal Name:	Customer_
Comment:	This is the customer view for the Portlet_
Name shown in the Web:	Customer_
Data source	
Database:	Customer_
Selection Type:	<input type="radio"/> FullText Selection <input checked="" type="radio"/> Formula Selection
Selection(s):	Customer_Profiles_Only_

Figure 7-44 Query definition Basics tab

On the Basic tab of the Query, select the database and selection definition for the Query. These are the elements defined in the previous steps.

Basics Layout Queries History	
Templates	
Template:	Customer_Profile_List_
Form Template:	Customer_Profile_
Failure Text:	There are no customer profiles available._
Comment:	_

Figure 7-45 Query definition Layout tab

On the layout tab, choose the template for the data display of the individual result set row as well as the Form template that is used to open the customer profile once the user clicks on a customer name inside of the Portlet.

5. Finally, the query gets put into a Portlet definition. Portlet definitions can contain more than one query. This is relevant if you want to merge data from multiple Domino databases. In this example we just have one query.

Basics | Layout | XML | History

Basic Data

Internal Name:

Admin Keyword:

Comment:

Title for Portlet:

Dependent Queries:

Figure 7-46 Portlet definition

- This was the final step in the definition process. The Portlet is now ready for creation. This is done by an automatic process that takes the Portlet definition and renders a portlet out of it.

Name	Queries	Template	Results	Last change
Customer	Customer	Customer	10	05/27/2003 10:36:55 AM

Figure 7-47 Automatic portlet creation

- Once this process is finished you can deploy the Portlet to your Portal and place it on the Sales Workplace. The result should look similar to Figure 7-48.

Customer Directory ?

Please select Customer

Name:

Location:

Customer #:

#	Company	Location	Company #
1	IBM-Lotus	Cambridge	4786
2	IBM-Tivoli	Rochester	9555
3	IBM-DB2	Bogota	9876
4	IBM-IGS	Cologne	2453
5	IBM-IS/SL	Dever	3214

1 2 next 12 Locations found

Figure 7-48 The resulting Portlet

Search

In this section we highlight how we implemented the search and Portlet cooperation.

To enable the search in the customer Portlet we added the code in Example 7-2 to the Portlet Definition Template.

Example 7-2 Customer search

```
<form "Search" action="<@@PortletViewAction@>" method="post">
  <input name="<@@ContentSearchName@>" size="40">
  <input type="submit" name="Search" value="Search">
</form>
```

In the layout definition, we created a basic HTML form with a search field. The trick is to give the search field the special name `<@@ContentSearchName@>` and use the `@` formula `<@@PortletViewAction@>` to create an Action URI to submit the form to the Portlets action listener. In the query definition we can define what fields Portlet Factory should include in the search.

Portlet cooperation

Enabling Portlet cooperation is as straightforward as implementing searches. In our example we want to implement the following Portlet cooperation:

- ▶ We search in the customer Portlet. Once we select a customer, the customer detail Portlet displays the customer details information.
- ▶ The customer contacts show only the contact personnel of the selected customer.
- ▶ The customer activity Portlet displays a view with the latest activities with that customer.

Example 7-3 Portlet cooperation

```
<a href="<@@sendmultiplemessage(PF_Customer_Detail;0;@DocumentUniqueID;
PF_Contact-Person;1;<@CustomerNumber@>;PF_Customer_Activity;1;<@CustomerNumber@
>)"><@CustomerName@></a>
```

These three actions are triggered with one line of `@` formula code. We build an HTML link that embeds the `@sendmultiplemessage` formula. This allows us to send multiple specific messages to other Portlets and have them respond to it.

After a user clicks on a customer name link, the other three Portlets will respond according to the defined behavior.

Summary

Building Portlets with CONET's Portlet Factory is easy for Domino developers who know the `@` formula language. The building process is straightforward and similar to building forms and views with the Domino designer. Results can be achieved fast and with minimum learning effort.

We were especially impressed by the good performance and scalable architecture of the tool.

However, we want to point out the following consideration with respect to the development environment: The use of the Notes client as the development environment makes coding difficult. We also found that a debugger would be a useful enhancement.

7.6 Other portlet builders

7.6.1 Sofor Interactive Portlet Builder for Domino

Sofor Interactive Portlet Builder for Domino is a tool for bringing Lotus Domino database views and documents into the portal. The portlet builder can connect to any Domino database. No changes to existing databases are needed. All customization is done in the edit mode of the portlet.

Portlet Builder portlets work in all browsers that are supported by WebSphere Portal or WebSphere Everyplace Access. With WebSphere Everyplace Access, off-line functionality is available as well: Domino database access through Portlet Builder can be defined as off-line portlets, and can be used on a PDA even when no connection to the portal is available.

Views are displayed in portlets, and the user is able to navigate through the view. Clicking a line displays those fields of the document that have been configured to be published in the portal. The portal also is used to authenticate users.

Views and document fields from any Lotus Domino database can be published in a WebSphere portal, without any changes to the existing databases, through configuration wizards with no programming.

All traffic from the portal to the Domino Server goes through the internal firewall. Terminals never access Domino servers directly.

CustomerName	Number	ownerName
My Bank.com	19	Marko Viksten
MyCompanyPortal	4	Chris Heltzel
Neo Portals	18	Michael Ticknor
Portal-makers	3	Camilo Rojas
Portal People eaters	15	Marko Viksten
Portals-R-Us	11	Chris Heltzel
The company	6	Marko Viksten
Wanda's World of Wigs	14	Chris Heltzel
Winging Portals	12	Chris Heltzel
WP Experts	16	Gary Someone

Figure 7-49 Sofor Interactive Portlet for Lotus Domino in our Scenario, retrieving data from the customer database

Considerations

Sofor Interactive Portlet Builder for Domino could be used when you want an easy, basic tool for exposing any Domino database views in a portal without programming skills, or if your company has WebSphere Everyplace Access and your mobile users need to access Domino views or documents.

The limitations of this solution are similar to those of using existing portlets, described in Chapter 3. Specifically, you are limited to the functionality the Portlet Builder offers. Also, the lack of session pooling might be an issue if your portal has large numbers of portlets and users.

Implementation details

Sofor Portlet Builder can be downloaded from the IBM Portlet catalog at:

http://www-3.ibm.com/services/cwi/portals/_page/105/

Once you have installed and placed the portlet in a page, you need to configure it in edit mode.

Table 7-5 Parameters for Sofor Interactive Portlet Builder for Domino

Parameter	Description	Example
Host	Host name/IP address of the Domino server	itsotest-dom
UserID	Web user ID. User must have restricted Java/COM agent execution rights.	wpsadmin
Password	Password of user above.	password

Parameter	Description	Example
Database	Database name (and path if needed).	apps/Custom.nsf
Title	Title of portlet loaded in title bar.	MyTitle
View	Name or alias of used view/COM agent execution rights.	MyView
Columns	Zero-based column number and title of selected columns. Multivalue separator is comma ",".a/COM agent execution rights.	0=My first column,2=My second column
Link column	Zero-based number of column to show as link.	1
Fields	Fieldname and title to show when document selected. field=title. Multivalue separator is comma ",".	field1=My first field,field2=My second field

Click OK to finish your configuration changes and log off. During a test of the portlet we faced an issue where all the configuration changes made in edit mode did not become active until we logged off and the logged back in to the portal.

7.6.2 Aptrix Portlet Connector

Aptrix Portlet Connector is a WebSphere Portal application that can query and aggregate data from multiple data sources, and display that content in a single portlet. Portlet Connector uses the Aptrix Connect product (installed as middleware on WebSphere Portal) to natively query data hosted on Domino servers, relational databases, and “flat” data sources (HTML, ASCII text, and so forth). Individual Aptrix Portlet Connector portlets can be installed that access one or more data sources, and aggregate the content into a single portlet view.

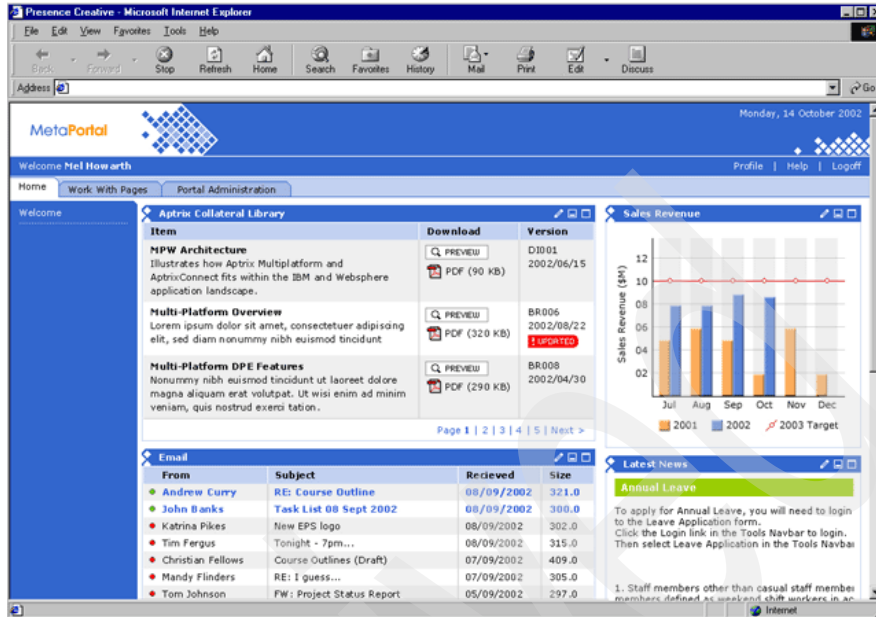


Figure 7-50 Example portal page created with Aprix Portlet Connector

Considerations

Aprix Portlet Connector is a good choice if you are already using Aprix Connect, or if you are looking for a way to aggregate multiple data sources into one or more portlets.

Implementation details

Aprix Portlet Connector is very easy to configure. All configuration is performed through the edit interface of each portlet. No Java skills are required, but if the functionality does not meet your specific needs, it is not possible to customize the portlets created with the Aprix Portlet Connector.

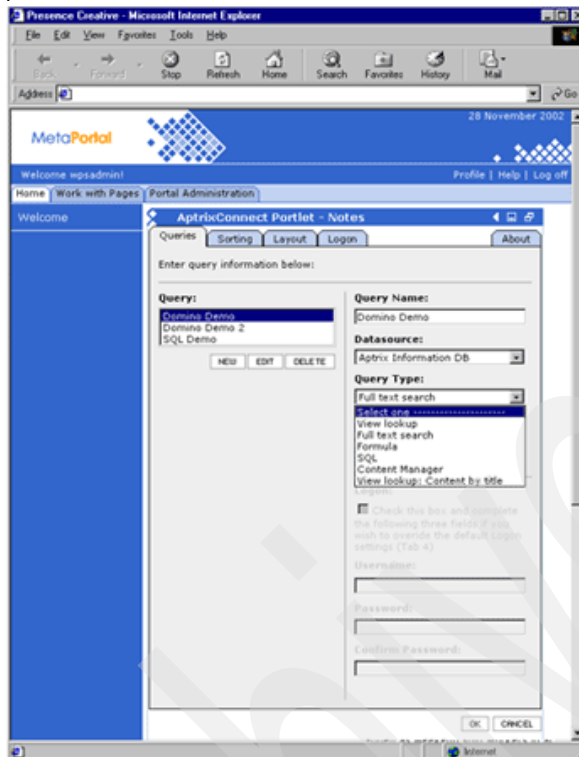


Figure 7-51 Aptrix Portlet Connector configuration interface

Aptrix Portlet Connector uses a proprietary Java socket server to communicate directly with the Domino server, and thus does not require HTTP or IIOp to be running on all back-end Domino servers.

It supports Domino 5 and 6 and WebSphere Portal v4.1 and v4.2.

Archived



Data dictionary for case study

This appendix contains the specific field names, field labels, and data types for each document type in our fictional Sales Tracking application.

A.1 Product Form

Field name	Label	Data type	Description
Form		Text	"Product"
ProductName	Name	Text	Required
ProductNumber	Number	Number	Identifier unique to this product No white space in value Required
ProductPrice	Price	Number	Required Not negative Currency format
ProductDescription	Description	Text	
Comments	Additional Information	Rich text	

A.2 Sales Person Form

Field name	Label	Data type	Description
Form		Text	"Sales Person"
SName	Name	Authors	Required
SNameFormat	Hidden	Text	<Last Name>, <First Name>
SNumber	Employee #	Text	Identifier unique to this employee Required No white space in value
SPhone	Phone #	Text	
SRegion	Region	Text	Drop-down list Keywords hard coded
STitle	Job Position	Text	Drop-down list Keywords hard coded
Comments	Additional Information	Rich text	
Owner	Owner	Authors	Authorized to edit

DateCreated	Date Created	Date	computed
DateModified	Last Modified	Date	computed
\$\$Return	Hidden		Save success message on Web

A.3 Customer Form

Field Name	Label	Data type	Description
Form		Text	"Customer"
CustomerName	Customer Name	Text	Required
CustomerNumber	Customer Number	Text	Identifier unique to this customer No white space in value Required
OwnerNumber	Account Owner	Text	Drop-down list Uses Sales People to populate keyword values. Stores EmployeeNumber, displays EmployeeName (EmployeeNumber) Required
OwnerNameFormat	hidden	Text	Computed <Last Name>, <First Name> Formatted name value for OwnerNumber
CustomerAddress	Address	Text	Multi-line, multi-value Use TEXTAREA on Web
EmployeeTotal	# of Employees	Number	Not negative Integer only
Comments	Additional Information	Rich text	
Owner	Owner	Authors	Authorized to edit
DateCreated	Date Created	Date	Computed
DateModified	Last Modified	Date	Computed
\$\$Return	Hidden		Save success message on Web

A.4 Customer Contact Form

Field name	Label	Data type	Description
Form		Text	"Customer Contact"
ContactName	Contact Name	Text	Required
ContactNameFormat	Hidden	Text	Computed <Last Name>, <First Name>
CustomerNumber	Employer	Text – drop down list	Uses Customers to populate keyword values. Stores CustomerNumber, displays CustomerName (CustomerNumber) Required
CustomerName	Hidden	Text - computed	Value from CustomerName field in Customer Record
ContactPhone	Contact Phone	Text	
ContactAddress	Contact Address	Text – multiline	Use TEXTAREA on Web
ContactJobTitle	Contact JobTitle	Text	
Comments	Additional Information	Rich Text	
Owner	Owner	Authors	Authorized to edit
DateCreated	Date Created	Date	Computed
DateModified	Last Modified	Date	Computed
\$\$Return	Hidden		Save success message on Web

A.5 Sales Activity Form

Field name	Label	Data type	Description
Form		Text	"Sales Activity"
AType	Activity Type	Text	Drop-down list Hard coded keyword values Required
ADate	Activity Date	Date/Time	Defaults to today
CustomerNumber	Customer	Text	Drop-down list Uses Customers to populate keyword values. Stores CustomerNumber, displays Customer Name (CustomerNumber) Required
CustomerName	hidden	Text	Computed
ContactName	Customer Contact	Text	Drop-down list Uses Customer Contacts to populate keyword values
ContactNameFormat	Hidden	Text	Computed Uses ContactName <Last Name>, <First Name>
SNumber	Sales Person	Text	Drop-down list Uses Sales People to populate keyword values. Stores EmployeeNumber, displays Employee Name (EmployeeNumber) Required
SNameFormat	Hidden	Text	Computed Uses SNumber to lookup Sales Person name <Last Name>, <First Name>
MadeSale	Made a Sale?	Text	Radio button Hard coded keyword values: Yes / No
Product1	Product	Text – Drop down list	Uses Products to populate keyword values. Stores ProductName (ProductNumber)
Product2			

Product3			
Product1Q	Quantity	Number	
Product2Q			
Product3Q			
Comments	Additional Information	Rich text	
Owner	Owner	Authors	Authorized to edit
DateCreated	Date Created	Date	Computed
DateModified	Last Modified	Date	Computed
\$\$Return	Hidden		Save success message on Web

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

<ftp://www.redbooks.ibm.com/redbooks/SG247004>

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG24-7004-00.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

<i>File name</i>	<i>Description</i>
DominoDbs.zip	Zipped Domino databases

Portlets.zip

Zipped WAR files for Portlets created in this book.

How to use the Web material

Create a subdirectory (folder) on your workstation, and unzip the contents of the Portlets.zip file into this folder.

Deploy the portlets into your WebSphere Portal 4.2 server.

Unzip the contents of the DominoDbs.zip file into **apps** subdirectory of your Domino Server's data directory.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 426. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *IBM Websphere Portal V4 Developer's Handbook*, SG24-6897
- ▶ *WebSphere Portal 4.12 Collaboration Services*, REDP-0319
- ▶ *Domino and WebSphere Together, Second Edition*, SG24-5955
- ▶ *WebSphere Studio Application Developer Programming Guide*, SG24-6585
- ▶ *Access Integration Patterns using IBM WebSphere Portal*, SG24-6267

Other publications

These publications are also relevant as further information sources:

- ▶ *Building a Portlet within the Model-View-Controller Paradigm using WebSphere Portal*
http://www7b.software.ibm.com/wsd/library/techarticles/0210_kwong/kwong.html
- ▶ *WebSphere Portal Programming: Portal Aggregation for Pervasive Devices*
http://www7b.software.ibm.com/wsd/techjournal/0210_godwin/godwin.html
- ▶ *Hello World Portlet Revisited: Adding Globalization Support for Multi-languages Using WebSphere Portal 4.1.2*
http://www7b.software.ibm.com/wsd/library/techarticles/0210_xu/xu.html
- ▶ *WebSphere Portal V4 programming, Part 1: Portlet application programming*
<http://www-106.ibm.com/developerworks/library/i-portalv4/?n-dd-8222>
- ▶ *WebSphere Portal V4 programming, Part 2: Portlet application programming*
<http://www-106.ibm.com/developerworks/library/i-portal2v4/?n-dd-8222>

- ▶ *WebSphere Portal Programming: Pervasive Portlet Development*
http://www7b.software.ibm.com/wsdd/techjournal/0207_wanderski/wanderski.html
- ▶ *Lotus Collaborative Services JavaDoc*
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/4.2api/collaborative/>
- ▶ *Introduction to JavaServer Pages*
<http://www-105.ibm.com/developerworks/education.nsf/java-onlinecourse-bytitle/882707E838C672A185256770004BDE72?OpenDocument>
- ▶ *Using JSPs and custom tags within VisualAge for Java and WebSphere Studio*
<http://www7b.software.ibm.com/wsdd/library/tutorials/vajwebsph353/Part-I/JSP11Part-I.html&origin=cmp>
- ▶ *Guide to WebSphere Portal*
<ftp://ftp.software.ibm.com/software/websphere/portal/pdf/Guide-to-WebSphere-Portal.pdf>
- ▶ *WebSphere Portal Product Documentation*
<http://www7b.software.ibm.com/wsdd/zones/portal/proddoc.html>
- ▶ *WebSphere Portal for Multiplatforms - Library*
<http://www-3.ibm.com/software/genservers/portal/library/>
- ▶ *Portlet Development Guide*
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/portletdevelopmentguide.html>
- ▶ *Portlet Best Practices Guide*
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/portletcodingguidelines.html>
- ▶ *Struts in WebSphere Portal 4.1*
<http://www.ibm.com/support/docview.wss?rs=0&org=SW&doc=7002247>
- ▶ *Using Click to Action to Provide User-Controlled Integration of Portlets*
http://www7b.software.ibm.com/wsdd/library/techarticles/0212_roy/roy.html

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ IBM Lotus Domino Developer Domain
<http://www.lotus.com/idd>

- ▶ IBM WebSphere Developer Domain - Portal Zone
<http://wilson.boulder.ibm.com/wsdd/zones/portal/>
- ▶ IBM WebSphere Portlet Catalog
<http://www.ibm.com/software/Webservers/portal/portlet/catalog/>
- ▶ The Lotus Domino Toolkit for Java/CORBA / Domino Java API
<http://www-10.lotus.com/1dd/toolkits>
- ▶ CORBA
<http://www.omg.orgb>
- ▶ WebSphere Portal InfoCenter
<http://publib.boulder.ibm.com/pvc/wp/current/index.html>
- ▶ WebSphere EveryPlace Access InfoCenter
http://www-3.ibm.com/software/pervasive/products/library/ws_everyplace_access.shtml
- ▶ Portlet API
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/4.1api/>
- ▶ Sun Microsystems J2EE Web site
<http://java.sun.com/j2ee/>
- ▶ Sun Microsystems JSP Web site
<http://java.sun.com/jsp>
- ▶ WebSphere Portal 4.2.1 Infocenter
<http://publib.boulder.ibm.com/pvc/wp/42/index.html>
- ▶ Portlet JSP Tag Library Syntax
<http://www7b.software.ibm.com/wsdd/zones/portal/portlet/V41jsptaglib.html>
- ▶ Portal Zone
<http://www7b.software.ibm.com/wsdd/zones/portal/>
- ▶ Portal Struts support
<http://jakarta.apache.org/struts/index.html>
- ▶ Struts enablement package
<http://www-3.ibm.com/software/webervers/portal/portlet/catalog/action/ChangePage/pg/74/.reqid/3?viewPage=detail&NAVCODE=1WP10003N>
- ▶ Log4j
<http://jakarta.apache.org/log4j/docs/>

- ▶ Object pooling
<http://jakarta.apache.org/commons/pools>
- ▶ Click to Action Enablement package
<http://www-3.ibm.com/software/webservers/portal/portlet/catalog/action/ChangePage/.pg/74/.reqid/1?viewPage=detail&NAVCODE=1WP1003L>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

\$\$Return field 53
*webuser 178
.WAR file 323
@DbLookup 53
@Functions 53

A

Access Control portlet 54
Access integration 2
Accessing a Domino document 260
Accessing a Domino view 259
ACL 329
Action URIs 269
Aggregating informaion from multiple databases 23
Aprtix Portlet Connector 47, 411
Authentication 31

B

Basic portlet 169
Blended applications 154
Bowstreet Designer 373
Bowstreet Domino View & Form Builder 381
Bowstreet Portlet Factory
 benefits 372
 builder 373
 Click to Action 386
 code execution architecture 374
 considerations 375
 deploying 384
 development environment 373
 Domino form builder 379
 Domino view builder 379
 implementation example 390
 Lotus Collaboration extension 372
 model 373
 opening the developer 378
 overview 372
 performance 376
 profile 373
 rich text 376
 sample portlets 391

 server components 374
 skillset 372, 376
 testing a model 383
Bowstreet Portlet Factory for WebSphere 46, 372
Brokered cooperation 12
Building the portlet project with WebSphere Studio 168

C

c2a.tld 204
Case Study
 application details 51
 application functionality 53
 databases 47
 document types 49
 overview 47
 portlets 51
 relationships 51
 sales workplace 50
 use cases 52–53
 user roles 52
Case study 47
Catching exceptions 264
Choosing an integration technique 38
Click to Action 12, 199
 199
 architecture 202
 broadcasting 199
 broker 200
 chained propagation 199
 concepts 200
 enabling source portlets 205
 enabling target portlets 207
 encodeProperty 206
 event broker 202
 features 199
 functionality 199
 implementation 203
 model 199
 namespace 215
 operating diagram 201
 request-response flow 203
 source 200

- source portlets 200
 - target 200
 - target portlets 200
 - web.xml 210
 - Collaboration services 270
 - Collaborative Components 217
 - CSCredentials 270
 - CSEnvironment 270
 - CSFactory 270
 - DominoService 270
 - people tags 270
 - PeopleService 271
 - QPService 271
 - setting up the environment 271
 - Collaborative Components API 270
 - Collaborative workplace 2
 - Column container 34
 - Combining data from multiple databases 25
 - Concrete portlet instance 231
 - Concrete portlets 231
 - Conet Portlet Factory
 - @formulas 402
 - architecture 396
 - benefits 394
 - caching 396
 - component interaction 397
 - components 394
 - considerations 398
 - creating portlets 400
 - data source 404
 - defining the layout 405
 - deployment architecture 398
 - Domino connector 397
 - implementation example 403
 - overview 394
 - performance 399
 - portlet cooperation 408
 - portlet elements 402
 - Portlet Factory connector 395
 - Portlet Factory service 395
 - rich text 399
 - scalability 394, 400
 - search 407
 - selection 404
 - skillset 398
 - Conet Portlet Factory for Domino 46, 394
 - Connecting to a Domino database 174, 357
 - Copying the RSS portlet 113
 - CORBA 139, 247, 249
 - accessing Domino 251
 - architecture 251
 - functionality 251
 - stub objects 251
 - Creating a new document 260
 - Creating a WSDL file 210
 - Credential vault 31
 - CSS 32
 - custom JSP tags 160
- ## D
- Data aggregation 28
 - Deployment descriptors 188
 - Design considerations 22
 - DIIOP 247, 249
 - configuration data 253
 - console commands 253
 - debugging 254
 - diiop_ior.txt 274
 - enabling on Domino server 251
 - IOR 252
 - logging 253
 - new features in Domino 6 273
 - session time-out 275
 - task, loading 253
 - task, refreshing 253
 - transactions 254
 - DIIOP task 168, 252
 - Displaying rich text data 265
 - Displaying rich text using HTTP 266
 - DNS 253
 - DOM 247
 - Domino 1, 64
 - accessing from portlets 247
 - applications 7
 - creating a subform 122
 - desing modifications 109
 - distributed environment 29
 - enabling DIIOP 251
 - functionality 6
 - history 6
 - hub configuration 29
 - infrastructure 29
 - infrastructure bottlenecks 30
 - integration options 35
 - integration techniques 37
 - Java classes 255
 - local access 249

- Portal integration benefits 8
 - remote access 249
 - rich text 26, 264
 - scalability 7
 - server infrastructure 29
 - single server 29
 - Sofor Interactive Portlet Builder for Domino 409
 - using credentials from Portal login 178
 - versions 6
 - Workplace 6
 - XML 110
 - XSLT 111
 - Domino application
 - choosing a design pattern 24
 - Domino applications
 - characteristics 16
 - example 47
 - features 39
 - integration into the Portal 9
 - migrating into the Portal 21
 - portalizing challenges 15
 - portalizing process 14
 - reusable components 22
 - transforming into the Portal 21
 - type of use 17
 - types 17
 - Domino classes 257
 - ACL 263
 - database 258
 - document 260
 - item 261
 - NotesException 263
 - session 257
 - view 259
 - Domino Designer 122, 149, 154
 - Domino Directory 252
 - Domino JSP tag issues 167
 - Domino JSP tag libraries 44, 137
 - overview 160
 - rich text 167
 - taglib 160
 - Domino JSP tags
 - adding a Domino view 176
 - adding to a portlet 172
 - customer contacts portlet 158
 - customer detail portlet 158
 - customer list portlet 157
 - customer sales activities portlet 159
 - data access 161
 - data input 163
 - database 162
 - docloop 163
 - document 162
 - ftsearch 162, 192
 - ifcategoryentry 164
 - ifdbrole 165
 - ifdocauthor 165
 - ifdocumententry 164, 192
 - implementation example 168
 - integration techniques 156
 - item 162
 - making a connection to a Domino database 174
 - nodocument 165
 - overview 138
 - performance 168
 - process control 164
 - runagent 165
 - sample applications 156
 - scalability 168
 - session 161
 - support 137
 - technologies 138
 - types 161
 - unid 162
 - utility tags 166
 - view 162, 192
 - viewitem 162
 - viewloop 163, 197
 - Domino Objects architecture 255
 - Domino toolkit for WebSphere Studio
 - adding custom tags 154
 - exposing Domino objects 154
 - features 154
 - Domino view tag 178, 192
 - Domino viewloop tag 185, 197
- ## E
- Eclipse 373
 - Embedding a view 123
 - Existing portlets
 - combining multiple portlets 121
 - Notes portlets 98
 - overview 63
 - QuickLinks 65
 - RSS portlet 107
 - skillset 64
 - Web clipping portlet 79

- Web page portlet 74
 - XML helper portlet 107
- F**
- Full text search 308
- H**
- Handling exceptions 264
 - Hide-when formulas 53
 - Horizontal portals
 - functionality 3
 - hosts file 253
 - HttpServletRequest 147
 - HttpServletResponse 148
 - HttpSession 147
- I**
- IBM HTTP plug-in 267
 - IBM Portal Builder
 - Presence Awareness 357
 - IBM Portlet Builder
 - authentication options 368
 - Click to Action 357
 - Click to Action parameters 366
 - configuring a portlet 361
 - connecting to a Domino server 364
 - considerations 358
 - creating a new portlet 361
 - features 357
 - form display options 368
 - implementation details 359
 - implementation example 371
 - installing 359
 - performance 359
 - rich text 358
 - scalability 359
 - selecting views and forms 366
 - skill set 358
 - view configuration 366
 - IBM Portlet Builder for Domino 46, 356
 - features 357
 - IBM WebSphere Portal Application Integrator 356
 - connectivity 357
 - features 357
 - Identify project requirements 39
 - IFrame 43, 369
 - iFrame 74, 265
 - displaying rich text 265
 - IFRAME tag 43
 - Improving performance 274
 - Input translation formulas 53
 - Input validation 53
 - Integration project
 - considerations 41
 - functional requirements 39
 - identifying requirements 39
 - portlet pattern 39
 - preparation 38
 - selecting the integration technique 40
 - steps 38
 - training 38
 - using existing portlets 42
 - Integration technique
 - selecting 40
 - Integration techniques 37
 - described 42
 - Interportlet communication 12
 - IOR 252
 - iSeries 253
- J**
- J2EE 138–139
 - application server 143
 - benefits 139
 - Client tier 143
 - combining with Domino 138
 - database tier 142
 - described 139
 - EJB container tier 142
 - overview 139
 - platform 139
 - portletlets 141
 - separation of tiers 140
 - standards 140
 - technologies included 141
 - Web container tier 143
 - Jakarta 150, 287, 342
 - Jakarta Commons pool 342
 - Java 45, 248
 - local access to Domino 249
 - remote access to Domino 249
 - Java 2 Enterprise Edition
 - See J2EE
 - Java Connector Architecture 141
 - Java development option

- Browsing Domino ACL example 329
- creating a session 313
- Domino toolkit for Java 306
- reusing a session 313
- technologies involved 226
- Java development options
 - portlets 226
- Java Domino classes 255
- Java Messaging Service 141
- Java programs 139
- JavaBean
 - creating 325
- JavaBeans 325
- JavaServer Pages
 - See JSP
- JCA 141
- JDBC 139, 357
- JMS 141
- JSP 143
 - adding a Domino view 177
 - adding tag library definitions 178
 - declarations 146
 - described 144
 - directives 145
 - elements 145
 - expressions 146
 - limitations 149
 - objects 147
 - processing 145
 - request 147
 - response 147
 - scriptlets 147
 - session 147
 - stateful interaction 147
 - tags 144
- jsp include tag 167
- JspWriter 148

L

- Layout and functional aspects 23
- Layout definition 23
- Load balancing 30
- load diiop command 251
- Log4j
 - appender 286
 - elements 285
 - features 285
 - implementation example 338

- layout 286
- logger 285
- Logging 282
 - implementaion example 338
 - JLog 282
 - log for java 284
 - log4j 284
 - message logging 283
 - sophisticated logging 284
 - WebSphere Portal log 282
- Lotus Discovery Server 219, 270
- Lotus Domino Toolkit for WebSphere Studio 154
- Lotus QuickPlace 270
- Lotus technology strategy 155
- LTPA Token 31

M

- MIMEEntity 266
- Mobile support 357
- Model-view-controller design pattern 227
- MVC 227
 - controller 228
 - implementation example 325
 - model 228
 - view 228

N

- NCSO.jar 256
- NCSOW.jar 256
- News portlet 25
- Notes 64
- Notes portlets 98
- Notes view portlet 43, 99
 - configuration 101
 - considerations 99
 - edit options 103
 - performance 99
 - results 106
 - rich text 99
 - setup 100
 - skillset 99
 - specifying view options 104
- NOTES.java 256
- NotesException class 263

O

- Object Pooling

- benefits 274
- benefits in portlet development 280
- client 276
- enhancing the pool manager 281
- implementation example 342
- load balancing 274
- object pool 275
- object pool manager 276
- performance comparison 350
- process 277–278
- query objects 281
- sharing resources 274
- virtual objects 275

Offline browsing support 357

Opening a database 259

P

Page

- adding a portlet 180

Page customization 33

Pages

- containers 34

- layout example 34

Pagination 24, 306, 310

Paging through the view 310

People Awareness 215

- chat 216

- finding documents 216

- implementing 218–219

- online presence 216

- online status 216

- people links 216

PeopleService tags 218

PeopleSoft 357

Performance 138, 359

Performance considerations 17

Personalization 23

Places

- described 11

- example 13

- organizing pages 11

Portal 2

- access to applications 2

- architecture 31

- architecture considerations 31

- authentication 31

- benefits 2

- Click to Action 202

- dedicated server for Domino 30

- description 2

- developing portlets using Java 45

- functionality 2

- horizontal 3

- integration techniques 37

- page 9, 11

- page aggregation 31

- page customization 33

- page example 10

- people awareness 215, 219

- Sametime integration 24

- scalability 17

- search 24

- server infrastructure 29

- skins 32

- styles 32

- themes 32

- users 244

Portal application 9

Portal infrastructure 29

Portalizing process 14

Portals

- infrastructure 29

Portlet API 229, 233

- class 234

- destroy() 235

- destroyConcrete() 235

- elements 233

- init() 234

- PortletResponse 235

- PortletSession 235

- service() 235

Portlet applications 232

Portlet Builders 46, 355

- Conet Portlet Factory for Domino 394

Portlet builders

- Aprtix Portlet Connector 411

- Bowstreet Portlet Factory for WebSphere 372

- IBM Portlet Builder for Domino 356

Portlet builders option

- advantages 356

- benefits 356

- disadvantages 356

- overview 356

- skillset 356

Portlet cooperation 12

Portlet events 244

Portlet pattern examples 25

- Portlet patterns 19
 - advantages 40
 - disadvantages 40
 - selecting 39
- Portlet perspective 168
- portlet.xml 188, 320
- PortletApplicationSettings object 241
- PortletConfig object 240
- PortletContext object 242
- PortletData 232, 239, 241
- PortletRequest 235
- PortletRequest object 236
- PortletResponse object 238
- Portlets 227
 - accessing Domino data 247
 - action events 245
 - action listening 183
 - ActionListener 183
 - actions 182
 - adding Domino JSP tags 172
 - adding NCSO.jar 320
 - adding search functionality 308
 - adding to a page 180
 - administering 230
 - application integration 28
 - applications 10, 232
 - applications 230
 - basic 169, 316
 - brokered cooperation 12
 - choosing the design pattern 24
 - Click to Action 199
 - coding guidelines 295
 - communicating 199
 - concepts 231
 - concrete 231
 - concrete portlet application 232
 - concrete portlet instance 231
 - configuration 240
 - considerations 28
 - container 230
 - cooperation 12
 - data 232, 241
 - described 9
 - design considerations 22
 - design patterns 19
 - developing in Java 318
 - developing in WebSphere Studio 153
 - development guidelines 295
 - display 20
 - displaying information 20
 - displaying rich text 265
 - enabling people awareness 219
 - events 244
 - examples of design patterns 25
 - exchanging information 12
 - exporting as .WAR file 323
 - folder structure 319
 - granting access 179
 - improving performance 274
 - in Java 313
 - installing 179
 - integrated 21
 - integrating the application logic 21
 - interportlet communication 199
 - J2EE development 141
 - layout and functional aspects 23
 - life cycle 234
 - link 19
 - logging 282
 - message events 247
 - messaging 12
 - migrated 21
 - model-view-controller 227
 - modes 10, 171, 227
 - MVC 325
 - MVC portlet 316
 - object pooling 274
 - opening an application 20
 - packaging guidelines 297
 - pagination 310
 - performance 274
 - performance considerations 302
 - portlet API 229
 - portlet.xml 188
 - preparing for Click to Action 204
 - previewing 181
 - rich text 26
 - running 11
 - scalability 274
 - session management guidelines 298
 - session specific data 231
 - states 10, 227
 - storing parameters 231
 - technical description 11, 227
 - Toolkit for WebSphere Studio 153
 - transactions 12
 - transferring data 12
 - transmitting information 199

- types 316
- UI components 22
- use cases 19
- use of persistent data 231
- user portlet instance 231
- web.xml 188
- window 227
- window events 246

PortletSession 239

PortletSession object 239

PortletSettings 231

PortletSettings object 240

PortletWindow object 243

Q

QuickLinks portlet 42, 65

- adding a link 69
- configuration 68
- considerations 66
- deleting a link 69
- example 70
- functionality 66
- implementation 67
- importing Internet Explorer favorites 69
- importing Netscape bookmarks 69
- setup 68

R

recycle method 256

Recycling Domino Objects 257

Recycling guidelines 257

Redbooks Web site 426

- Contact us xii

Reverse proxy 267

Rich Site Summary 43, 108

Rich text 26, 167, 264, 358, 399

- containing URLs 268
- created using a Notes client 266
- created using RichText applet 266
- displaying as plain text 266
- reverse proxy 267
- URL rewriting 268
- using Domino HTTP to render 266

RichTextItem 266

Row container 34

RSS portlet 107–108

- activating 117
- configuring 115

- copying 113
- locale 115
- parameters 116
- results 118

S

Sales Tracking application 47

Sametime 24, 215, 270

Sample application 47

SAP 357

Scalability 7, 17, 138

Search 3, 13, 24, 26, 192, 306, 357, 407

Search functionality 308

Security 8

Selecting the integration technique 40

Selecting the portlet pattern 39

Servlet API 229

Session information 148

Siebel 357

Single Sign On 31, 178, 377, 400

Sofor Interactive Portlet Builder

- considerations 410
- implementation details 410
- overview 409
- parameters 410

Sofor Interactive Portlet Builder for Domino 47, 409

Struts 150, 287

- Action Form Bean 288
- actions 288
- architecture 289
- basic elements 288
- benefits 287
- description 288
- in a Portal server 289
- in a WebSphere Studio 294
- MVC 287
- pages 288
- process flow 289

T

Tag Library Descriptor file 160

TLD 160

Types of Domino applications 17

U

UDDI directory 54

URL rewriting 148, 268

Using existing portlets option 63

V

Virtual objects 275

W

WAR 153

Web archive file (WAR) 54

Web Clipping portlet 43, 79, 121

administration portlet 79

authentication 89

caching 80

clipping types 86

configurin administration portlet 83

configuring 83

considerations 80

firewall options 87

national language support 84

overview 79

performance 81

results 95

rich text 81

runtime portlet 79

security 93

selecting content 94

setup 82

URL rewriting 91

URL to clip 85

Web page portlet 43, 74, 121

benefits 74

configuration 76

considerations 74

disadvantages 74

embedding page in the Portal 74

functionality 74

implementation 75

performance 75

results 78

rich text 75

setup 75

Web Services portlet 54

web.xml 188, 320

WebSphere Portal 1–2, 64

architecture 4

Click to Action 202

connectivity 4

deploying an application 178

Domino integration 2, 6

Domino integration benefits 7–8

federated access 2

log 282

out-of-the-box portlets 63

page 11

people awareness 219

personalization 2

portal types 2

Sametime integration 24

services 4

Single Sign On 31

using Domino LDAP 34

WebSphere Portal Content Publishing 357

WebSphere Portal Toolkit for WebSphere Studio 153

WebSphere Portlet Catalog 359

WebSphere Studio

building a portlet project 168

creating a folder structure 172

creating a portlet project 314

deployment descriptors 188

exporting a project 179

including custom JSP tag libraries 173

looking at content of a Domino database 176

perspective 168

portlet project 170

Struts applications 294

Web Content folder 172

WebSphere Studio Application Developer 138, 150
tools 150

WML 357

Workplace 4

application integration 5

described 4

example 13

functionality 4

personalization 5, 7

sales workplace example 50

Workspace

Domino integration 6

WSDL 199, 201

X

XML 107, 168

XML helper portlet 107

implementation example 109

performance 108

results 118

rich text 108
skillset 108
XSLT style sheet 111

Z
zSeries 253

Archived



Portaling Domino Applications for WebSphere Portal

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages

Archived



Redbooks

Portalizing Domino Applications for WebSphere Portal

Integrating existing Domino applications into a portal

This IBM Redbook describes how to integrate existing Domino applications into the IBM WebSphere Portal. We have coined the term 'portalizing' to describe this effort.

Portlet builders from IBM, Bowstreet, and CONET

We begin by explaining why portal integration is so useful for any company that has a Domino environment, and the importance of integrating Domino applications into the WebSphere Portal. We also explain some of the key concepts of portals and Domino application integration, and outline some recognized design patterns for Domino application integration.

Step-by-step integration techniques applied to practical scenarios

Next, we preview the recognized integration options which are described in detail later in the book. We also introduce the sample Domino application we used for our portalizing exercises throughout the book.

The following chapters present detailed discussions about the integration options currently available:

- Using existing portlets that ship with WebSphere Portal, including QuickLinks, Web page, Web Clipper, NotesView, XML/XSL Helper, and RSS portlet
- Using custom Domino JSP tag libraries
- Using Java programming
- Using portlet builders, including software products from IBM, Bowstreet, CONET, and others

For each of the integration options, we provide an overview of the technology, an introduction to the software and tools used, and step-by-step examples of using the techniques to portalize our sample Domino application.

This book is aimed at Domino application developers or anyone else who wants to learn how to portalize Domino applications.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks