Implementing Web Applications with CM Information Integrator for Content and OnDemand Web Enablement Kit

Introducing Web application implementation basics

Covering Information Integrator for Content and ODWEK

Real-world case study with sample codes

Wei-Dong Jackie Zhu
Javier Almaraz
Gordon Campbell
Hanaa Hammad
Nirva Morisseau-Leroy
Bud Paton

Redbooks

ibm.com/redbooks

International Technical Support Organization

Implementing Web Applications with CM Information Integrator for Content and ODWEK

August 2004

SG24-6338-00

Note: Before using this information and the product it supports, read the information in "Notices" on page xi.

First Edition (August 2004)

This edition applies to Version 8, Release 2 of IBM DB2 Content Manager for Multiplatforms, IBM DB2 Content Manager Information Integrator for Content for Multiplatforms (product numbers 5724-B19, 5724-B43), Version 7, Release 1, Modification 1 of IBM DB2 Content Manager OnDemand for Multiplatforms (product number 5697-G34).

Contents

Notices

Trademarks

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

Advanced Function Presentation™
AFP™
AIX®
AS/400®
DataJoiner®
Domino.Doc®
Domino®
DB2 Universal Database™
DB2®
@server®
@server®
ImagePlus®
Infoprint®
Informix®
IBM®
ibm.com®
iSeries™
Lotus®
Lotus Notes®
MQSeries®
OS/390®
OS/400®
Print Services Facility™
Redbooks™
Redbooks (logo) ™
Tivoli®
VideoCharger™
VisualInfo™
WebSphere®
z/OS®
zSeries®

The following terms are trademarks of other companies:

Intel Inside (logos) and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, 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.

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

Preface

This IBM® Redbook deals with Web application implementation using IBM DB2® Content Manager Information Integrator for Content Version 8 and IBM DB2 Content Manager OnDemand Web Enablement Kit. It is aimed at designers and developers of Content Manager systems.

In Part 1, we provide a brief introduction to Web application basics (Chapter 1), Content Manager (Chapter 2), and OnDemand (Chapter 3). In addition, we introduce the case study we use throughout the entire redbook when showing and demonstrating how to develop Web applications (Chapter 4).

In Part 2, we work specifically with Information Integrator for Content. We start with a brief programming overview (Chapter 5), and then show you how to get a quick start with developing Web application using the Java™ OO APIs provided by Information Integrator for Content (Chapter 6). The quick start chapter included detailed setup of WebSphere® Studio Application Developer, sample codes in defining attributes, item types, and items, and procedures to run and test the sample code. We show you how to build a generic Content Manager application using non-visual Java beans (Chapter 7). Using the case study described in Part 1, we develop a customized Web application (Chapter 8). Lastly, we show you how you can add a text search feature (Chapter 9) and document rendering (Chapter 10) in your application.

In Part 3, we work specifically with OnDemand Web Enablement Kit (ODWEK). We start with an overview of Web enabling OnDemand (Chapter 11), installation and configuring of ODWEK (Chapter 12), and developing a Web application using ODWEK (Chapter 13).

There are many sample codes provided along with this redbook. They provide the basic concept and code in developing Content Manager and OnDemand Web applications. In many instances, you can learn by simply working or reading through the codes. We recommend that you check them out along with the redbook. Please note that it is not the responsibility of the IBM Technical Support Team or the IBM International Technical Support Organization (the IBM Redbook team) to provide support of the sample codes in this redbook. The sample codes are meant to be used as-is.

Enjoy the world of Content Manager and OnDemand Web application development!

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, San Jose Center.

Wei-Dong Zhu (Jackie) is a Content Manager Project Leader with the International Technical Support Organization at the Almaden Research Center in San Jose, California. She has more than 10 years of software development experience in accounting, image workflow processing, and digital media distribution. She holds a Master of Science degree in Computer Science from the University of Southern California. Jackie joined IBM in 1996. She is a certified Solution Designer for IBM DB2 Content Manager.

Javier Almaraz is an Advisory IT Specialist in IBM Mexico with 6 years of experience in the Information Technology field within Informix® Software Mexico and IBM Software Group. He holds a degree in Computer Systems Engineering from ITESO University. His areas of expertise include J2EE design and development, IBM Content Manager, and Informix. He is a certified Solution Designer for IBM DB2 Content Manager.

Gordon Campbell is a Senior I/T Specialist in the United States working for IBM on the Advanced Technical Support Team for Content Management. He holds a Master's degree in Computer Science from the American University in Washington D.C. He has over ten years experience supporting VisualInfo™ and CM, concentrating primarily on performance, sizings, and custom applications. This is the second IBM Redbook he has participated in, and he has presented numerous times at conferences and customer sites.

Hanaa Hammad is an IT Specialist from Cairo, Egypt, on the IBM BIS team. She has over 16 years of experience in the data management and communications fields, and has worked at IBM for 3 years. Her areas of expertise include data management, mainframe communication, and Web application development using IBM tools on a variety of platforms. She holds a Masters Degree in Telecommunication Software from the INRS-Telecommunications University of Quebec in Canada.

Nirva Morisseau-Leroy is a Senior Database Administrator and Java Application Developer with the Cooperative Institute for Marine and Atmospheric Studies (CIMAS) of the University of Miami, in Miami, Florida. She has over 18 years of experience in software and database analysis, design, and implementation, and has published four Oracle related books. She holds a Master of Science degree in Computer Science and is a PhD student in computer science at Florida International University (FIU), in Miami, Florida.

Bud Paton is a Consulting IT Specialist located in Brookfield, Connecticut currently with the Content Management East Team. He has 20 years of experience as a System Engineer / IT Specialist and 10 years of experience with IBM OnDemand. He is Product Certified in OnDemand. His areas of expertise include OnDemand, Advance Function Presentation, and print data streams.

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

Mike Reece
IBM Global Services, Gaithersburg, MD

Kshitij Vichare
Ying-Pong Chen
Chandra Shekar
Hua Chen
Greg Felderman
Jerald Schoudt
Byrant S Daniel
IBM Software Group, San Jose, CA

Ronnie Roberts
IBM Software Group, Atlanta, GA

Hakan Carlberg
IBM Nordic & SWG Services, Sweden

Dr. Naphtali Rishe
Florida International University, Florida

Dr. Mark Powell
NOAA/Hurricane Research Division, Florida

Dr. Joseph Prospero
University of Miami, Florida

Yvonne Lyon
Deanna Polm
Mary Comianos
Emma Jacobs
International Technical Support Organization, San Jose 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. QXXE Building 80-E2
  650 Harry Road
  San Jose, California 95120-6099

Part 1

Introduction

In this part of the book, we introduce Web application basics, provide an overview for IBM DB2 Content Manager and IBM DB2 Content Manager OnDemand, and describe the case study we use to develop our sample applications.

1

Web application basics

The use of the Internet has spawned a whole new generation of application programs that allow interaction over the World Wide Web. Business (e-business or e-commerce) activity over the Internet has expanded tremendously over the last few years. Web programming techniques have matured rapidly to provide enterprise-level functionality, reliability, and scalability in business applications.

In this chapter, we introduce the Web application concepts and technologies.

We cover the following topics:

- Overview
- Java servlet
- JavaServer Pages (JSP)
- JavaServer Faces technology (JSF)
- Model-View-Controller architecture
- Java 2 Platform Enterprise Edition (J2EE)
- Struts

1.1 Overview

A *Web application* is a collection of programs and resources. Typically, a Web application is designed to perform a specific task. A fundamental characteristic of Web applications is that they use HTTP for their core communication protocol and deliver Web-based information to the user. A simple Web application may consist of static HTML pages and a Common Gateway Interface (CGI) interface or PHP (Hypertext Preprocessor). PHP is a general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

Complex Web applications can consist of static HTML pages, XML, logos and images, Web components such as Microsoft® Component Object Model (COM) and Distributed Component Object Model (DCOM), Java servlets and JavaServer Pages (in the Java 2 Platform), software components such as JavaBeans and Enterprise JavaBeans, supporting utility classes, and various other resources. In this chapter, we focus on building Web applications that use technologies based on the Java programming language.

1.1.1 Common features

Here is a list of features common to Web applications:

- They use a Web site as a front-end.
- A Web application may be an *Internet* or an *Intranet* application.
- Web application client business logic is executed in a back-end server and not in the client machine.
- HTTP request and validation — Web applications communicate with users via the HTTP protocol. HTTP request handling is done by a Web server.
- There exists a unique session and relationship between a Web user and a Web application.
- Communicate with other Web resources such as databases (IBM UDB DB2, Oracle, Microsoft SQL Server, and MySQL) and other applications.
- They support interactive data entry, transactions, and presentation through any standard Web browser.
- A Web application exists as a structured hierarchy of directories.

There are two types of Web applications:

- **Presentation-oriented:** In this book, we show you how to develop presentation-oriented Web applications. A presentation-oriented Web application is of two types:

  a. One that consists of static HTML pages.

   b. One that generates dynamic Web pages containing various types of markup language (such as HTML and XML).

- **Service-oriented:** A service-oriented Web application implements the end-point of a Web service.

The steps for developing a Web application are as follows:

- Design the Web application, which consists of various Web components.
- Implement the Web components.
- Build the Web application components along with any static resources (for example, images) and helper classes referenced by the component.
- Package the Web application.
- Install or deploy the application into a Web container.
- Access a URL that references the Web application.

1.1.2 Web server

A *Web server* is a program that receives requests arriving over a network, executes some logic based on the parameters in the request, and returns the results back to the client application. Each Web application generally corresponds to a distinct path within the Web server and is rooted at a specific path within a Web server. Typically, a Web user opens a Web browser and sends an HTTP request to a Web server in the form of a URL. In turn, the server answers by sending a response back to the client with the request information (see Figure 1-1). The response generated by the Web server is handled and displayed by the browser.

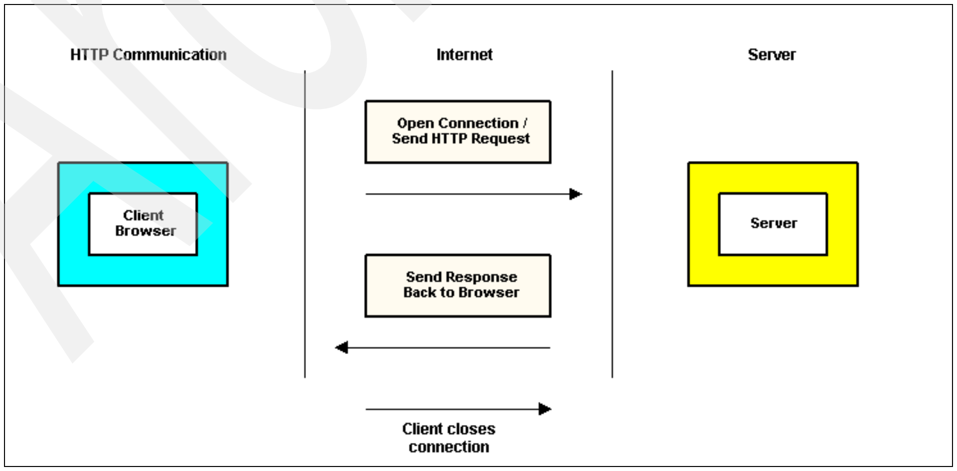


Figure 1-1 HTTP communication

1.1.3 Web application packaging

Web applications are packaged into a Web application archive (WAR), that is, a JAR file similar to the packaging format used for Java class libraries. A WAR file consists of a variety of files such as Java classes, HTML, GIF, JPEG. In later chapters, we show how you can create WAR files.

1.1.4 Web services: Service-oriented application

Distributed computing technologies such as the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (Java RMI), and Microsoft Distributed Component Object Model (DCOM) provide the capability to integrate applications within homogenous environments inside local area networks (LANs).

CORBA is an open distributed object computing infrastructure being standardized by the Object Management Group (OMG). Java RMI enables developers to build distributed Java-based applications. Microsoft DCOM, previously called “Network OLE”, is based on the Open Software Foundation's DCE-RPC Specification and is designed for use across multiple network transports, including Internet protocols such as HTTP.

One of the major problems with these distributing computing technologies was the lack of interoperability between the protocols used by these technologies. The emergence of Web services introduces a new mechanism to enable the exchange of information over the Internet. Web services are based on the concept of service-oriented architecture (SOA), which is the latest evolution of distributed computing. Using industry standards, Web services encapsulate applications and publish them as services. Web services are services that are based on the Extensible Markup Language (XML) messaging. These services deliver XML-based data that can be dynamically accessed using a wide range of computing platforms and hand-held devices.

The basics characteristics of Web services are as follows:

- Based on XML messaging — Data exchange are defined in XML.
- Provide a cross-platform integration of business applications.
- Language-independent and flexible — Developers can use any common language such as COBOL, Perl, Python, Java, C, C++, C#, and Visual Basic.
- Use of industry-standard protocols such as HTTP.

Figure 1-2 shows the Web services operational mode with roles and relationships.

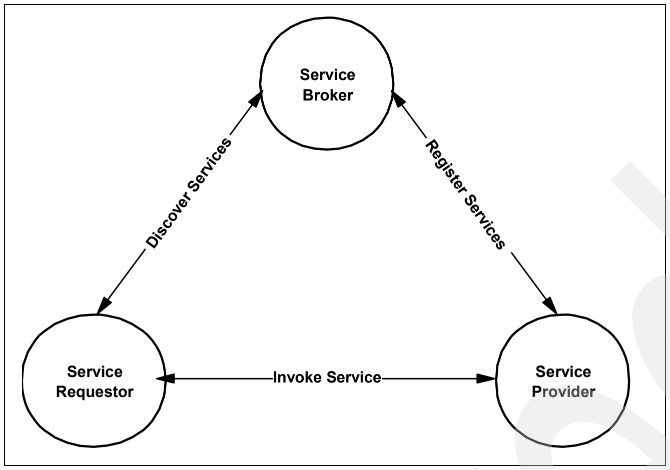


Figure 1-2 Web services operational mode

1.1.5 Web services standards

Web services technology is based on the following specifications:

- **XML:** In 1998, the Worldwide Web Consortium (W3C) officially endorsed XML as a standard data format. XML is the standard for structuring data, content, and data format for electronic documents. For more about XML, see:

  `http://www.w3.org/XML`

- **Simple Object Access Protocol (SOAP):** A standard protocol for light weight XML-based messaging protocol. SOAP enables exchanges of information between two or more peers and enables them to communicate with each other in a decentralized and distributed application environment. To learn more about SOAP, see:

  `http://www.w3.org/TR/soap`

- **Web services Definition Languages (WSDL):** An XML format for describing the network services. WSDL defines a binding mechanism to attach a protocol, data format, abstract message, or set of end-points defining the location of services. To learn more about WSDL, see:

  `http://www.w3.org/TR/wsdl`

- **Universal Description, Discovery, and Integration (UDDI):** Defines the standard interfaces and mechanisms for registries. Similar to the yellow pages, UDDI registries are used by businesses to register their Web services. UDDI working group includes IBM, HP, SAP, Oracle, and Microsoft. To learn more about UDDI, refer to:

  `http://www.uddi.org`

- **Electronic Business XML (ebXML):** A project jointly initiated by UN/CEFACT (The United Nations body for Trade Facilitation and Electronic Business) and OASIS to standardize XML business specifications. To learn about ebXML, refer to:

  `http://www.ebxml.org`

Many vendors (IBM, Microsoft, Oracle, HP, and etc.) support Web services. If you want to learn about Web services, we recommend that you start with the Java Web Services Developer Pack (Java WSDP). Java WSDP is a free integrated toolkit that allows Java developers to build and test XML applications, Web services, and Web applications with the latest Web services technologies and standards implementations. To learn more about Java WSDP, refer to:

`http://www.java.sun.com/features/2002/03/jwsdp.html`

1.2 Java servlet

A *servlet* is a Java class that extends the capabilities of a Web server. The file extension for a servlet is .java. The Web servers host applications that are accessed via a request-response programming model. Servlets can respond to any type of request, but they are typically used to extend the applications hosted by Web servers.

To learn more about servlet concepts, see the following URLs:

```
http://java.sun.com/products/servlet/index.jsp
http://www.jcp.org/aboutJava/communityprocess/final/jsr053/index.html
```

In later chapters of this redbook, we show you how to develop Java servlets.

1.2.1 Servlet container

Servlets are Java classes that are dynamically loaded and executed by a servlet container. *Servlet containers* or servlet engines, are Web server extensions that provide servlet functionality. Servlets classes are platform-independent and can be loaded by Java technology-enabled Web or application servers.

The servlet container can be a part of a Web server or application server. All servlet containers must support HTTP as a protocol for requests and responses, but additional protocols such as HTTPS (HTTP over SSL) may also be supported. Servlets are highly used components in J2EE Web applications. See 1.6.1, “J2EE architecture” on page 17, to learn more about J2EE.

1.2.2 Servlet API

The Java servlet API consists of the javax.servlet and javax.servlet.http packages, which provide interfaces and classes for writing servlets. All servlets must implement the servlet interface, which defines life-cycle methods. The API also provides the GenericServlet class that allows you to write generic services. The GenericServlet class was designed to provide simple versions of the servlet life-cycle methods such as init( ) and destroy( ), and of the methods in the ServletConfig interface.

1.2.3 Servlet life cycle

A servlet runs within a Web server. The life cycle of a servlet is controlled by the Web container in which the servlet has been deployed. When a Web user makes a request, the container performs the following steps:

1. Loads the servlet class if an instance of the servlet does not exist.
2. Creates an instance of the servlet class.
3. Calls the init( ) method of the servlet to initialize an instance of the servlet.
4. Invokes the service method using a request and response object as input parameters.
5. Calls the destroy( ) method when the servlet object needs to be removed from the Web container.

1.2.4 Sharing information

Servlets, similar to any other Java class, use objects to perform tasks and share Web resources. They can use Java beans, regular Java classes (sometimes, called helper classes), share objects that are attributes of a public scope, use data stores, and invoke other Web resources. Objects that need to be shared are maintained as attributes of four scope objects. Use the getter and setter attribute accessor methods to access the attributes of the class representing the scope.

Table 1-1 shows the scope objects.

Table 1-1 Scope objects

Scope object	**Class**
Web context	javax.servlet.ServletContext
session	javax.servlet.http.HttpSession
request	subtype of javax.servlet.ServletRequest
page	javax.servlet.jsp.PageContext

1.2.5 Filtering requests and responses

The Java servlet API provides the capability to filter requests and responses. A filter object allows you to transform both the header and content of a request or response and to attach functionality to Web resources. Use filters for authentication, logging, image conversion, data compression, encryption, tokenizing streams, and XML transformation applications. The main tasks of filters are as follows:

- Query the request object.
- Block the request and response object pair.
- Modify request headers and data.
- Modify response headers and data.
- Interact with external resources.

Some examples of filtering components are (Servlet Specification 2.4):

- Authentication filters
- Logging and auditing filters
- Image conversion filters
- Data compression filters
- Encryption filters
- Tokenizing filters
- Filters that trigger resource access events
- XSL/T filters that transform XML content
- MIME-type chain filters
- Caching filters

1.2.6 Maintaining client and application state

Many interactive applications require that the client's state — that is, the client's requests — be associated and maintained. The Java servlet API provides mechanisms to manage user sessions.

In a servlet, a user's session is represented by the *HTPPSession* object. The HTTPSession interface provides methods that allow the application to identify a user across several page requests. To access a session, use the getSession( ) method of a request object.

In some cases, Web applications may need to maintain not just session-related state, but state that is associated with an entire application. For example, a pool of database connection objects may need to be shared across multiple sessions.

Each Web application corresponds to one *ServletContext* instance. There is one ServletContext object per Java Virtual Machine. The ServletContext interface defines a set of methods to communicate with the servlet container. A handle to this object can be retrieved by all servlets and JSPs in the application by calling the getServletContext( ) method. For example, use a ServletContext object to get the MIME type of a file, dispatch requests, or write to a log file.

1.2.7 Packaging servlets

Similar to a Web application, servlets are packaged into a Web application archive (WAR). We show you about build the WAR files that contains servlet classes in later chapters.

To learn more about the Servlet Specification 2.4, see the following URL:

`http://www.jcp.org/aboutJava/communityprocess/final/jsr154/index.html`

1.3 JavaServer Pages (JSP)

A *JavaServer Page* (JSP) enables Java code to be mixed with static HTML or XML templates. JSP is built on the Java servlet technology. The purpose of JSP is to generate dynamic content in the page, while the markup language handles structuring and presentation of data. The main features of JSP are:

- A text-based language for developing JSP pages. JSPs are documents that describe how to process a request and construct a response.
- An expression language that provides mechanisms to access server-side objects and to define extensions to the JSP language.

A JSP page contains static template data (HTML, SVG, WML, and XML) and JSP elements. The file extension for a JSP page is .jsp. JSP programs are executed by a JSP engine that runs within the Web server. The JSP engine generally runs as a servlet that is invoked whenever the request URL ends with a .jsp extension. This invocation scheme is known as extension mapping.

JSP has two categories:

- **Tag Libraries:** Define declarative and modular functionality that can be reused by any JSP page.
- **JavaServer Pages Standard Tag Library:** Defines a standard set of tags that developers can use on multiple JSP containers. See 1.3.2, “JSP Standard Tag Libraries (JSTL)” on page 12, to learn more about JSTL.

1.3.1 JSP elements

The JSP Specification defines HTML-like or XML tags, called elements, that enclose the code in JSP:

- **Directive elements:** These provide information to the JSP container about the page.
- **Scripting elements:** These are elements in the page that include Java code.
- **Action elements:** These are known as Standard Actions because they are defined by the JSP Specification. Action elements are used to include applets or Java beans in the HTML page generated by a JSP page.

Some of the advanced features of JSP are as follows:

- **Expression language:** Provide a way to use run-time expressions outside JSP scripting elements. Expression language is a new feature of JSP 2.0 Specification.
- **Custom actions:** Provide a way to encapsulate Java code so that the page designer does not need to know Java syntax. JSP provides a way for Java developers to create their own custom actions or tag extensions. Classic tag extensions are from JSP 1.2 whereas Simple tag extensions are from JSP 2.0.
- **JSP Standard Tag Library (JSTL):** Use JSTL to avoid multiple developers of creating conflicting tag libraries for basic actions.

1.3.2 JSP Standard Tag Libraries (JSTL)

JSTL and custom tags are the preferred way for performing dynamic tasks within a JSP program. JSTL defines a standard set of tags that developers can use on multiple JSP containers. Note that a standard tag library is more likely to have an optimized implementation.

JSTL contains several sub libraries that handle flow control, URL management, XML document manipulation, internationalization, SQL database access (query, update, transactions), and functions such as String and Array. JSTL supports iteration and conditionals tasks, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating existing custom tags with JSTL tags.

Custom tags are user-defined JSP language elements that encapsulate recurring tasks. They are distributed in a tag library that contains the definitions of the custom tags as well as the objects that implement the tags.

Use JSTL or custom tags to access databases, services such as e-mail and directories, and flow control. In earlier versions of JSP technology, such tasks were performed with Java beans components in conjunction with scripting elements. Although scripting elements are still available in JSP 2.0, the current trend or preferred approach is to use JSTL or custom tags instead of scripting elements.

1.3.3 JSP life cycle

A JSP page services requests as a servlet; thus, the life cycle of a JSP program is determined by the Java servlet technology.

1.3.4 JSP translation and compilation

At translation and compilation time, a JSP class becomes a servlet class. Once the JSP page has been translated, it follows the servlet life cycle as described in 1.2.3, "Servlet life cycle" on page 9.

1.3.5 JSP advantages

The advantages of using JSP are manifold:

- Strong support for template data.
- Excellent expression language to access Java objects.
- Flexible and easy to extend.

JSP pages are used for views in the Model-View-Controller model applications. See 1.5, "Model-View-Controller architecture" on page 15, to learn more about this model.

1.3.6 Packaging JSPs

Similar to a Web application and servlet, JSPs are packaged into a Web application archive (WAR).

In this section, we presented a brief overview on the JavaServer Pages technology.

To learn more about JSP concepts, go to the following URL:

`http://www.java.sun.com/products/jsp/docs.html#syntax`

1.4 JavaServer Faces technology (JSF)

JavaServer Faces technology is a user interface framework for building Web applications. It is a model for rendering components in different kinds of HTML, or different markup languages and technologies. JSF architecture is designed to be independent of specific protocols and markup.

The JSF technology includes the following:

- A set of APIs that represents user interface (UI) components and provides mechanisms to manage their state, handle events and input validation, define page navigation, and support internationalization and accessibility.
- A JavaServer Pages (JSP) custom tag library to express a JavaServer Faces interface within a JSP page.

1.4.1 JSF APIs

JSP APIs consist of the following packages:

javax.faces	Contains top level classes for the JavaServer Faces API.
javax.faces.application	Contains APIs to link the application's business logic objects to JavaServer Faces as well as pluggable mechanisms to manage the execution of an application that is based on JavaServer Faces.
javax.faces.component	Contains APIs for user interface components.
javax.faces.component.html	Contains concrete base classes for each valid combination of component and renderer.
javax.faces.context	Contains classes and interfaces defining per-request state information.
javax.faces.convert	Contains classes and interfaces defining converters.
javax.faces.el	Contains classes and interfaces for evaluating and processing reference expressions.
javax.faces.lifecycle	Contains classes and interfaces defining life cycle management for the JavaServer Faces implementation.
javax.faces.event	Contains interfaces describing events and event listeners, and concrete event implementation classes.

javax.faces.render	Contains classes and interfaces defining the rendering model.
javax.faces.validator	Contains interface defining the validator model, and concrete validator implementation classes.
javax.faces.webapp	Contains classes required for integration of JavaServer Faces into Web applications, including a standard servlet, base classes for JSP custom component tags, and concrete tag implementations for core tags.

To learn more about JSF, go to the following URL:

http://www.java.sun.com/j2ee/javaserverfaces/download.html

1.5 Model-View-Controller architecture

Model-View-Controller (MVC) is well-suited for interactive Web applications, especially Web applications in which a Web user interacts with a Web site, with multiple iterations of screen page displays and multiple round-trips of requesting and displaying data.

MVC organizes an application into three separate layers:

- **Model:** The application model with its data representation and business logic.
- **View:** The module(s) to create the views for data presentation and user input. The views focus on rendering the contents of the model.
- **Controller:** The layer that dispatches requests and controls flow. It defines application behavior, interprets user requests, maps these requests into actions that the model must perform. In a Web application, they are HTTP get and post requests to the Web tier. Some applications have one controller for each set of related functionality.

1.5.1 MVC advantages

The advantages of using the MVC model are manifold:

- Separate design concerns — Data persistence and behavior (the model), presentation (the view), and control are clearly defined and separated.
- Decreased code duplication.
- Centralized control.
- Necessary tools to build flexible applications, thus more easily modifiable.
- Allows developers with different skill sets to focus on their core skills.

The MVC model is widely used for interactive applications. In subsequent chapters of the redbook, we show you how to use the MVC model to build Web applications that access and manipulate contents in Content Manager, OnDemand, and UDB DB2.

Figure 1-3 illustrates the MVC architecture.

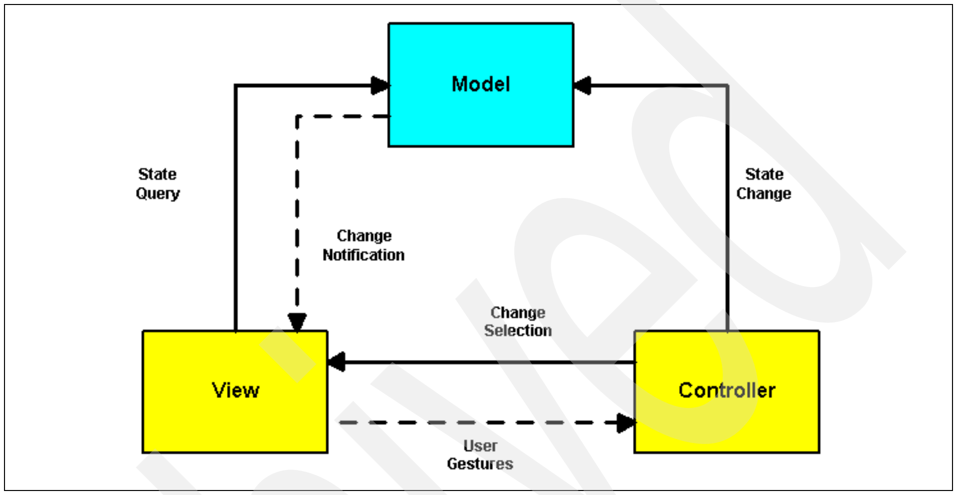


Figure 1-3 Model-View-Controller architecture

1.6 Java 2 Platform Enterprise Edition (J2EE)

Java 2 Platform, Enterprise Edition (J2EE) specifies a set of component technologies (Enterprise JavaBeans, JavaServer Pages, Java servlets, and Java API for XML Based RPC) that allows developers to build multi-tier enterprise applications. The platform has added support for Web service-specific components and technologies (J2EE 1.4.x). See 1.1.4, “Web services: Service-oriented application” on page 6, to learn about Web Services.

The J2EE platform specifies support for the following capabilities:

- **Multi-tier model:** Multi-tier distributed applications and Web services. The platform defines different tiers: client tier, one or more middle tiers, and a back-end tier.
- **Component-based development:** Wireless clients, rich Java-based GUI client, and non-Java clients.
- **Container-based component management:** Containers are runtime environments that provide services to components, such as, session management APIs, database transactions, security, standardized access to EISs, and standardized deployment of applications.

- **Portability:** Enables development of distributed, portable, and interoperable enterprise applications and Web services.
- **WS-I standard for interoperability:** WS-I is an open organization chartered to promote Web services interoperability across platforms, operating systems, and programming languages. The J2EE 1.4 platform includes specifications and technologies that support Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL), Universal Discovery, Description, and Integration (UDDI) specification, and Electronic Business using eXtensible Markup Language (ebXML). The platform includes technologies such as Java API for XML-Based RPC (JAX-RPC), Java API for XML Messaging (JAXM), SOAP with Attachments API for Java (SAAJ), Java API for XML Registries (JAXR), Java API for XML Processing (JAXP), and Java API for XML Binding (JAXB).

1.6.1 J2EE architecture

J2EE applications are generally considered to be three-tiered (See Figure 1-4) applications because they are distributed over three locations: client machines, the J2EE-based server machine, and the database or legacy machines at the back-end. Three-tiered applications extend the standard two-tiered client and server model by placing a middle-tier or an application server between the client application and back-end data storage.

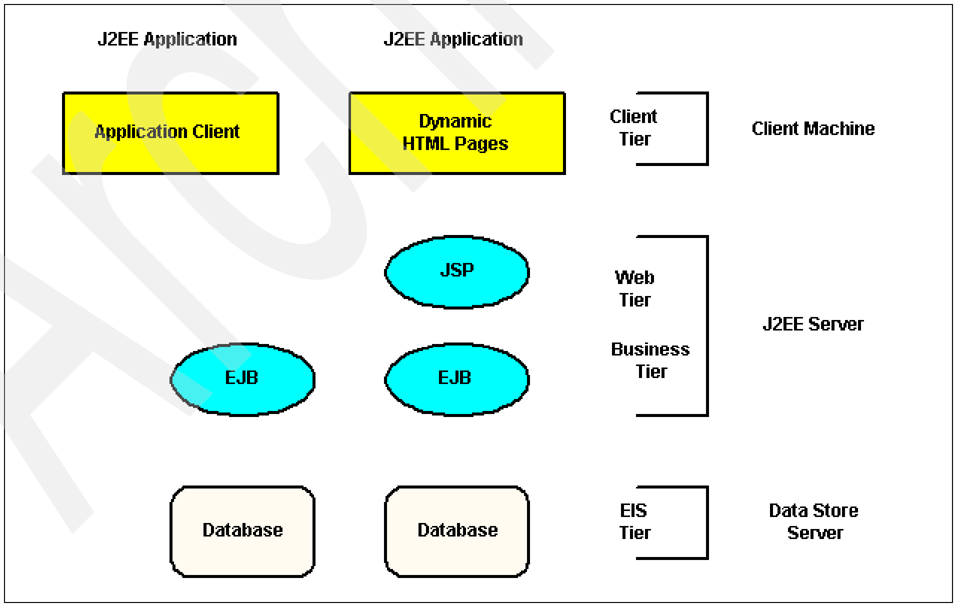


Figure 1-4 J2EE multi-tier architecture

The Model-View-Controller architectural design is recommended for interactive applications. Most Web-tier application framework use the MVC model or some variation of it. Interactive J2EE applications use the MVC architecture.

1.6.2 Component technologies

The J2EE platform supports the following application component technologies:

- **Client component:** Applets, Java-Web-Start-enabled rich clients and Java-based clients, and non-Java-based clients.
- **Web component:** Servlet and JSPs are Web components that run on the server. Servlets and JSPs are J2EE Web components. Note that static HTML and applets are not considered as Web components by the J2EE Specification.
- **Middle-tier components:** J2EE provides support for Enterprise Java beans (EJB) components. EJBs are business components that run on the server. There are of four EJB types: session bean, entity bean, message-driven bean, and timer bean.

Note that the client and server tiers can include components based on the Java bean component architecture. Java beans (see Figure 1-5) are used to manage the data flow between application clients and components running on a J2EE server or between a J2EE server and a database (see Figure 1-6).

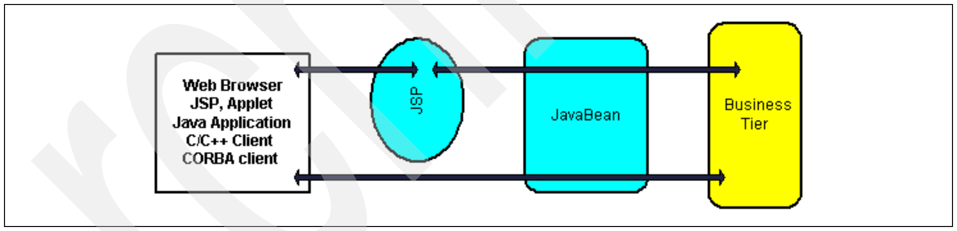


Figure 1-5 J2EE Web-tier components and Java beans

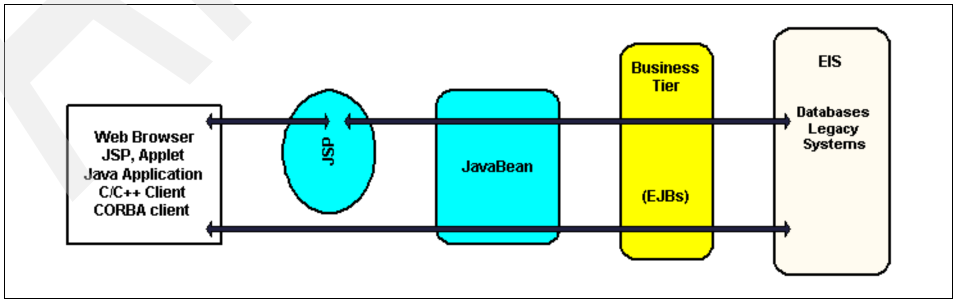


Figure 1-6 J2EE business-tier, Java bean, and Enterprise Information System (EIS)-tier

Note that, similar to static HTML pages and applets, Java beans are not considered J2EE components by the J2EE Specification.

1.6.3 Packaging J2EE applications

A J2EE application is packaged into an Enterprise Archive (EAR). This file may contain software components, Web applications, and resources.

To learn more about J2EE, go to the following URL:

    http://www.java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf

1.7 Struts

Struts is an open source framework for building Java Web applications. This framework is based on standard Java technologies such as Java servlets, Java beans, resource bundles, and XML, and various Jakarta Commons packages. Struts provides the control layer for Web applications. Developers can use data access technologies of their choice such as JDBC, EJB, and object-relational bridge to provide data access. Struts also supports the use of various presentation technologies such as JavaServer Pages (including JSTL and JSF), Velocity Templates, and XSL Transformations (XSLT).

The JavaServer Standard Tag Library (JSTL) is a set of JSP tags. JavaServer Faces (JSF) is a specification. Both technologies are complementary to Struts.

Velocity is a Java-based template engine that allows developers to use the template language to reference object defined in Java code. Velocity allows developers of different skills to work in parallel. XSLT is a language for transforming XML documents into other XML documents.

To learn more about Struts, go to the following URL:

    http://www.jakarta.apache.org/struts

2

Content Manager overview

In this chapter, we introduce Content Manager architecture and the various components that make up a Content Manager system. In addition, we cover the key concepts, including data modeling and document routing, that you should be familiar with, in order to implement Content Manager applications. We also briefly introduce Content Manager Information Integrator for Content.

We cover the following topics:

- Overview
- Data modeling
- Document Routing
- Security and privilege sets
- Information Integrator for Content

2.1 Overview

IBM DB2 Content Manager (Content Manager) provides a scalable, enterprise-wide repository system for the capture, creation, organization, and retrieval of business content. In addition, it provides workflow routing, life cycle management, sharing and reusing of the content. The digitized content supported by Content Manager includes HTML and XML-based Web content, images, electronic office documents, and rich media such as digital audio and video.

Content Manager uses a triangular architecture as shown in Figure 2-1. Client applications, running either in end-user desktops or in mid-tier application servers, use object-oriented C++ and Java APIs to invoke Content Manager services that are divided between a Library Server and one or more Resource Managers. The *Library Server* manages the content metadata and is responsible for access control to all of the content and interfaces to one or more Resource Managers. The *Resource Managers* manage the content objects. Both the Library Server and the Resource Managers can utilize LDAP services for user management and access control.

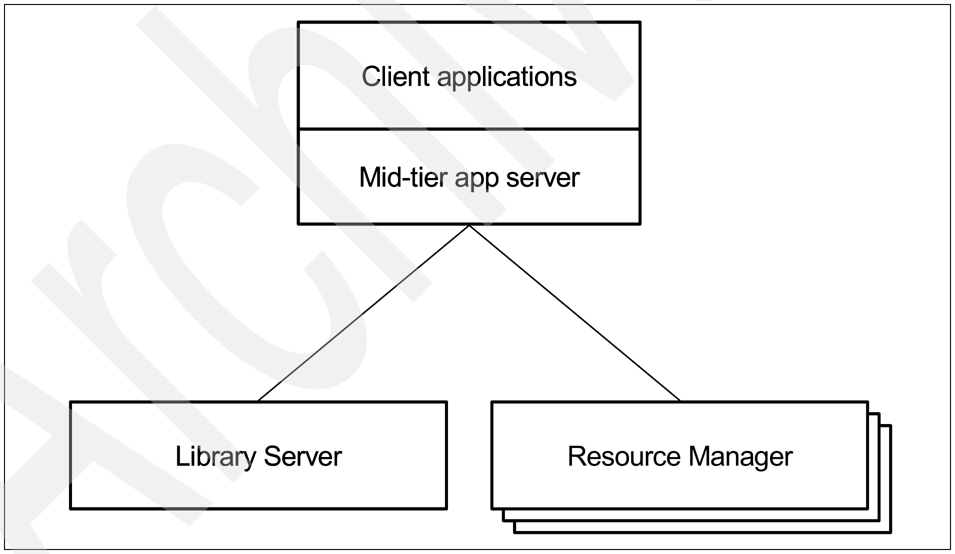


Figure 2-1 Content Manager triangular architecture

All access to the Library Server is through the database query language SQL. Results from the queries are passed back to the client with object tokens, locators for requested content that the user is authorized to access. The client then directly communicates with the Resource Manager using the Internet protocols HTTP, FTP, and FILE.

2.1.1 System components

There are three main components that make up a Content Manager system. They are:

- Library Server
- Resource Manager
- Content Manager clients (System Administration Client, Client for Windows®, eClient, and customized clients)

In the following sections, we explore each Content Manager component.

Library Server

The Library Server is the *key* component of a Content Manager system. It consists of a Library Server database and database stored procedures (Figure 2-2). It acts as the library catalogs do in a real library, and it is where you define the information that you store in your library. The majority of the Library Server code that interacts with the underlying Library Server databases is implemented as the database stored procedures.

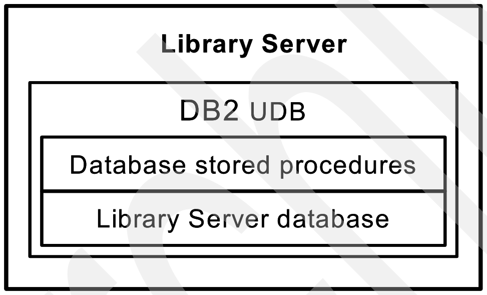


Figure 2-2 Library Server component architecture

The Library Server is the central system control point. It stores, manages, and provides access control for objects stored on one or more Resource Managers. The Library Server processes requests, such as update and delete from one or more clients, and maintains data integrity between all of the components in the Content Manager system.

When the user is accessing objects in a Content Manager system, this is directly controlled by the Library Server. A Library Server relies on a relational database management system (RDBMS), such as IBM DB2 Universal Database™ to manage the content and perform parametric searches, text searches, and combined (parametric and text) searches. All access is done through SQL.

A Content Manager system requires only one Library Server. The Library Server can run on the Windows, AIX®, or Solaris platforms.

Resource Manager

The Resource Manager stores objects for Content Manager. A typical Resource Manager consists of a file system, a Resource Manager database, a WebSphere Application Server, and Resource Manager servlets (Figure 2-3). Users store and retrieve digital objects on the Resource Manager by routing requests through the Library Server. A single Library Server can support multiple Resource Managers. Content can be stored on any of the Resource Managers.

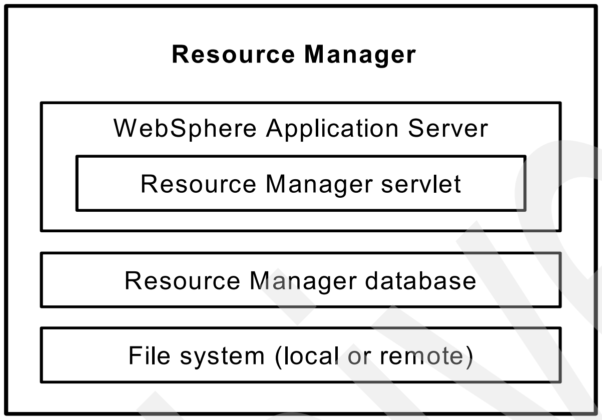

Resource Manager
WebSphere Application Server
Resource Manager servlet
Resource Manager database
File system (local or remote)

Figure 2-3 Resource Manager component architecture

The Resource Manager can be a LAN-based Resource Manager (LBOS), an IBM Video Charger, or a Content Manager Hierarchical Storage Management System, which makes use of the Tivoli® Storage Manager (TSM) to manage tertiary storage subsystems such as optical and tape libraries. Besides these integrated “storage managers,” support is provided for generic local and remote file systems as externally managed storage collections. Resource Managers can be distributed across networks to provide convenient and efficient user access.

Starting from Content Manager Version 8, the Resource Manager works with WebSphere Application Server to provide content retrieval and archiving services.

Content Manager clients

There are four main types of Content Manager clients:

- System Administration Client
- Client for Windows
- eClient
- Customized client

System Administration Client

The System Administration Client is used to perform the majority of the system administration tasks for a Content Manager system. With the System Administration Client, system administrators can perform the following tasks:

- Define data models.
- Define users and their access to the system.
- Define and configure Library Server and Resource Managers.
- Manage object storage and retrieval in the system.

The System Administration Client component can be installed on any workstations that have other components installed, or it can be installed standalone without other components.

Client for Windows

Client for Windows, also referred to as Windows client, is a desktop thick client that provides out-of-the-box capabilities for supporting production-level Content Manager applications. It exploits two-tier client server architecture, and uses ICM C++ APIs to access services provided by Library Server and Resource Managers (see Figure 2-4).

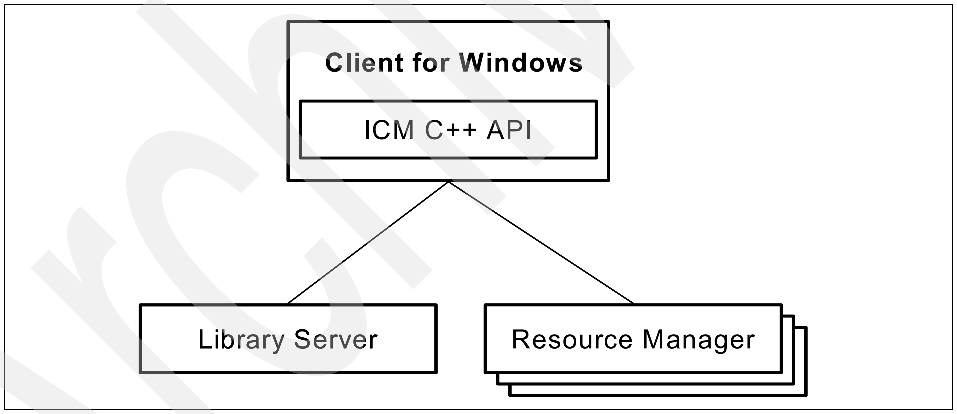


Figure 2-4 Client for Windows architecture

Client for Windows is installed on a Windows system. With the Windows client, you can perform the following tasks:

- Scan and import documents into the system
- View and work with documents
- Perform simple workflow functions
- Store and retrieve documents

Client for Windows user exit routines allow you to specify your own processing routines to enhance or replace the default functions.

eClient

The eClient is a browser-based thin client that provides out-of-the-box capabilities for Content Manager systems. eClient uses beans that are build from ICM Java APIs to access services provided by Library Server and Resource Managers (see Figure 2-5). Supported by a “mid-tier” server with direct access to Resource Managers, eClient enables fast and secure delivery of objects while maintaining full transactional support with referential integrity.

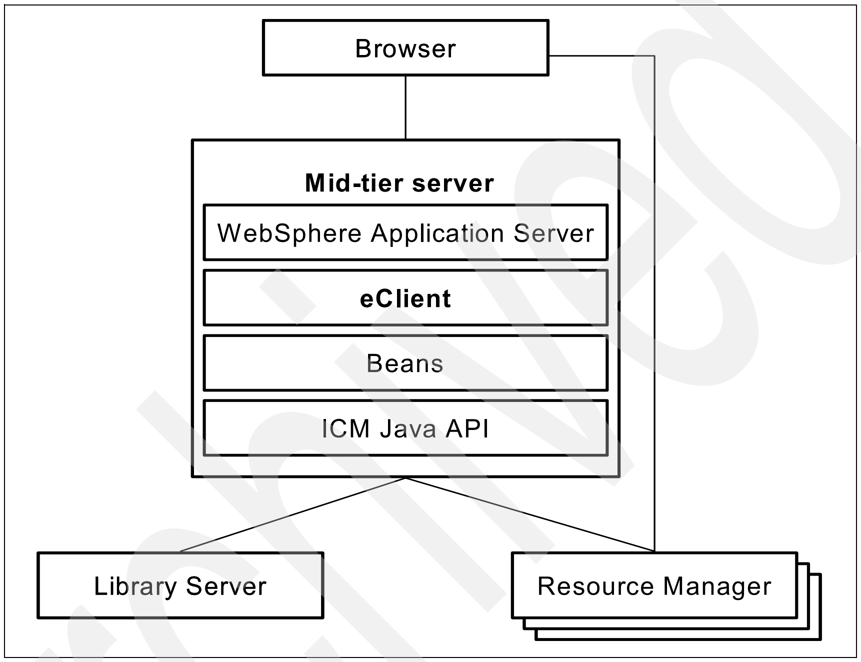


Figure 2-5 eClient architecture

The eClient can be installed on any system that has an Internet Explorer (Version 5.0 or higher) or a Netscape Navigator (Version 4.6 or higher) browser installed. This browser-based client allows users to connect, query, create, update, delete, and display documents and folders.

Customized client

In addition to the out-of-box thin client (eClient) or thick client (Client for Windows), one can also create a customized client to fit the business needs of an enterprise. Customized Content Manager client applications can be created using client APIs, beans, and user exit routines that are part of the ICM connector that comes with Information Integration for Content.

These APIs and beans provide functionality to:

- Access information in Library Server and Resource Managers
- Design data models
- Customize document processing

In this IBM Redbook, we discuss how you can create a Web application using Information Integrator for Content. Using a case study, we develop a sample application to demonstrate the implementation of a customized application.

2.1.2 System configuration

Content Manager architecture is open, portable, and extensible. As mentioned earlier, Content Manager uses triangular architecture, with Library Server and Resource Manager as the foundation, and client applications on top. There are various client options (Client for Windows, eClient, and customized client) available. Depending on your business need, Content Manager system architecture can be build with two-tier or three-tier configuration using different client options, on one or mixed platforms.

Two-tier configuration consists of Client for Windows or customized clients at the top tier, the Library Server, and the Resource Managers at the bottom (see Figure 2-6).

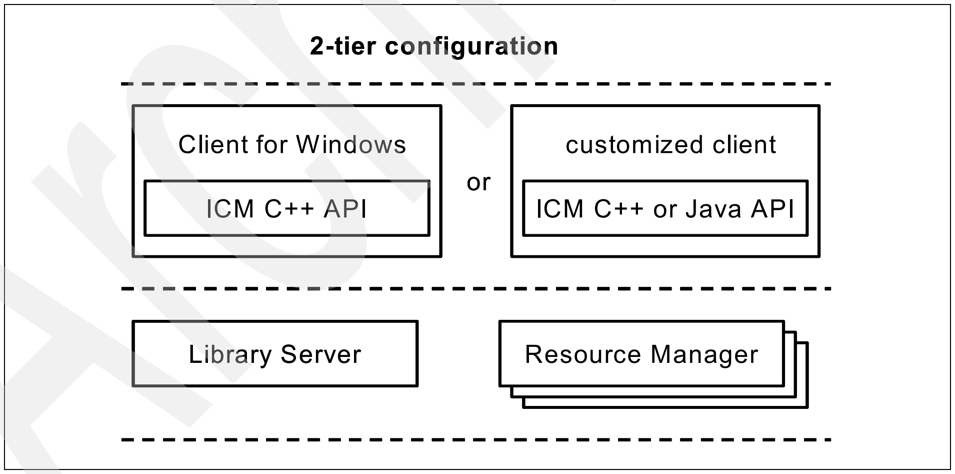


Figure 2-6 Content Manager two-tier configuration

A three-tier configuration consists of a browser at the top tier, the eClient at the mid-tier, and the Library Server and Resource Managers at the bottom tier (see Figure 2-7).

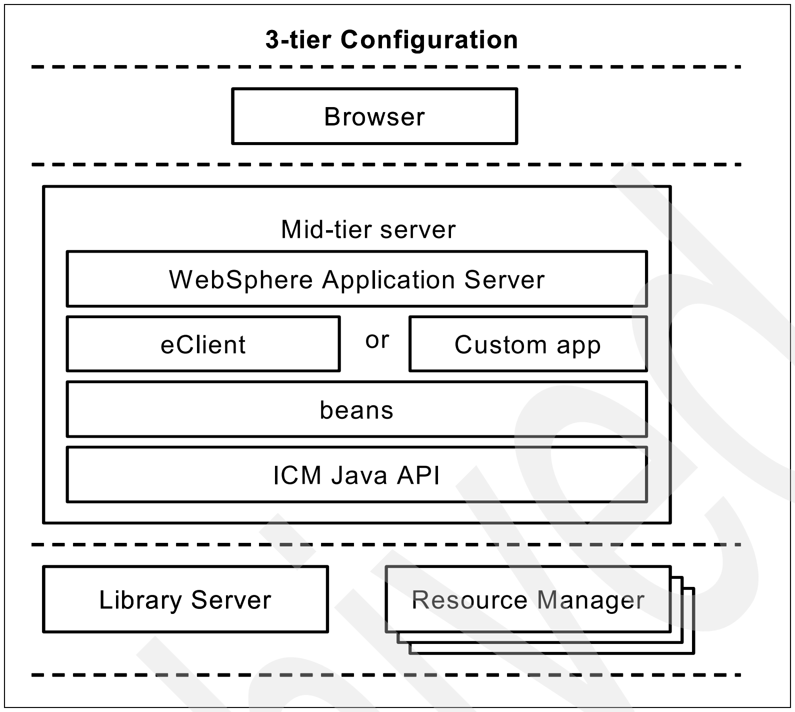


Figure 2-7 Content Manager three-tier configuration

Depending on different business requirements, you can set up different system configurations as follows:

- **All components on one machine:** Install all components (Library Server, Resource Manager, System Administration Client, Client for Windows, and or eClient) on a single machine. This is a good option for a Content Manager prototype system.
- **Separate servers on different machines:** Split the Content Manager's main components to different machines — install the Library Server on one machine, and the Resource Manager on another machine. Install either Client for Windows or eClient on a workstation. System Administration Client can be installed on the same machine as the client.
- **Multiple Resource Managers:** Install multiple Resource Managers with a single Library Server running on the same platform. Access the system using either Client for Windows, or eClient on a workstation.
- **Mixed platforms:** Set up a Content Manager system with the Library Server and one or more Resource Managers running on different platforms. Access the system using either the Client for Windows or the eClient on a workstation.

2.2 Data modeling

To store and manage content, you must first define their metadata using data modeling. Data modeling is the foundation for a Content Manager system. A thorough understanding of Content Manager data modeling is necessary before designing and implementing a Content Manager solution. In this section, we introduce the some of the basic concepts in data modeling for Content Manager.

Attention: Refer to the following IBM Redbook and publication for additional information on data modeling concepts and specific steps in defining a data model through a System Administration Client:

- *Content Manager Implementation and Migration Cookbook*, SG24-7051
- *IBM Content Manager for Multiplatforms System Administration Guide*, SC27-1335

Basic data modeling elements (entities) and concepts include:

- Items and item types
- Attributes and attribute groups
- Root and child components
- Item type classifications
- Versioning
- Links, references, and foreign keys
- ACL

2.2.1 Items and item types

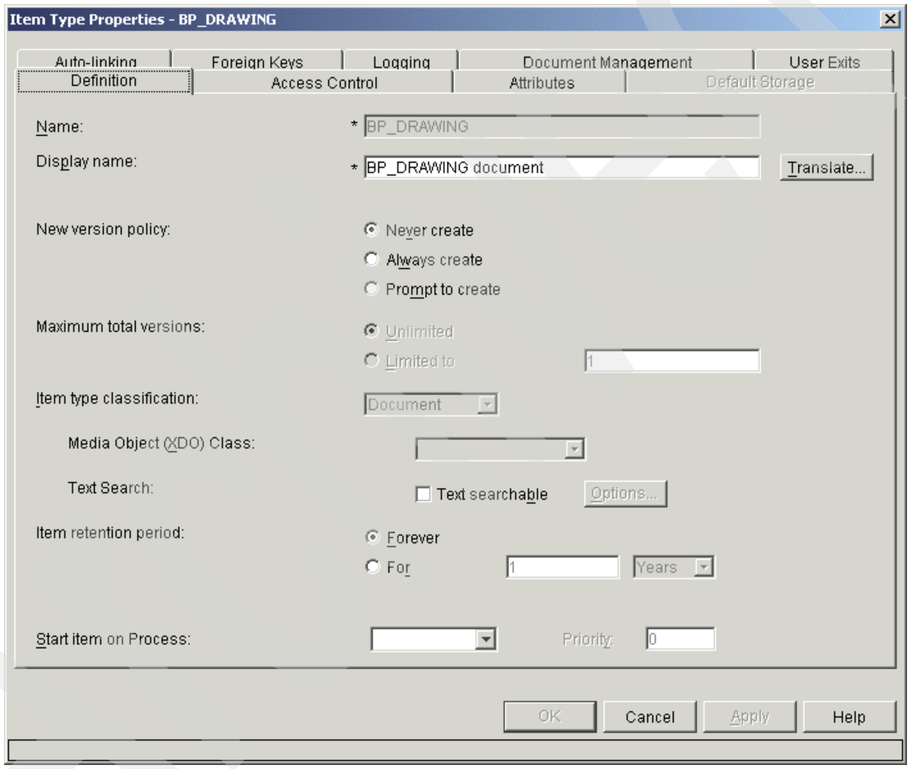
An item is a piece of or a collection of content managed by a Content Manager system. It can be a document, a folder, or a collection of data. It is an instance of an item type. An item type is an item definition. Using a simple analogy, item type is similar to a database table. You must first define an item type (a database table) before creating (instantiating) an item (a record in the database table).

For example, if a property owner wishes to make some modification to his current property, he must submit a proposed property drawing and a plot drawing along with a building permit application to a government agency. In this scenario, we can define item types as shown in Table 2-1.

Table 2-1 Sample item types

Item type	Description
BP_DRAWING	Proposed modified drawing of a property
BP_PLOT	Property plot drawing
BP_APPLICATION	Building permit application

Figure 2-8 shows the screen where you define an item type from a System Administration Client.



Name: * BP_DRAWING

Display name: * BP_DRAWING document

New version policy: (● Never create) Always create, Prompt to create

Maximum total versions: (● Unlimited) Limited to 1

Item type classification: Document

Media Object (XDO) Class:

Text Search: ☐ Text searchable Options...

Item retention period: (● Forever) For 1 Years

Start item on Process: Priority: 0

Figure 2-8 Defining an item type, BP_DRAWING

This is a very simplistic introduction to items and item types. As we cover other data modeling entities and concept, we will explore items and item types in more depth.

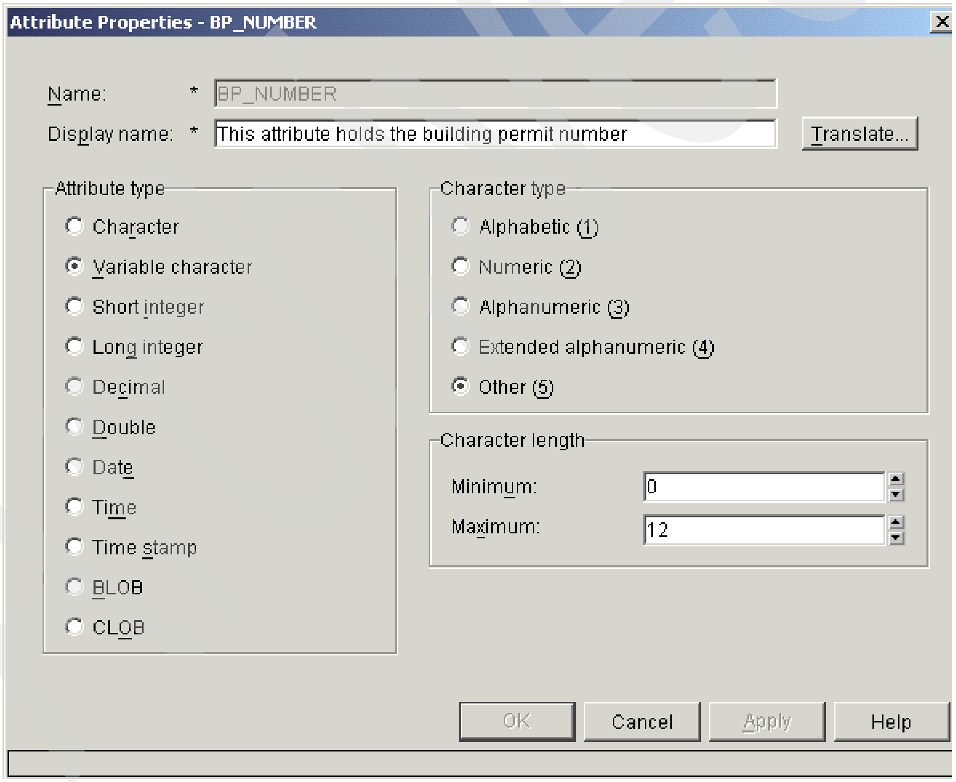
2.2.2 Attributes and attribute groups

An *attribute* describes a characteristic or a property of an item. Table 2-2 lists the attributes that describe a proposed property drawing.

Table 2-2 Sample attributes for a proposed property drawing, BP_DRAWING

Attributes	Description
BP_NUMBER	Building permit number
BP_DOC_DESC	Description for the drawing
BP_SUB_DATE	Submission date

Figure 2-9 shows the specific screen where you define the attribute, BP_NUMBER, from a System Administration Client.


Attribute Properties - BP_NUMBER

Name: * BP_NUMBER

Display name: * This attribute holds the building permit number

Attribute type: Character [✓ Variable character] Short integer Long integer Decimal Double Date Time Time stamp BLOB CLOB

Character type: Alphabetic (1) Numeric (2) Alphanumeric (3) Extended alphanumeric (4) [✓ Other (5)]

Character length

Minimum: 0

Maximum: 12

Figure 2-9 Defining attribute, BP_NUMBER

You can search an item by its attributes. For example, you can search for all drawings submitted prior to a certain submission date.

Attributes can have multiple values and multiple versions. A property owner can have multiple phone numbers, for example: one is a home phone number, one is a cell phone number, and one is a work phone number. As another example, a property drawing can have many versions; one is the original drawing, one is the modified one years later, and one is the most current drawing of the property.

You must create the associated attributes before defining an item type. When creating an item using the item type, Content Manager system assigns a set of system-defined attributes (such as timestamp and item ID) in addition to the user-defined attributes (such as SSN, First name, and Last name). An item ID is used to access all of the data associated with the item.

An *attribute group* is a set of attributes that are grouped together for convenience.

For example, instead of using four attributes (such as street, city, state, and zip code) to define where a property owner lives, you can create an attribute group called address that includes these four attributes to define where an owner lives.

Note that the sample application we show in this redbook does not use attribute groups. This is to simplify the development aspect of the application.

2.2.3 Root and child components

A *component* is the building block used to form the hierarchical tree of data for an item. There are two types of components, root and child. You can build item types by using one root component and zero or more child components.

A *root component* is the first level of an item type. It consists of both the system and user-defined attributes.

For example, we may want to define an item type for a building permit application. The root component includes Item ID and Component ID as system-defined attributes, as well as App ID, Status, Note as user-defined attributes (see Table 2-3).

Table 2-3 Building permit application package - root component

System-defined attributes			**User-defined attributes**			
Item ID	Component ID	...	BP_ NUMBER	BP_ STATUS	BP_ DOC_DESC	...

This root component should contain all attributes (or attribute groups) that apply directly to the item, and for which only one value is expected. If an attribute, or a logical collection of attributes, is expected to have multiple values, a child component should be created.

A *child component* is the optional second (or lower) level of the hierarchical item type. Each child component is directly associated with the level above it. Use child components for attributes (or sets of attributes) where multiple values may exist.

For example, associated with a building permit application, it may contain a number of architectural drawings for the property. The property drawing can be defined as a child component of the building permit application (see Table 2-4).

Table 2-4 Building permit application package - Child component (Drawing)

System-defined attributes				**User-defined attributes**		
Item ID	Parent ID	Component ID	..	BP_DOC_DESC	BP_SUB_DATE	..

The Parent ID of a child component links to its higher (parent) component's Component ID. There are no limits to the number of levels in an item hierarchy. The usage of child components, however, may impact performance. In our sample application, we do not define any child components.

Note that the sample application we show in this redbook does not use the child component. This is omitted to simplify the development aspect of the application.

2.2.4 Item type classifications

An item type consists of a root component, zero or more child components, and a classification. There are four *item type classifications*:

Non-resource	Represents entities that are not stored in a Resource Manager; metadata in Library Server only.
Resource	Represents objects stored in a Resource Manager. Examples: image files, video clips, LOBs (large objects) in database tables, and text.
Document	Represents entities that contain document parts.
Document part	Similar to Resource item type; represents objects stored in a Resource Manager; however, it is part of a document, contained and owned by a document item type.

Depending on the item type classification, you can create resource items, document parts, documents, and folders (special types of documents). In the following section, we briefly discuss each entity, starting with an introduction to objects and MIME types.

Objects and MIME types

An *object* (also known as resource content) is any digital content that a user can store, retrieve, and manipulate as a single unit. For example, objects can be JPEG images, MP3 audio, AVI video, and a text block from a book. An object is always stored in a Resource Manager. Access to an object is controlled by the Library Server.

MIME type is an Internet standard for identifying the type of object that is being transferred across the Internet. MIME types include many variants of text, audio, image, and video data.

When you create an object in Content Manager, you specify its MIME type. When an object of that type is retrieved from the Resource Manager, Content Manager reads the MIME type and determines how to handle the object. For example, if the MIME type for an object is GIF, the Content Manager application may launch a Web browser when displaying the object.

Resource items

A resource item is created from an item type that is classified as “Resource.” It represents the object that is stored in the Resource Manager.

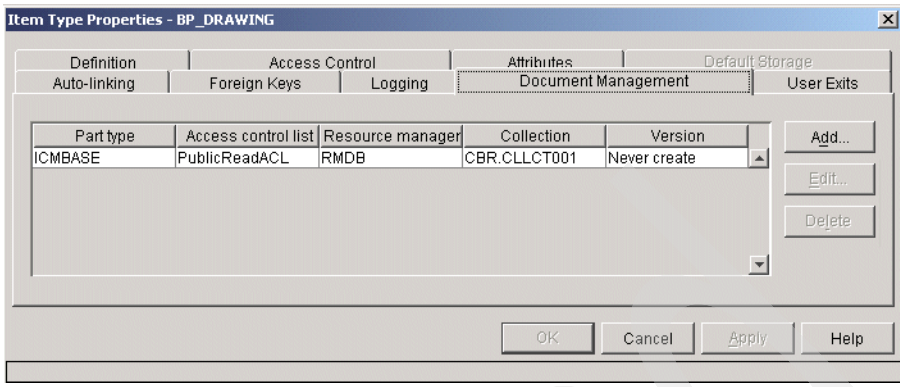
Documents and document parts

A document is created from an item type that is classified as “document” (also known as the “document model”). A document may contain document parts. The document parts can include various types of content, including text, images, and spreadsheets.

There are several pre-defined document parts:

ICMBase	Fundamental document part. Used to store documents, images, photos, or any object stored in the system.
ICMBaseText	Used for storing textual documents or files that are intended to be indexed for full text searches.
ICMBaseStream	Used for storing videos that can be used with Content Manager VideoCharger™.
ICMNoteLog	Used to store notes log information for an item.
ICMAnnotations	Used to hold the markups (sticky notes, color highlights, stamps, or other graphical highlights) added to objects by users.

See Figure 2-10 for an ICMBase document part example in an item type definition.



Part type	Access control list	Resource manager	Collection	Version
ICMBASE	PublicReadACL	RMDB	CBR.CLLCT001	Never create

Figure 2-10 Define document part for an item type

In general, using a resource item — thus, a resource item model — is more efficient. The document model requires intermediate database tables to hold the links between the document and the parts of which it is comprised. The document model, however, with its pre-defined set of document parts and associated supporting API classes, makes programming much easier. We recommend the use of the document model. In our sample applications, we use only the document model.

Folders

A folder is an item that may contain a list of other items. For example, a folder can contain documents, resource items, or other folders.

2.2.5 Versioning

Versioning enables storing and maintaining of multiple versions of an item, including versions of the item's child components. You can set a maximum number of versions or allow an unlimited number. If you set a maximum number, and the maximum number is reached, Content Manager automatically deletes the oldest version when saving a new version.

2.2.6 Links, references, and foreign keys

Depending on your requirements, relationships between items in Content Manager can be established in three ways: via links, references, or foreign keys.

Links

A *link* is a bi-directional association between one root component to another root component. The *link* relationships are between selected common attributes defined at the root level of each item type. Attributes defined at the child levels (components) in the item type cannot be used for link relationships. Using link relationships avoids duplicating the resources in the linked item type when needed. Links are directional. There is a source item and a target item.

For example, suppose you have property owners and properties. An owner may own multiple properties. Instead of making the property a child component of the owner, you can specify both property owner and property as two item types, linked to each other.

Another example of linking is the *auto-linking* support in Content Manager. An item type is created for users to import images. This item type has base parts so the imported objects can be stored. A second item type is created to represent the folder, but without document part definitions. A single common attribute between the item types is used to set up the auto link. When users import objects into the first item type, users who have access to the second item type representing the folder can search the system and locate objects imported with the first item type.

References

A *reference* is a single direction one-to-one association between the root or child component of an item to the root component of another item.

For example, you can create a reference between the property owner item type to the property drawing item type, instead of using linking.

Once created, the reference can be included in the attribute listing of the source item type, similar to any other attribute. Note that links are more dynamic. They can be added during run-time.

Foreign keys

A *foreign key* is a column or a set of columns in a table that refers to a unique key or the primary key of a different table. A unique key ensures that no rows in that table are duplicated. Each table can have only one primary key, but it may have more than one unique key.

You use a foreign key to enforce referential integrity among tables. In Content Manager, you can define foreign keys to another item type or to a database table that is not part of the Content Manager system.

For example, a legacy system contains the records of all property owners in a particular county, and you need to create a folder that will contain all the building permit application related information in a Content Manager system. Assuming that the applicant must be the property owner in the county, then, whenever you create a new folder, the tax ID of the building permit applicant must match with the social security number of a property owner in the legacy system.

2.3 Document Routing

There are several workflow options available for Content Manager: Document Routing, Advanced Workflow, and MQSeries® Workflow.

In this redbook, we focus only on the Document Routing option. The other options are beyond the scope of this redbook.

2.3.1 Key concepts

A common work process in a business environment involves gathering the necessary paperwork and information, moving all of this from one person (department) to another, getting the appropriate approval, going through some special processing, and completing the process. The Document Routing feature of Content Manager enables you to go through this workflow process, paperless.

You can set up, configure, and program the Document Routing feature to move documents or folders within a pre-defined process.

There are several basic concepts in Document Routing:

- Work package
- Work node (work basket and collection point)
- Document Routing
- worklist

A *work package* refers to a set of documents or folders that move along the workflow process. The documents or folders can be added or deleted along the way. The work package contains the information (such as priority, state, resume time, and Item ID) that users need to complete tasks within the process.

A *work node* is a place (step) within a workflow process at which the documents or folders wait until users perform some actions on them (such as reject or approve). A work node can be a simple node, known as a *work basket,* or a special node, known as a *collection point.* The collection point (a special work node) is a place where a user or the system waits until a pre-defined set of documents or folders are gathered. After all documents or folders are in the system, the user or the system can perform actions on the documents or folders to continue the workflow process.

The routing of the documents or folders within the work nodes is known as *Document Routing*. At any point, you can define how users see the items (documents or folders) within the Document Routing process. The view you defined is known as a *worklist*. A worklist can contain the items from multiple work nodes or just the items from one specific node; this depends on the business requirements.

In the remainder of this section we describe the workflow process, and discuss the other key concepts in more detail.

2.3.2 Process

A process is a series of steps through which an item (document or folder) is routed. A process contains at least one start work node, one action, and one end work node. You can use one-step processes to create ad hoc processes. Processes can have as many steps and as many work nodes as you want.

There are a variety of processes:

- **Serial processes:** Takes work from start to finish without any deviations.
- **Dynamic process:** Enables the direction work through different routes depending on the action performed by a user or the system.

Content Manager provides two default actions: *Continue* and *Escalate*. Depending on which action the user chooses, you can direct items (documents or folders) to a different work node. Note that these default actions names are labels only. You can change them or add additional actions such as reject or approve.

For example, if you want a building permit application to go from one node to another, you can select Continue as the path it takes. At some point in the process, a reviewer must either approve or reject the application. This means you must create two additional work nodes: one is where the application will be routed to if it is being approved, one is where it will be routed to if it is being rejected. Figure 2-11 demonstrates this concept.

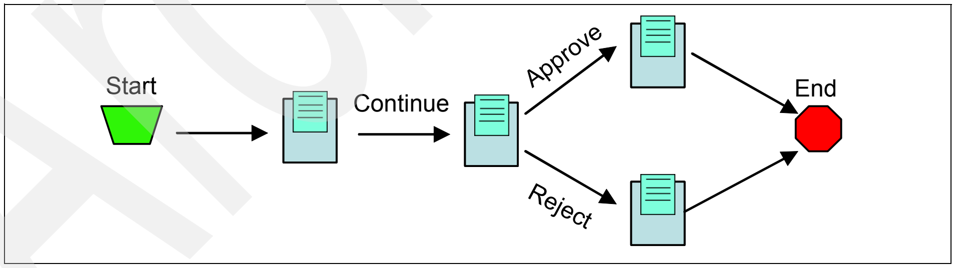
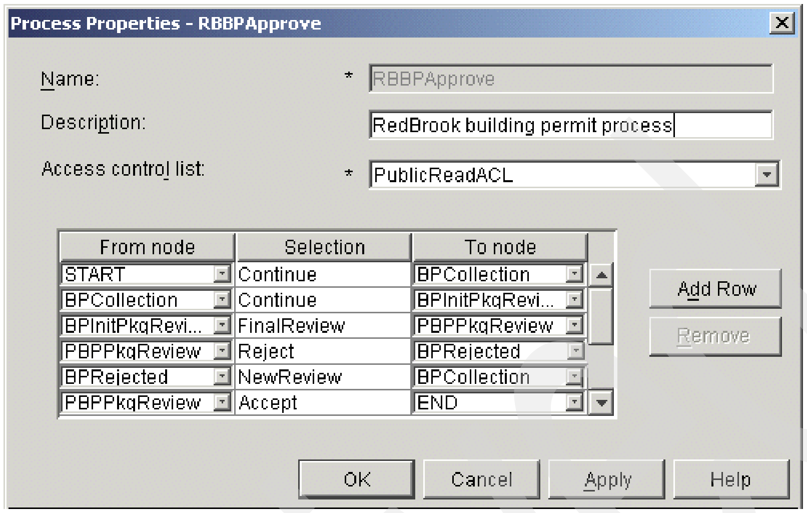


Figure 2-11 Simple workflow branching with for building permit application

Figure 2-12 shows the actual process we define for our building permit example.


Process Properties - RBBPApprove

Name: * RBBPApprove

Description: RedBrook building permit process

Access control list: * PublicReadACL

From node	Selection	To node
START	Continue	BPCollection
BPCollection	Continue	BPInitPkgRevi...
BPInitPkgRevi...	FinalReview	PBPPkgReview
PBPPkgReview	Reject	BPRejected
BPRejected	NewReview	BPCollection
PBPPkgReview	Accept	END

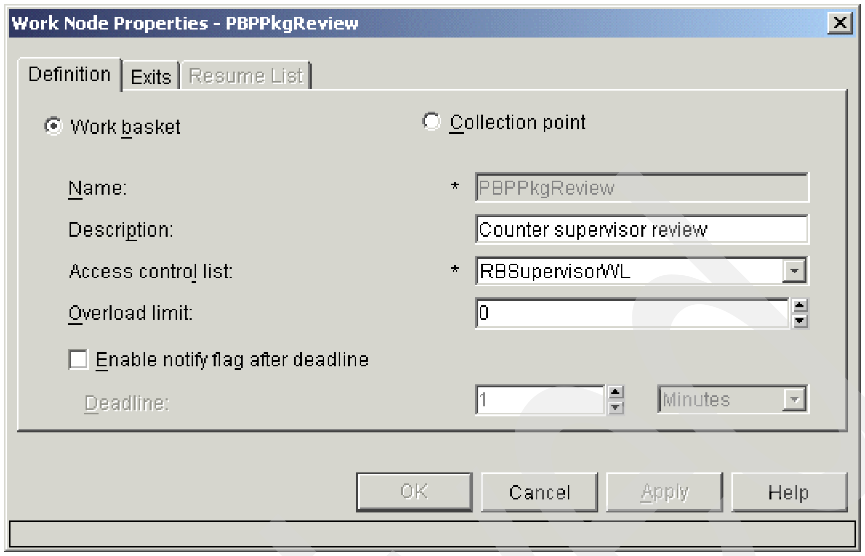
Figure 2-12 Define process in Document Routing

We will discuss more about the building permit example later.

2.3.3 Work nodes

Each step in a process is referred to as a *work node*. As mentioned earlier, a work node can be either a work basket or a collection point.

Figure 2-13 shows the specific screen where you define a work node from a System Administration Client.


Work Node Properties - PBPPkgReview

Definition | Exits | Resume List

[✓ Work basket] Collection point

Name: *	PBPPkgReview
Description:	Counter supervisor review
Access control list: *	RBSupervisorWL
Overload limit:	0
Enable notify flag after deadline	☐
Deadline:	1 Minutes

Figure 2-13 Defining work node

Work baskets

Each step in a process corresponds to a real-world task, such as verifying a record or rejecting an application. *Work baskets* contain work packages. A *work package* contains the location of documents or folders in a database and its priority. A work basket does not perform any actions on the content; rather, it is an indicator of where a work package is in a process. You can control which users can perform specific actions on the work packages.

A work basket is more than just a virtual basket that has a pile of work stacked in it. You decide what functions a work basket requires to get a work package to where it needs to go. You can specify, through dynamic link libraries (DLLs), what tasks work packages complete upon entering and leaving a work basket. You can also specify what a work package must do when a work basket cannot contain it by using a DLL on the condition that the work basket is overloaded. Your DLLs must reside on the same machine as the Library Server.

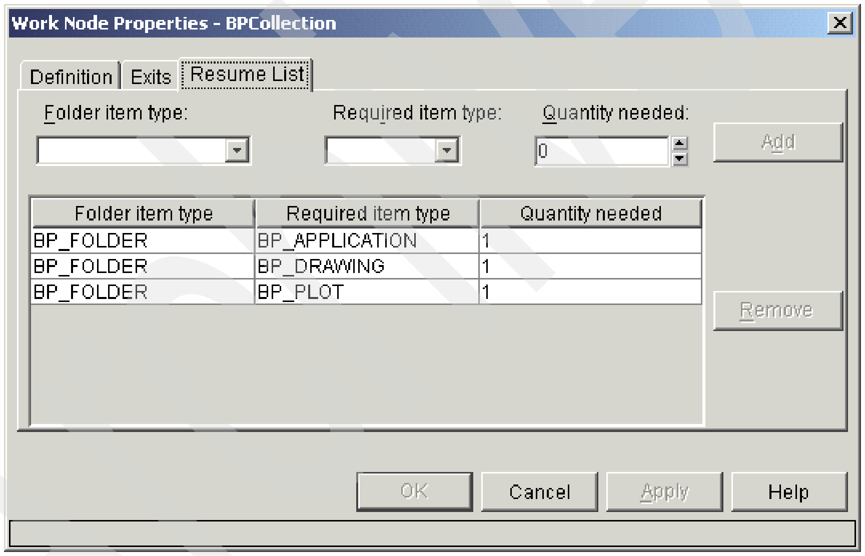
Collection points

A *collection point* is a special work node that waits for external documents to be collected in a folder. It collects required documents and sends them to another work node when it either completes a folder, or the time allotted to wait for the documents has expired.

A collection point is strictly used in document routing processes. It is not a Resource Manager collection.

Note: When using collection points, you need to start a *folder* on the process. When the folder reaches the collection point, it remains in that collection point until the defined number of items (of the defined item type) are added to this folder before moving to the next step of the process.

Using the building permit application example mentioned in the previous section, you can add a collection point (BPCollection) in the beginning to ensure that all the necessary documents, such as the building permit application (BP_APPLICATION), the property drawing (BP_DRAWING), and the plot drawing (BP_PLOT), have been received before a clerk forwards them to another node for approve or reject. Figure 2-14 shows the specific screen where you define the collection work node from a System Administration Client.



Folder item type:

Required item type:

Quantity needed: 0

Folder item type	Required item type	Quantity needed
BP_FOLDER	BP_APPLICATION	1
BP_FOLDER	BP_DRAWING	1
BP_FOLDER	BP_PLOT	1

Figure 2-14 Defining a collection work node

Ad hoc routing processes

Ad hoc routing processes allow you to remove a document or folder from one process and put it in another.

2.3.4 Work packages

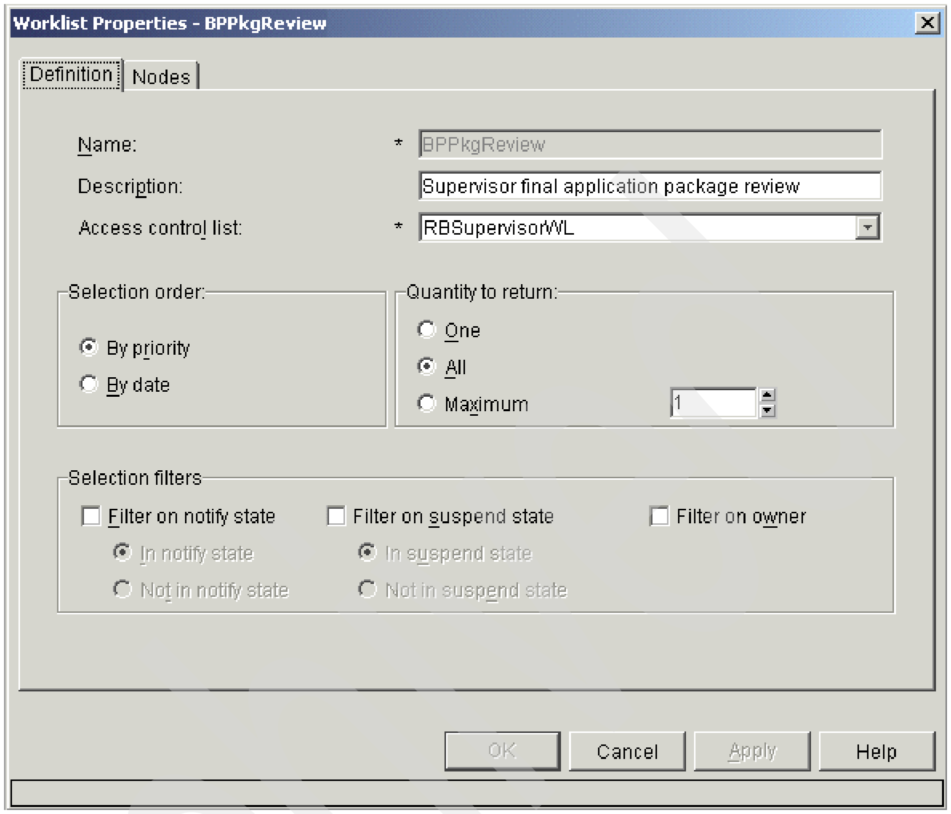
A *work package* is a set of information that includes priority, state, resume time, and Item ID of the item being routed. This item can be a folder or an item. It is used to relate an item to a work node. You do *not* create work packages. Work packages are created by the system with the information from the user who starts a process. The user logs on to Content Manager and proceeds to start a process. Content Manager prompts the user to specify the process, the item ID that uses this process, and the item priority. Content Manager takes this information and creates a work package that proceeds through the process.

2.3.5 Worklists

A *worklist* controls the selection and presentation of work to a user. They are used to display work packages that are in one or more work nodes. A worklist spans all work nodes that it is defined to cover, regardless of the processes that the work packages at these nodes are in. For example, if a worklist has been defined to include all work packages for a particular work node, then all work packages are displayed to users regardless of the process they are in.

You need to assign work nodes to a worklist and give the worklist an access control list (ACL). The ACL filters out the users that can access the work nodes. The ACL of the work nodes further restricts access to the work packages in them.

For example, using the building permit example, we create a worklist, BPPkgReview, where only supervisor can work with this worklist. Figure 2-15 shows the screen that defines this specific worklist.


Worklist Properties - BPPkgReview

Definition | Nodes

Name: * BPPkgReview

Description: Supervisor final application package review

Access control list: * RBSupervisorWL

Selection order:
- [✓] By priority
- By date

Quantity to return:
- One
- [✓] All
- Maximum 1

Selection filters
- [ ] Filter on notify state
  - [✓] In notify state
  - Not in notify state
- [ ] Filter on suspend state
  - [✓] In suspend state
  - Not in suspend state
- [ ] Filter on owner

Figure 2-15 Define worklist

In implementing the sample application for this redbook, we define a simple data model and document routing for the application. See the sample application setup to get a better understanding of the concepts presented here.

2.4 Security and privilege sets

The Content Manager Version 8 access control model is applied at the level of the controlled entity. A *controlled entity* is a unit of protected user data. In different Content Manager installations, the controlled entity can be an individual item, item-type, or the entire library. Operations on the controlled entities are regulated by one or more control rules.

2.4.1 Access control list (ACL)

The *access control list (ACL)* is the container for the control rules. Every controlled entity in a Content Manager system must be bound to an ACL. Default settings can be configured by the system administrator, if appropriate. An ACL is associated with an item type to enforce access control at the item type level. Similarly, an item level access control is established by binding an ACL to the desired item when the item is created. When a user initiates an operation on the item, the system will check the user's privilege and the ACL associated with this item to determine if this user has the right to perform such an operation on this item. The right to access an item also requires the right to access the item type based on which this item type is defined.

2.4.2 Privilege and privilege set

A *privilege* is an ability to use the Content Manager system. Content Manager privileges are used to grant to individual users and to define access control lists (ACLs). Content Manager privileges must be grouped into privilege sets before they can be used. Content Manager provides a number of un-modifiable pre-defined privileges, called *System-defined Privileges*. Each of these privileges authorizes a certain operation(s). System-defined Privileges are enforced by the Content Manager Library Server stored procedures.

Content Manager also allows users to define their application-specific privileges, called *User-defined Privileges*. Each Privilege has a system-generated unique code, called the *Privilege Definition Code*. Privilege Definition Code 0 to 999 are reserved to store Content Manager System-defined Privileges. Code 1000 and up are open for user-defined privileges User-defined privileges contains the rights to access and modify the Content Manager controlled entities. Access to the controlled entities, in addition to ACL checking, is controlled by this category of privileges. Some examples of user-defined privileges include the privilege to:

- Query item types and attributes.
- Query items.
- Create items.
- Move items between different item types.
- Delete items.

A Content Manager *privilege set* is a named group of privileges. The purpose of using privilege sets is to ease system administration. Content Manager privileges must be grouped into privilege sets before they can be granted to users or used to define ACLs. Privilege sets granted to users are called user privileges. They define the individual users' individual bounds on using the system. When applied to ACL specifications, the privilege set specify the limits of the operations allowed on the bound Controlled Entity. Only *individual users* can be associated with privilege sets. User groups cannot hold privilege sets for their members.

2.5 Information Integrator for Content

IBM DB2 Information Integrator for Content Version 8 (Information Integrator for Content), formerly known as Enterprise Information Portal, provides a single point of access to unstructured and structured content stored on one or more content servers.

Information Integrator for Content comes with an administration database and an administration client. It is a framework that consists of two parts:

- Information access
- Services

In the following section, we briefly describe these two parts.

2.5.1 Information access

A *content server* is any system that stores multimedia, business forms, documents, and related data, along with metadata that allows users to process and work with the content. Within an organization, there may be disparate content servers across different business function units and over different geography. To access the information stored in these content servers, an application can either use one of the content server connectors or a federated connector, as illustrated Figure 2-16.

Content server connectors provide the communication interface among the applications, the content servers, and the administration database. A connector can be implemented for arbitrary content servers. Information Integrator for Content Version 8 provides connector implementations for content servers such as DB2 or Content Manager Version 8.

The *federated connector* has the same interface as all other connectors but it does not have a physical store and it is configured with any number of supported connectors to become the single point of access for multiple content servers. You can use it to search, retrieve, and update data objects in the content servers; however, you need to call the content server connector directly to create and delete data objects.

To allow federated access, a *schema mapping* between the federated content server and each participating content server is required. The schema mapping handles the difference between how the data is physically stored and how the user wants to process the data in an application. This applies to attributes as well as user IDs and passwords. In an application, the persistent data objects are represented by the Dynamic Data Objects (DDO). A DDO is a server-neutral and self-describing data object for transferring data into and out of a content server. A DDO has a single persistent ID (PID), an object type, and a set of data items.

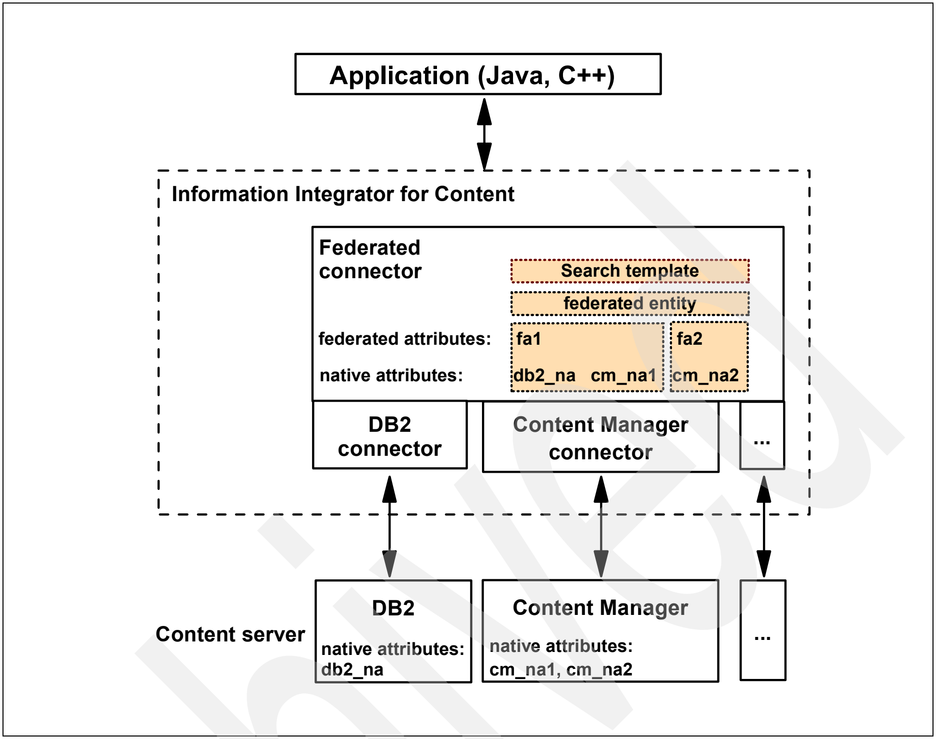


Figure 2-16 Information access overview

Figure 2-16 illustrates the information access using connectors and the mapping of native attributes to the federated attributes. Federated attributes can be grouped to a federated entity and then be used to define a search template.

Defining search templates is a convenient way to predefine queries and control the access to those queries. To search, you simply retrieve a search template from the administration database, input the search values for the federated attributes, and perform a search. If the template contains federated attributes that are mapped to the native attributes of different content servers, the search runs simultaneously across these servers.

The programming interface is available for C++ and Java. Java applications can also use the client/server implementation, which is based on Remote Method Invocation (RMI).

2.5.2 Services

Information Integrator for Content services provide added values for information access and programming interfaces that are aligned with the information access interfaces.

Currently Information Integrator for Content provides two services:

- **Information Mining Service:** This service allows you to automatically analyze and organize documents on content servers. Because nearly 80 percent of your business data is unstructured, you cannot do this manually. Information Mining provides tools such as automatic categorization, summarization, and information extraction. If the analysis results get stored together with the original document, you can, for example, restrict searches to certain categories and display a summary for each search result.
- **Workflow:** You can use the workflow service to control the flow and performance of work in your business. When users work with the results of federated searches, they often must make decisions on what actions to perform. You can determine in advance how you want users to perform the work. The actual documents can reside on any of the supported content servers. You can automate the workflow by setting up profiles and rules.

2.5.3 Administration database and client

Information Integrator for Content comes with an *administration database*. This is a DB2 UDB database. It stores all of the information, including schema mappings, that are needed to manage Information Integrator for Content and its components.

Information Integrator for Content also comes with an *administration client*. The system administrator uses this client to do the following operations:

- Define each content server for federated searching.
- Identify native entities and attributes on content servers and map them to federated entities.
- Create search templates.
- Identify and manage users who can access search templates, the information mining feature, and workflow processes.
- Define business workflow processes.

The client interface is similar to that of the Content Manager System Administration Client (see Figure 2-17). You also can access Information Integrator for Content from the Content Manager System Administration Client.

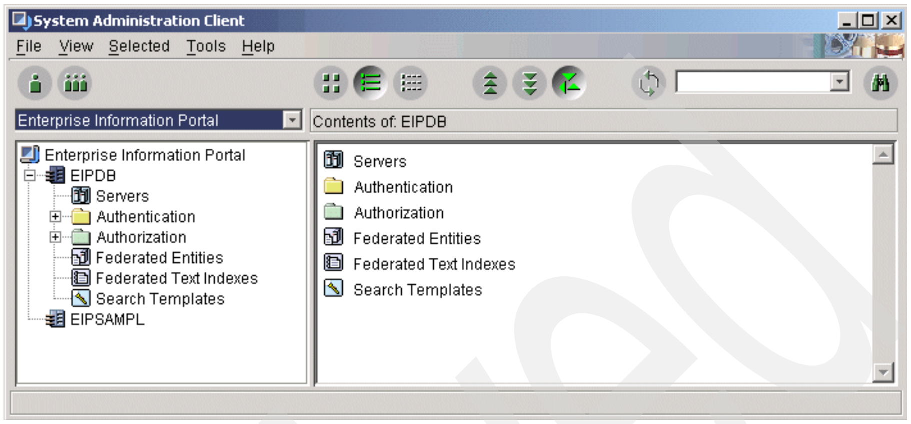


Figure 2-17 Information Integrator for Content system administration client

In this figure, the information is stored in the EIPDB, which is the Information Integrator for Content administration database.

3

OnDemand overview

In this chapter, we introduce Content Manager OnDemand.

We cover the following topics:

- Overview
- Basic concepts
- Servers and server programs

3.1 Overview

IBM DB2 Content Manager OnDemand supports any organization that can benefit from hard copy or microfiche replacement and instant access to information. An OnDemand system can support small office environments and large enterprise installations with hundreds of system users. It can dramatically improve productivity and customer service in many businesses by providing fast access to information stored in the system.

OnDemand processes the print output of application programs, extracts index fields from the data, stores the index information in a relational database, and stores one or more copies of the data in the system. With OnDemand, you can archive newly created and frequently accessed reports on high speed, disk storage volumes, and automatically migrate them to other types of storage volumes as they age.

OnDemand fully integrates the capabilities of Advanced Function Presentation™ (AFP™), including management of resources, indexes, and annotations, and supports full fidelity reprinting and faxing of documents to devices attached to a PC, OnDemand server, or other server on the network.

OnDemand provides administrators with tools to manage OnDemand servers, authorize users to access OnDemand servers and data stored in the system, and back up the database and data storage. OnDemand also provides users the ability to view documents, print, send, and fax copies of documents, and attach electronic notes to documents.

OnDemand offers several advantages; it can:

- Easily locate data without specifying the exact report.
- Retrieve the pages of the report that you need without processing the entire report.
- View selected data from within a report.

OnDemand can provide you with an information management tool that can increase your effectiveness when working with customers.

OnDemand has the following capabilities:

- Integrates data created by application programs into an online, electronic information archive and retrieval system.
- Provides controlled and reliable access to all reports of an organization.
- Retrieves data that you need when you need it.
- Provides a standard, intuitive client with features such as thumbnails, bookmarks, notes, and shortcuts.

These features mean that OnDemand can help you quickly retrieve the specific page of a report that you need to provide fast customer service.

An OnDemand system consists of OnDemand clients and OnDemand Library Server and OnDemand Object Server(s). See Figure 3-1 for an OnDemand system architecture.

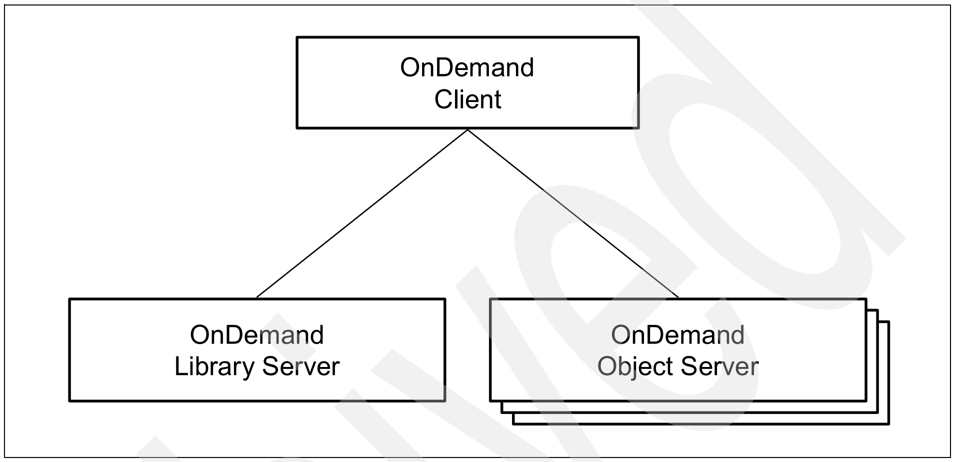


Figure 3-1 OnDemand system architecture

OnDemand Library Server manages a database of information about the users of the system and the reports stored on the system. It maintains index data and server control information. *OnDemand Object Server* manages the reports on disk, optical, and tape storage devices.

An OnDemand system has one Library Server and one or more Object Servers. An Object Server can operate on the same machine or node as the Library Server or on a different machine or node than the Library Server.

OnDemand clients run on personal computers or terminals attached to the network and communicate with OnDemand servers. OnDemand client programs run Windows. Using the client programs, user can search for and retrieve reports, documents, view, print, and FAX copies or pages of documents, and attach electronic notes to pages of a document.

OnDemand servers manage control information and index data, store and retrieve documents and resource group files, and process query requests from OnDemand client programs.The documents can reside on disk, optical, and tape storage volumes. New reports can be loaded into OnDemand every day, so that OnDemand can retrieve the latest information generated by application programs.

When a user submits a query, the client program sends a search request to the OnDemand Library Server. The Library Server returns the list of documents that match the query to the user. When the user selects a document for viewing, the client program retrieves a copy of the document from the Object Server where the document is stored, opens a viewing window, and displays the document.

3.2 Basic concepts

To store, manage, retrieve, view, and print reports and index data in OnDemand, you must design and implement applications, application groups, and folders. In this section, we introduce some of the basic concepts:

- Application
- Application group
- Folder
- Indexing method and documents

3.2.1 Applications

An *application* describes the physical characteristics of a report to OnDemand. Typically, you define an application for each program that produces output to be stored in OnDemand. The application includes information about the format of the data, the orientation of data on the page, the paper size, the record length, and the code page of the data. The application also includes parameters that the indexing program uses to locate and extract index data and processing instructions that OnDemand uses to load index data in the database and documents on storage volumes.

3.2.2 Application groups

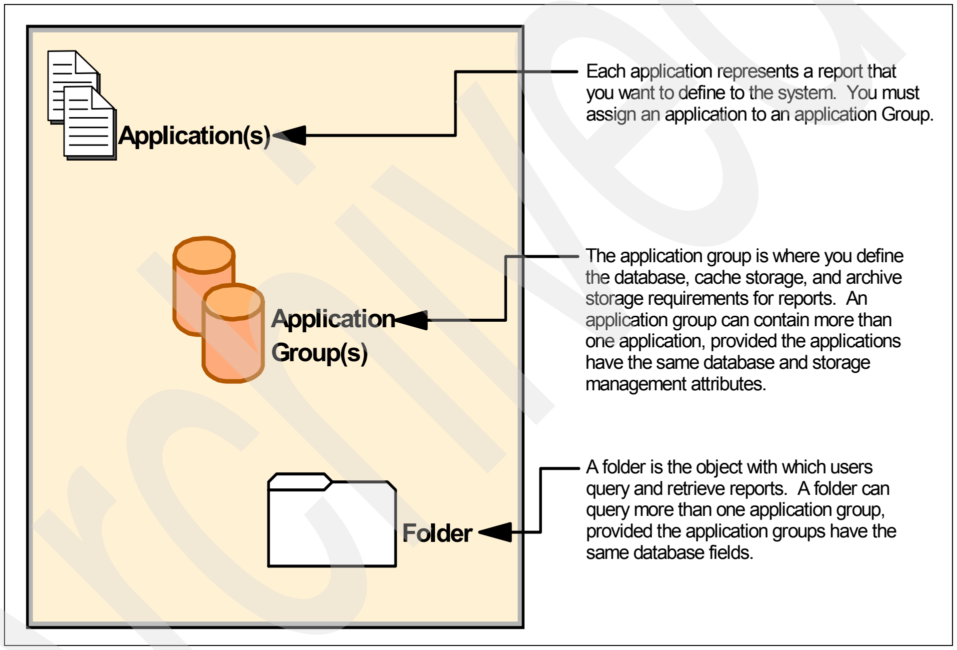
An *application group* is a collection of one or more applications which contains common indexing and storage management requirements. The application group group contains the database information which is used to load, search for, and retrieve reports. The application group defines the data which to be loaded into the database.

You can group several different reports in an application group so that users can access the information contained in the reports with a single query. All of the applications in the application group must be indexed on at least one of the same fields, for example, customer name, account number, and date.

3.2.3 Folder

A *folder* provides users with a convenient way to find related information stored in OnDemand, regardless of the source of the information or how the data was prepared. A folder allows an administrator to set up a common query screen for several application groups that may use different indexing schemes, so that a user can retrieve the data with a single query. For example, a folder called Student Information might contain transcripts, bills, and grades, which represents information stored in different application groups, defined in different applications, and created by different programs.

Figure 3-2 illustrates the concepts described in this section.



Application(s)
Application Group(s)
Folder
Each application represents a report that you want to define to the system. You must assign an application to an application Group.
The application group is where you define the database, cache storage, and archive storage requirements for reports. An application group can contain more than one application, provided the applications have the same database and storage management attributes.
A folder is the object with which users query and retrieve reports. A folder can query more than one application group, provided the application groups have the same database fields.

Figure 3-2 Folders, application groups and applications

Each report has indexed fields and data type.

3.2.4 Indexing methods and documents

In this section, we discuss indexing methods and OnDemand documents.

Indexing methods

OnDemand provides two basic ways to index data:

- *Document indexing* is used for reports that contain logical items such as policies, and statements. Each of the items in a report can be individually indexed on values such as account number, customer name, and balance. OnDemand supports up to 32 index values per item. With document indexing, the user does not necessarily need to know about reports or report cycles to retrieve a document from OnDemand.
- *Report indexing* is used for reports that contain many pages of the same kind of data, such as a transaction log. Each line in the report usually identifies a specific transaction, and it would not be cost effective to index each line. OnDemand stores the report as groups of pages and indexes each group. When reports include a sorted transaction value (for example, invoice number), OnDemand can index the data on the transaction value. This is done by extracting the beginning and ending transaction values for each group of pages and storing the values in the database.This type of indexing lets users retrieve a specific transaction value directly.

Documents

OnDemand documents represent indexed groups of pages. Typically an OnDemand document is a logical section of a larger report, such as an individual customer statement within a report of thousands of statements. An OnDemand document can also represent a portion of a larger report. For reports that do not contain logical groups of pages, such as transaction logs, OnDemand can divide the report into groups of pages. The groups of pages are individually indexed and can be retrieved to the client workstation much more efficiently than the entire report. Documents are always identified by date, and usually one or more other ways, such as customer name, customer number, or transaction number.

For example, a county agency produces billing report and transaction report on monthly basis. The billing report contains the property tax billing statement for every property owners in the county. The transaction report lists all the payment transactions by the property owners. To capture and retain the output of these two reports, you can produce an application (BILLS) for the billing report and an application (TRANS) for the transaction report. The BILLS application uses the *document indexing* method to divide the report into documents. Each property owner's billing statement in the report becomes a document in OnDemand.

Users can retrieve a billing statement by specifying the date and any combination of property owner or property number. The TRANS application uses the *report indexing method* to divide the report into documents. Each group of 100 pages in the report becomes a document in OnDemand. Each group is indexed using the first and last sorted transaction values that occur in the group. Users can retrieve the group of pages that contains a specific transaction number by specifying the date and the transaction number. OnDemand retrieves the group that contains the value entered by the user.

3.3 Servers and server programs

The OnDemand server environment includes a Library Server and one or more Object Servers residing on one or more workstations connected to a TCP/IP network. The Library Server maintains a central database about the reports stored in OnDemand. The database also contains information about the objects defined to the system, such as users, groups, printers, application groups, applications, folders, and storage sets. The database manager provides the database engine and utilities to administer the database. The Library Server processes client logons, queries, and print requests and updates to the database. The major functions that run on the Library Server are the request manager, the database manager, and the server print manager.

An Object Server maintains documents on cache storage volumes and, optionally, works with an archive storage manager to maintain documents in archive storage, such as optical and tape storage libraries.An Object Server loads data, retrieves documents, and expires documents. The major functions that run on an Object Server are the cache storage manager, OnDemand data loading and maintenance programs, and optionally, the archive storage manager.

3.3.1 System configuration

The basic OnDemand configuration is a Library Server and an Object Server on the same workstation or node. This single Library Server/Object Server configuration supports the database functions and cache storage on one workstation. You can add an archive storage manager to the single Library Server/Object Server configuration, to maintain documents in archive storage. You can also configure your OnDemand system with the Library Server on one workstation and one or more Object Servers on different workstations. This configuration is known as a distributed Library Server/Object Server system. The distributed Library Server/Object Server configuration supports caching of documents on different servers. You can add an archive storage manager to one or more of the Object Servers to maintain documents in archive storage that is attached to different servers.

The OnDemand server environment contains several components:

- A *request manager* that provides client, network, and operating system services, security, and accounting.The request manager resides on the Library Server.
- A *database manager* that maintains the index data for the reports that you store on the system.The database manager is a relational database management product, such asDB2 (included with your product package). The database manager resides on the Library Server.

- Database *control information* about the users, groups, application groups, applications, folders, storage sets, and printers that you define on the system. The control information determines who can access the system, the folders that a user can open, and the application group data that a user can query and retrieve. The database resides on the Library Server.
- A cache *storage manager* that maintains documents in cache storage. Cache storage is for high-speed access to the most frequently used documents.
- An *archive storage manager*, which is an optional part of the system. The archive storage manager is for the long-term storage of one or more copies of documents in archive storage, such as optical and tape storage libraries. Tivoli Storage Manager (included in your product package) is an example of an archive storage manager product. You can also use Tivoli Storage Manager to backup and restore DB2 databases. This capability means that you do not have to manage DB2 backup files on disk.
- A *download facility* that automatically transfers spool files to a server at high speed. IBM recommends that you use Download for OS/390®, a licensed feature of Print Services Facility™ (PSF) for OS/390. Download provides the automatic, high-speed download of JES spool files from an OS/390 system to OnDemand servers.
- *Data indexing and conversion* programs. These programs extract index data from input files or generate index data and, depending on the indexer, optionally collect resources and transform input data from one format to another. OnDemand provides several indexing programs:
  - The AFP Conversion and Indexing Facility (ACIF) can be used to index line data, ASCII data, and AFP input files. ACIF can collect the resources that are required to view AFP documents and convert line data input into AFP data to be stored on the system.
  - The Generic indexer can be used to create index data for almost any type of data that you want to store on the system, such as Hypertext Markup Language (HTML) files, Lotus® WordPro files, compressed and uncompressed Tagged Image File Format (TIFF) images, and so on.
  - The OnDemand PDF indexer can be used to create index data for Adobe Acrobat PDF input files.

The indexing programs may be run on the Library Server or an Object Server. ACIF may also run on an OS/390 system, and the output can be transferred to the OnDemand server for loading.

- *Data loading* programs that can be set up to automatically store report data into application groups and update the database. The data loading programs may run on the Library Server or on an Object Server.
- Archived reports and resources.

- A *server print* facility that allows users to reprint a large volume of documents at a high speed. OnDemand uses Infoprint®, which must be purchased separately, to manage the server print devices.
- OnDemand *management programs* to maintain the OnDemand database and documents in cache storage.
- A *system logging facility* that provides administrators with tools to monitor server activity and respond to specific events as they occur. The interface to the system logging facility is through the System Log folder and the System Log user exit.

Download

Download is a licensed feature of PSF for OS/390. Download provides the automatic, high-speed download of JES spool files from an OS/390 system to an IBM DB2 Content Manager OnDemand for Multiplatforms server. Download can be used to transfer reports created on OS/390 systems to the server, where you can configure OnDemand to automatically index the reports and store the report and index data on the system. Download operates as a JES Functional Subsystem Application (FSA) and can automatically route jobs based on a JES class or destination, reducing the need to modify JCL. Download uses TCP/IP protocols to stream data at high speed over a LAN or channel connection from an OS/390 system to the OnDemand server. See PSF for OS/390:Download for OS/390 for more information about Download.

Data indexing and loading

The reports that you store in OnDemand must be indexed. OnDemand supports several types of index data and indexing programs. For example., you can use ACIF to extract index data from the reports that you want to store on the system. An administrator defines the index files and other processing parameters that ACIF uses to locate and extract index information from reports. OnDemand data loading programs read the index data generated by ACIF and load it into the OnDemand database. The data loading programs obtain other processing parameters from the OnDemand database, such as parameters used to segment compress, and store report data in cache storage and in archive storage.

If you plan to index reports on an OnDemand server, you can define the parameters with the administrative client. The administrative client includes a report wizard that lets you create an ACIF indexing parameters by visually marking up sample report data. OnDemand also provides indexing programs that can be used to generate index data for Adobe PDF files and other types of source data, such as TIFF images. See the *IBM DB2 Content Manager OnDemand for Multiplatforms: Indexing Reference* for details about the indexing programs provided with OnDemand.

Figure 3-3 shows an overview of the data preparation process.

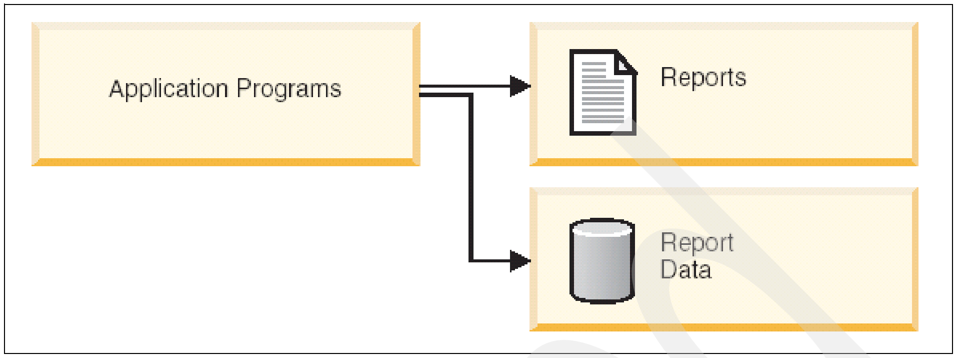


Figure 3-3 Data preparation, indexing and loading (part 1 of 2)

In the picture, user-defined application programs generate printed reports and save report data to disk. The report data can be transmitted to an OnDemand server for indexing and loading. There are a number of methods that you can use to transmit the report data to the server. For example, you can use Download to transmit data from the JES spool to an OnDemand server.

Figure 3-4 shows an overview of the data indexing and loading process.

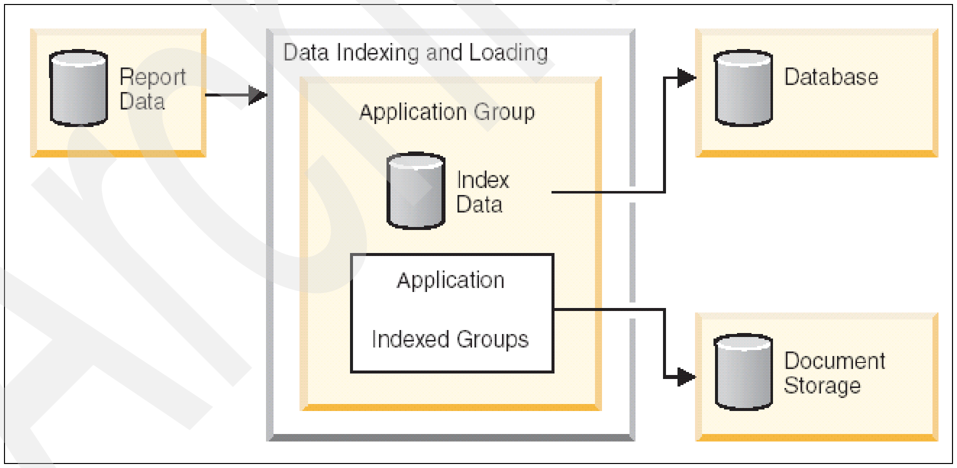


Figure 3-4 Data preparation, indexing and loading (part 2 of 2)

The OnDemand data loading program first determines whether the report needs to be indexed. If the report needs to be indexed, the data loading program calls the appropriate indexing program. The indexing program uses the indexing parameters from the OnDemand application to process the report data. The indexing program can extract and generate index data, divide the report into indexed groups, and collect the resources required to view and reprint the report.

After indexing the report, the data loading program precesses the index data, the indexed groups, and the resources using other parameters from the application and application group. The data loading program works with the database manager to update the OnDemand database with index data extracted from the report. Depending on the storage management attributes of the application group, the data loading program may work with the cache storage manager to segment, compress, and copy report data to cache storage and the archive storage manager to copy report data to archive storage.

Management programs

OnDemand provides programs to maintain and optimize the database and maintain documents in cache storage. An administrator usually determines the processing parameters for these programs, including the frequency with which the programs should run. When someone in your organization creates an application group, they specify other parameters that these programs use to maintain the report data stored in the application group.

For example, when creating an application group, the administrator specifies how long documents should be maintained on the system and whether index data should be migrated from the database to archive storage. The programs use the information to migrate documents from cache storage to archive storage, migrate index data from the database to archive storage, and delete index data from the database. The functions are useful because OnDemand can reclaim the database and cache storage space released by expired and migrated data. IBM recommends that you configure your OnDemand system to automatically start these management programs on a regular schedule, usually once every night or week.

The archive storage manager deletes data from archive storage when it reaches the storage expiration date. An administrator defines management information to the archive storage manager to support the OnDemand data it manages. The management information includes the storage libraries and storage volumes that can contain OnDemand data, the number of copies of a report to maintain, and how long to keep data in the archive management system.

OnDemand and the archive storage manager delete data independently of each other. Each uses its own criteria to determine *when* to remove documents; and each uses its own utilities and schedules to remove documents. However, for final removal of documents from the system, you should always specify the same criteria to OnDemand and the archive storage manager. For example, the Life of Data, which is used by OnDemand, and the Retention Period, which is used by Tivoli Storage Manager, should specify the same value.

Remote Library Server

The OnDemand Windows server configurator program can be used to add an instance of OnDemand that is running on some other system.This feature supports using the IBM DB2 Content Manager CommonStore product on a Windows system to access an OnDemand Library Server that is running on an AIX®, HP-UX iSeries™, Solaris, or z/OS® system or some other Windows system. To configure the system to use a remote Library Server, you must install the OnDemand software on the system that is running the CommonStore software, add a Local Windows Server by using the OnDemand configurator program, and configure an instance of the Local Windows Server to access the remote Library Server.

To configure an instance to access a remote Library Server, you must specify the TCP/IP host name alias, fully-qualified host name, or IP address of the system on which the remote Library Server is running and you must specify the TCP/IP port number of the remote Library Server.The programs on the local Windows system communicate with the remote Library Server through the specified TCP/IP port.

This feature also supports running the OnDemand server programs (such as ARSDB, ARSDOC, and ARSLOAD) from the command line on a Windows system to access an instance of OnDemand that is running on an AIX, HP-UX, iSeries, Solaris, or z/OS system or some other Windows system. This scenario requires that you install the OnDemand software on the local Windows system, add a Local Windows Server by using the configurator program, and configure an instance of the Local Windows Server to access the remote Library Server. When you run the programs, you must specify the -h and -I parameters to identify the host and the instance that you want to process.

4

Case study

In this chapter, we present the business case that we use throughout the redbook, to show how to implement Web applications to manage content in Content Manager, OnDemand, and UDB DB2 using Information Integrator for Content and OnDemand Web Enablement Kit.

We cover the following topics:

- Business case
- Redbrook County legacy system
- Business problem
- Proposed solution

4.1 Business case

In Redbrook County, the construction of a new structure or the remodeling of an existing property requires that the applicant obtains a building permit. A building permit is a document allowing for the construction of a structure in accordance with the terms of the permit. The building permit application and review process ensure that the plans for construction comply with Redbrook County's land use and construction standards. In addition to a building permit application, Redbrook County also requires that the applicant provides a drawing of the new or remodeled structure as well as a drawing of the plot.

4.1.1 Current building permit process

Figure 4-1 shows the process of acquiring a building permit until the completion of a remodeling project. Note that this process is limited to exterior and interior remodel projects by a property owner.

The current building permit process is outlined as follows:

1. The property owner goes to the Redbrook County office and picks up a building permit application package.
2. The owner completes the building permit application and the drawings of the remodeled structure and the plot.
3. The Owner submits the building permit application and the drawings to Redbrook County.
4. The Redbrook County clerk reviews the application package and determines the fees based on whether it is a new application or a resubmission.
5. The owner issues a check to pay the fees.
6. The Redbrook County clerk enters building permit data into the legacy system and sends the application package to the County Board. The County Board reviews the permit.
7. The County Board either approves or reject the application. If the County Board approves the application, then we go to Step 11.
8. If The County Board rejects the application, the owner is notified of the rejection.
9. The owner researches building codes adopted by Redbrook County, and then revises the drawings and the application.
10. The owner resubmits the application. The entire process starts again from Step 4.

11. The County Board approves the application, and issues the building permit.
12. The owner starts to remodel the property (with the help of a contractor, or doing the work themselves). During the remodeling process, certain aspects of remodeling must be reviewed. For the sake of simplicity, in this case study, we include only the electrical, mechanical, and plumbing reviews. For each review, an inspection must be performed. Once all the necessary inspections are done and passed, Redbrook County performs a final inspection.
13. When the remodeling is complete and the final inspection is performed and passed, Redbrook County issues a certificate of occupancy to the owner. The County clerk updates the legacy system.
14. Since there is a change in the property structure, Redbrook County reviews the remodeled structure to determine a revised property value.
15. The Legacy system generates a tax increase letter, and the Redbrook County mails the letter to the owner.
16. The owner receives the tax increase letter.
17. The owner can either:
    - Accept the tax increase and pay the tax.
    - Reject the tax increase and fight for the increase.

This concludes our simplified version of the existing building permit process.

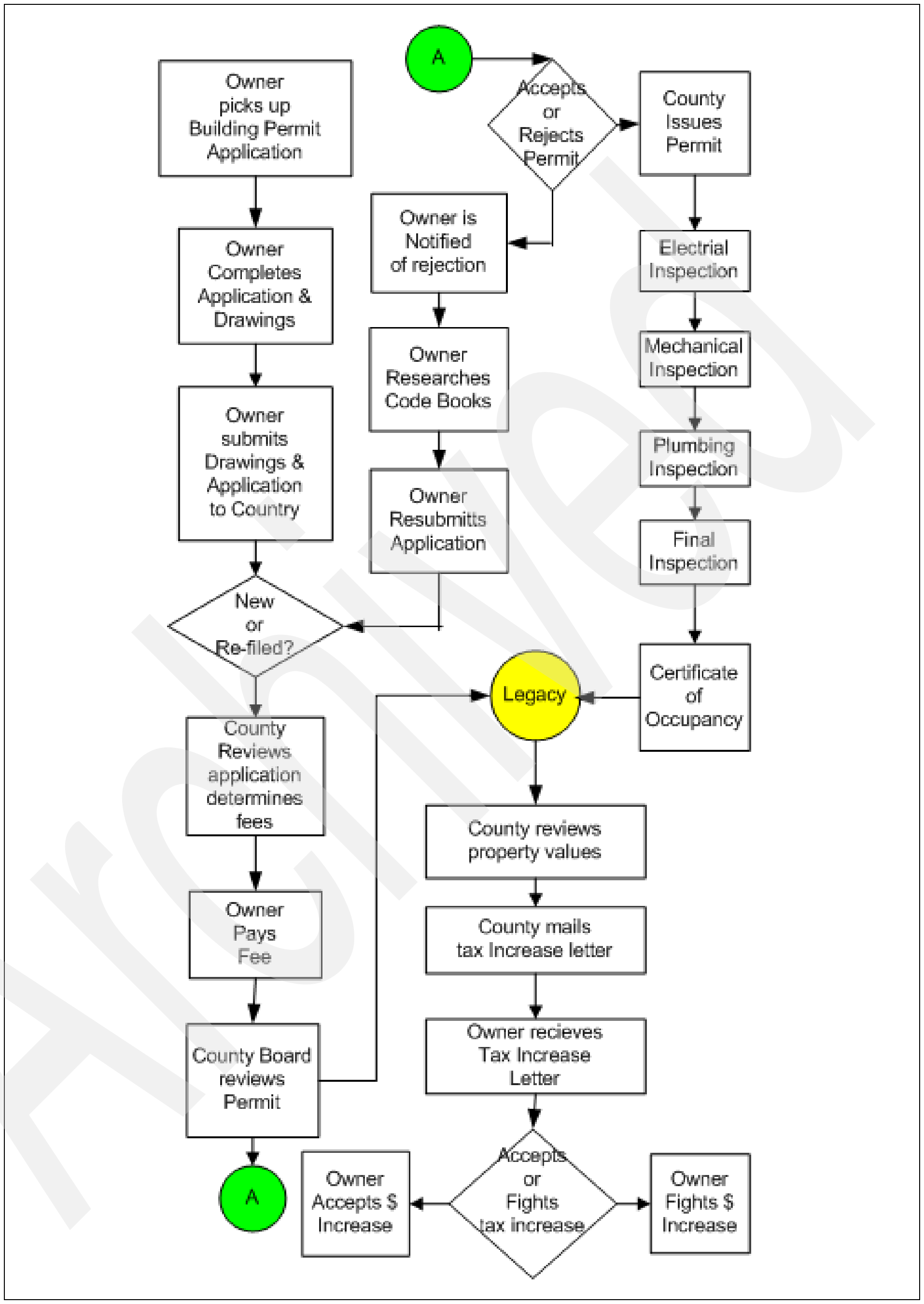

Owner picks up Building Permit Application
Owner Completes Application & Drawings
Owner submits Drawings & Application to Country
New or Re-filed?
County Reviews application determines fees
Owner Pays Fee
County Board reviews Permit
A
A
Accepts or Rejects Permit
Owner is Notified of rejection
Owner Researches Code Books
Owner Resubmitts Application
County Issues Permit
Electrial Inspection
Mechanical Inspection
Plumbing Inspection
Final Inspection
Certificate of Occupancy
Legacy
County reviews property values
County mails tax Increase letter
Owner recieves Tax Increase Letter
Accepts or Fights tax increase
Owner Accepts $ Increase
Owner Fights $ Increase

Figure 4-1 Current building permit process

4.2 Redbrook County legacy system

The Redbrook County legacy system is a client/server application that consists of a front-end software application and a back-end database server. It is used to handle the building permit applications and review process as described in 4.1.1, “Current building permit process” on page 62.

Redbrook County’s data is stored in an IBM UDB DB2 server. All data manipulation and management are done via the legacy application. Figure 4-2 shows the architecture of the Redbrook County legacy system.

The data input to the system includes the following information:

- Owner information
- New permit application
- Payment fees
- Board recommendations
- Permit issued
- Inspections
- Certificate of occupancy
- Tax increases

The system generates the following output:

- General ledger reports
- Tax increase letters
- Certification of occupancy
- Reject letters
- Tax statements

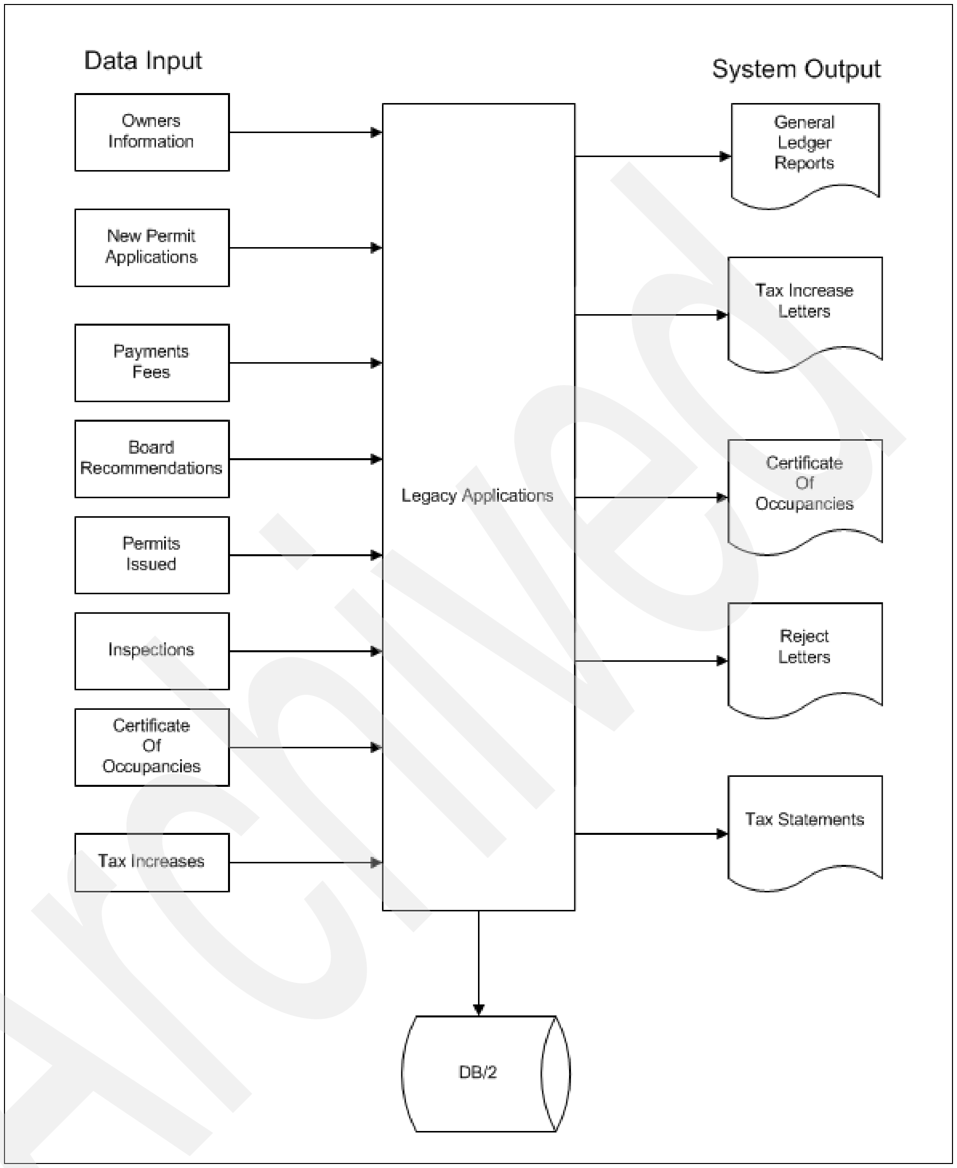


Figure 4-2 Redbrook County legacy system data flow

There are eight tables involved in the legacy system:

- **Property table:** Contains generic property information, such as property number, the address, zoning, number of rooms, living footage, lot size, and the year it was built.

- **Sale information table:** Contains sale information of a property, such as the sale amount, and the date of the sale.
- **Assessment information table:** Contains property assessment records. It includes information such as the property's land value, building value, market value, assessed value, and taxable value.
- **Owner table:** Contains property owner information, such as the owner's social security number, first name, last name, phone number, and current address.
- **Owner history table:** Contains a property owner's history information, such as the purchase date of the property, the amount paid, the property sold date, and the amount sold.
- **Tax discount table:** Contains a property tax discount information, such as total tax, yearly tax discount, and quarterly tax discount.
- **Tax information table:** Contains a property's tax information, such as the social security number of the paid owner, the amount paid, and the payment date.
- **Building permit table:** Contains building permit information, such as the associated property number, various inspection results and dates, fee paid, and who it is issued to.

Figure 4-3 shows the Redbrook County legacy system data schema, with the eight tables, the columns in the tables, and the relationship between all the database tables. Refer to Appendix A.7, "Redbrook County legacy system database schema" on page 525 for complete database schema for the legacy system.

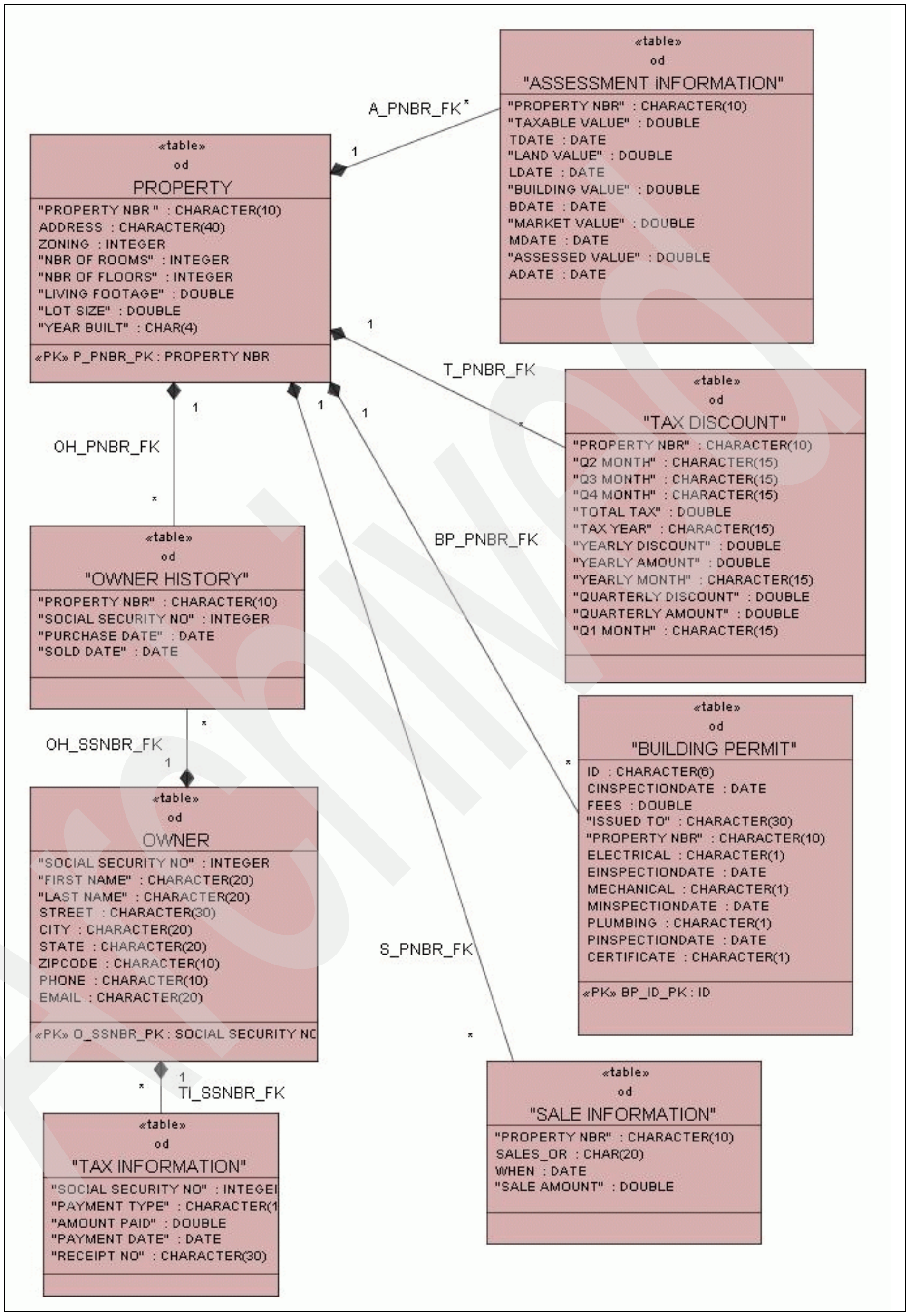


Figure 4-3 Redbrook County legacy system data schema

4.3 Business problem

The entire building permit process is described in 4.1.1, “Current building permit process” on page 62. The process is tedious. It relies on a semi-automated, paper-based system, where a high percentage of the County’s materials are stored in hard copy format. The building permit application forms, completed building permit applications, structure drawings and plots, building code books, credit card receipts, copies of owners’ payment, County’s notification letters, and register receipts are kept in file cabinets or on microfiche. There is no managed system to store and maintain these materials.

Here are some of the problems that Redbrook County has experienced:

- Difficulty in tracking down the current building permit applications
- Loss of materials, such as existing drawings of structures
- Difficulty in locating generated reports

4.3.1 User requirements

The user requirements of the Redbrook County system are as follows:

- Storage and retrieval of unstructured data such as building permit application forms, completed building permit applications, structure drawings and plots, code books, credit card receipts, copies of client’s checks, and County notification letters and register receipts
- Storage and retrieval of data stored on microfiche.
- Storage and retrieval of the code book contents
- Tracking of delinquent taxpayers
- Tracking of notification letters to homeowners, building permit approval and inspection history, and structural changes to property
- Obtaining meaningful reports on home owners, property, and building permit application history
- Viewing of documents relating to properties, including drawings, plots, and notification letters to owners
- A highly scalable, Web-based, content management system to manage and maintain Redbrook County’s materials, including structured and unstructured data
- A Web-based application enabling owners to access Redbrook County’s contents

Based on Redbrook County's user requirements, we implement a Web-based solution using IBM DB2 Content Manager, Information Integrator for Content, and OnDemand products.

4.4 Proposed solution

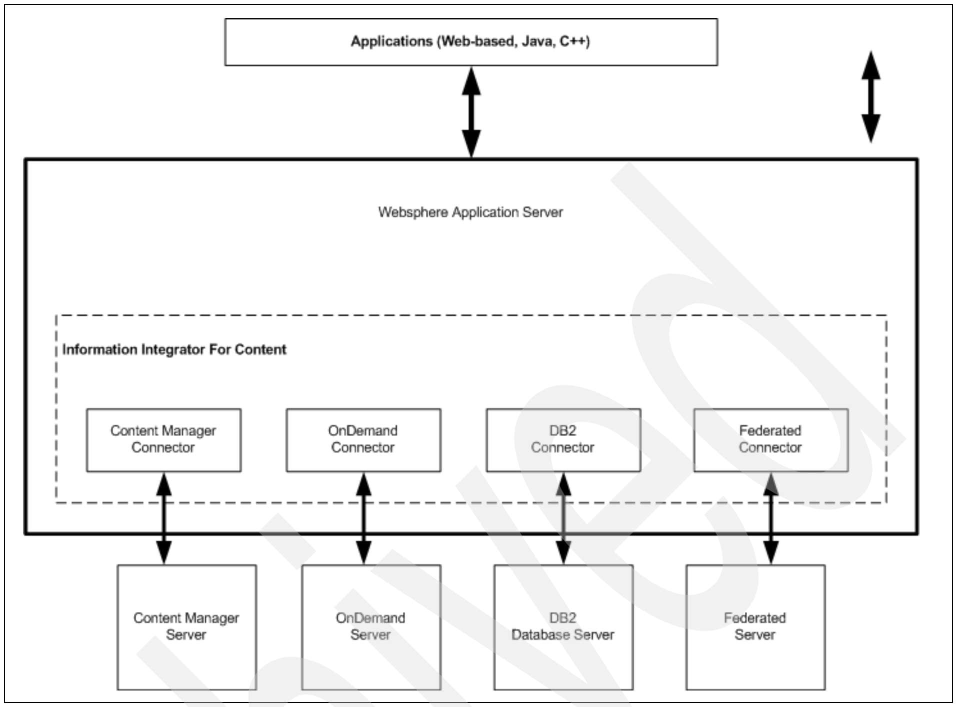
In this section, we present the overall system architecture diagram for the proposed solution, along with the modules that need to be developed for the Web-based application. In addition, we cover the proposed Content Manager data model and OnDemand system setup.

4.4.1 System architecture

Figure 4-4 presents the system architecture for the proposed solution.

The solution uses the following system components:

- **IBM DB2 Content Manager, Version 8.2:** Within this component, we store building permit application forms, completed building permit applications, property structure drawings, plots, and code book contents. We also use the Document Routing feature of Content Manager for managing the flow of the building permit application.
- **IBM DB2 Content Manager OnDemand, Version 7.1.1.1:** Within this component, we store soft copies of generated reports from the legacy system for easy retrieval in later time frame. In addition, we store copies of customers' checks and notification letters.
- **IBM UDB DB2, Version 8.1.2:** Currently, the legacy system uses DB2 for storing data. We continue to use DB2 in the same manner.
- **IBM DB2 Content Manager Information Integrator for Content, Version 8.2:** This product integrates and gives access to multiple repositories spread across a network. We propose to set up a federated server that enables the user to search content against the various types of back-end servers.
- **IBM OnDemand Web Enablement Kit (ODWEK) Version 7.1.1.1:** Using the APIs provided with this toolkit, we show an alternative way of accessing and retrieving data in OnDemand, in addition to Information Integrator for Content. In later chapters, we also show the difference between both sets of APIs. Note that, for simplicity, ODWEK is not shown in the architecture diagram.
- **IBM WebSphere Application Server, Version 5.0.2:** This component is part of the Content Manager system. It allows you to connect systems and applications with internal and external resources.



Applications (Web-based, Java, C++)
Websphere Application Server
Information Integrator For Content
Content Manager Connector
OnDemand Connector
DB2 Connector
Federated Connector
Content Manager Server
OnDemand Server
DB2 Database Server
Federated Server

Figure 4-4 Redbrook County system components

4.4.2 Web-based application modules

We propose to build a Web-based application to manage the electronic forms of the various contents and manage the building permit workflow process. Our Web application consists of seven modules (see Figure 4-5). Each module is designed to accomplish a specific function or set of functions.

The Redbrook County Web application modules are as follows:

- Check status — Check the status of the building permit application process.
- Doc Routing management — Managing the building permit application workflow process.
- Retrieve code book — Manage the retrieval of the building permit code book.
- Retrieve property history — Manage the retrieval of property history information.
- Retrieve tax and fee history — Manage the retrieval of the property tax and fee payment history.
- User account maintenance — Manage the user accounts.
- Import — Used to import application forms and drawings to the system.

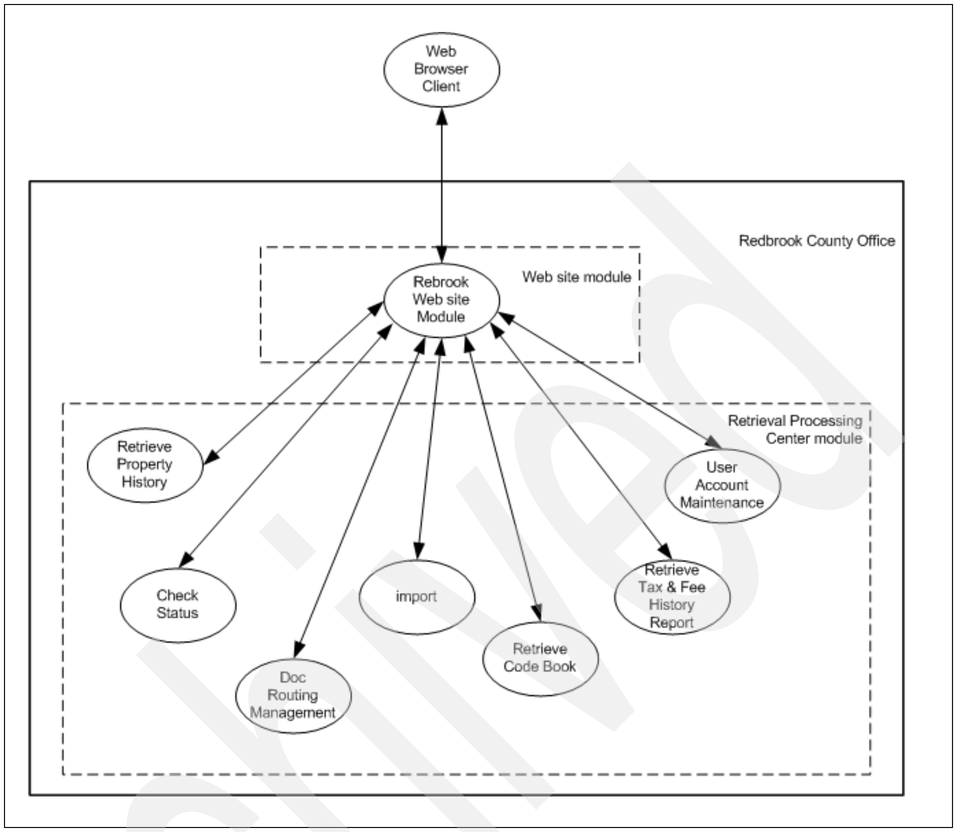


Figure 4-5 Redbrook County Web application modules

4.4.3 Content Manager data model

We implement a Content Manager system that stores and manages building permit applications and drawings of the proposed structure and plots that the owners wish to remodel. In addition, we want to use Content Manager to store the building permit code book. Note that within the existing system, these materials are maintained in file cabinets. Putting them into electronic format enables the applicants to quickly track down the current application status, and the clerks to retrieve, review, and manage the application and the drawings.

In later chapters, we describe how to implement Web-based applications to access data from the Content Manager system using Information Integrator for Content.

The Content Manager system we implement for this case study supports the following types of documents:

- Building permit application forms (GIF image)
- Completed building permit application forms
- Structure drawings (GIF image)
- Structure plots (GIF image)
- Code book contents (PDF or doc documents)

The Content Manager system includes the following attributes:

- BP_NUMBER — Building permit number
- BP_SUB_DATE — Building permit submission date
- BP_TAX_ID — Tax ID (social security number) of the owner
- BP_DOC_DESC — Building permit document description
- BP_CODEBOOK_SUB — Building permit code book description
- BP_CODEBOOK_ID — Building permit code book ID
- BP_REJ_DESC — Building permit application reject description
- BP_STATUS — Building permit application status

The Content Manager system uses the following item types:

- BP_FOLDER — Folder for building permit application
- BP_APPLICATION — Building permit application
- BP_DRAWING — Drawing of the proposed finished structure
- BP_PLOT — Drawing of the property plot
- BP_CODE_BOOK — Building permit code book

For specific information on attributes and item type definitions, see A.8, “Content Manager system definition” on page 529.

4.4.4 OnDemand system setup

We implement an OnDemand system that stores copies of customer’s checks, notification letters, and more importantly, generated reports from the legacy system. This enables County clerks and owners to quickly retrieve generated reports on-line, checks, and letters on-line.

In later chapters, we present two ways of developing Web-based applications to access data from the OnDemand system using Information Integrator for Content and ODWEK.

The OnDemand system we implement for this case study consists of folders, application groups, and reports. Each folder consists of several reports grouped under a specific application group.

We store nine reports in the OnDemand system:

- Property
- Owner hist
- Salesinfo
- Property Owner
- Permits
- Property (different from the first one)
- Assessment
- Tax Discount
- CheckImages

They are grouped into three application groups:

- **PropertyOwner:** This group contains three reports — Property, Owner hist, and Salesinfo.
- **Permits:** This group contains two reports: PropertyOwner and Permits.
- **Tax Assessement:** This group contains four reports — Property, Assessment, Tax Discount, and CheckImages.

There are three folders associated with these application groups:

- **Property Owner:** This folder is associated with the PropertyOwner application group.
- **Permits:** This folder is associated with the Permits application group.
- **Tax Information:** This folder is associated with the Tax Assessment application group.

For specific folder definition and report definition, see A.9, "OnDemand system definition" on page 532.

Part 2

Developing CM Web applications with Information Integration for Content

In this part of the book, we describe how to develop Web applications using Information Integrator for Content Manager.

5

Information Integrator for Content programming overview

Information Integrator for Content (II for Content) provides a set of class libraries and application programming interfaces (APIs) to access and manage content from Content Manager server and other types of content servers. You can integrate an existing application to search and manage data from these content servers or create custom Content Manager applications to fulfill business requirements.

In this chapter, we cover the following topics:

- ► API overview
- ► Information Integrator for Content Java classes
- ► Information Integrator for Content Java beans
- ► Java viewer toolkit
- ► Content server and DDO concept
- ► Mapping the terminology

This chapter provides only the overview of Information Integrator for Content. In subsequent chapters, we use the case study and provide sample code that works with some of the Information Integrator for Content APIs, including Java beans.

5.1 API overview

The Information Integrator for Content application programming interfaces (APIs) are a set of classes that access and manipulate either local or remote content.

The APIs support:

- A common object model for data access
- Multiple searches and updates across a heterogeneous combination of content servers
- A flexible mechanism for using a combination of search engines such as the Content Manager text search feature
- Workflow capability
- Text analysis capability (Information Mining Service)
- Document viewing capability
- Administration functions
- Client/server implementation for Java applications

There are Java and C++ Object Oriented (OO) APIs. The Java OO APIs are used to build a set of Java bean classes, non-visual beans, visual beans, and a Java viewer toolkit.

Within Java APIs, there are visual beans and non-visual beans.

In this redbook, we focus mostly on non-visual Java beans that are used to develop Web applications. We also cover the basics of the Java API classes with samples. The C++ API is not discussed.

The Java APIs can reside on both the server and the client (both provide the same interface). The client API communicates with the server to access data through the network via Java RMI (Remote Method Invocation). Communication between the client and the server is performed by classes. It is not necessary to add additional programs.

Java classes consist of four packages as part of the com.ibm.mm.sdk:

- server (com.ibm.mm.sdk.server) — Access and manipulate content server information. Communicate directly with the federated or back-end content server.
- client (com.ibm.mm.sdk.client) — Communicate with the server package using Remote Method Invocation (RMI).
- cs (com.ibm.mm.sdk.cs) — Connect the client or server dynamically.

- com.ibm.mm.sdk.common — Common classes for both the server package, client package, and the cs package.

The client and server classes provide the same APIs, but have different implementations. The common classes are shared by both the client and server.

Sometimes, an application does not know where the content resides. For example, an application can have content residing on the client at one time and on the server at another time. The com.ibm.mm.sdk.cs package connects the client and server dynamically depending on INI file settings.

A client application must import the client package, a dynamic application must import the cs package, and a server application must import the server package. All applications must use a common package, regardless.

Although the same API is provided for the client and server, the client package has an additional exception item because it communicates with the server package.

The Information Integrator for Content Java beans can be divided into:

- **Non-visual Java beans:** Use the non-visual beans to build Java and Web client applications that require a customized user interface. The non-visual beans support the standard bean programming model by providing default constructors, properties, events, and a serializable interface. You can use the non-visual beans in builder tools that support introspection.
- **Visual Java beans:** The visual beans are customizable, Swing-based, graphical user interface components. Use the visual beans to build Java applications for Windows. You can place them within windows and dialogs of Java-based applications. Because the visual beans are built using the non-visual beans (as a data model), you must use them in conjunction with the non-visual beans when building an application.

The sample applications discussed in the following chapters show how to create Web applications using the Java APIs and the Java beans.

Information Integrator for Content provides a *servlet* with pluggable actions that can be used when building Web applications. This servlet acts as a controller of a Model-View-Controller design Web application. It performs actions and initializes the beans (the model), which are then accessed in the JSPs (the views) either directly or indirectly by using JSP tags. This is illustrated in Figure 5-1.

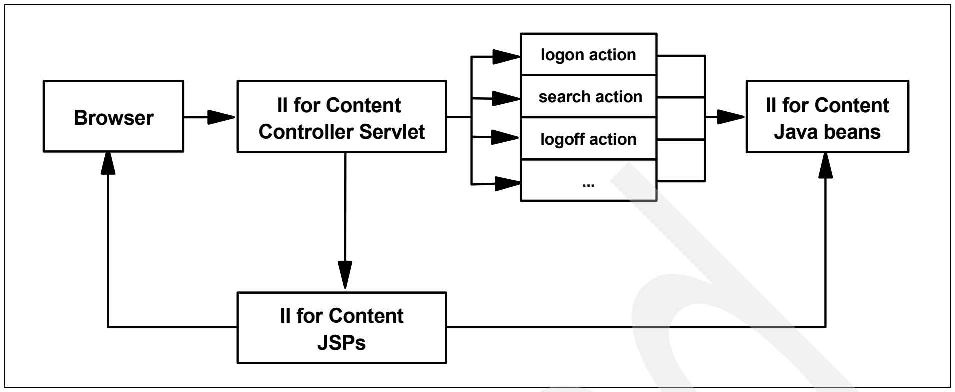


Figure 5-1 Controller servlet overview

Actions performed for typical application tasks include these:

- Log on and log off.
- Search.
- Create, retrieve, modify, and delete documents.
- Create folders, and add documents to or remove documents from folders.
- Launch documents and document pages for viewing.

In addition, the servlet performs common tasks before and after the action, such as management of the connection to the content server. After every action, a JSP is invoked to format the results and send them back to the browser.

You can customize the servlet to add new actions and associate JSPs with the actions.

The *JSP tag library* enables an application to dynamically create HTML files in JSPs without the use of Java code. The tag library consists of an XML file (taglib.tld) that defines each tag with its parameters and the name of the Java class that implements the tag behavior.

The tag library supports the creation of HTML files with functions such as these:

- Iterate through available data sources.
- Iterate through items in search results.
- Iterate through attributes of an item.
- Iterate through available search templates.
- Iterate through search criteria of a search template.
- Iterate through available entities.

In the following sections, we briefly cover Java classes and non-visual beans that we use to develop our sample Web applications.

5.2 Information Integrator for Content Java classes

In this section, we cover the main Java classes (Java APIs) used to access content. These classes are:

dkDatastore	Represents the content server as a datastore in your program. It manages the connections to the content server and the transactions and commands for the server, including the methods to perform a query.
dkDatastoreDef	Defines the methods to access the content in the datastore. It includes methods to access the data model for the content server. It has subclasses to represent administration of the datastore and the data model for that datastore.
dkDatastoreAdmin	Provides a generic interface to access administration functions in the datastore. It contains an imbedded interface dkAccessControl which provides access control functions, such as managing user IDs, passwords, user groups, and creating access control lists. It has two main subclasses: dkUserManagement and dkAuthorizationMgmt.
dkEntityDef	Defines an interface to access and manipulate the entities defined in the datastore. An entity is a category of the metadata representation of content. It corresponds to a native entity on the content server. For example, in Content Manager, it is item type. In relational database, it is a table.
dkAttrDef	Provides a generic interface to manipulate the attributes of the entity.
dkUserManagement	Defines a generic interface to manage user and group of users for the datastore. It has two main subclasses: dkUserDef and dkUserGroupDef.
dkAuthorizationMgmt	Defines a generic interface to represent and processes datastore authorization management functions. It has two main subclasses: dkAccessControlList and dkPrivilegeSet.
dkUserDef	Defines a generic interface to manage users in a datastore such as the user's name, description, password, and user's privilege set.
dkUserGroupDef	Defines a generic interface to manage a user group in a datastore with actions such as adding and updating users.

dkAccessControlList	Defines a generic interface representing an access control list in different content server.
dkPrivilegeSet	Defines a generic interface to manage a named group of privileges.

For each content server, concrete implementations of these classes are provided for specific content servers. The implementations have a suffix to distinguish them. Table 5-1 lists the content server types, their associated suffix, and some examples.

Table 5-1 Content server and its associated class name suffix

Content Server	Suffix	Example
Content Manager Version 8	ICM	DKDatastoreICM
Content Manager OnDemand	OD	DKDatastoreOD
Content Manager for AS/400®	V4	DKDatastoreV4
Content Manager ImagePlus® for OS/390	IP	DKDatastoreIP
Domino®.Doc®	DD	DKDatastoreDD
Extended Search	DES	DKDatastoreDES
Panagon Image Services (FileNet)	FN	DKDatastoreFN
Relational database classes	DB2 JDBC ODBC	DKDatastoreDB2 DKDatastoreJDBC DKDatastoreODBC
Earlier version of Content Manager	DL	DKDatastoreDL

5.3 Information Integrator for Content Java beans

Information Integrator for Content Java beans consist two sets of classes:

- **Non-visual beans:** These beans are useful in building Web applications, and other Java applications.
- **Visual beans:** These beans provide user interface "panels" of an application and can be used to quickly build Swing-based Java client applications.

The Information Integrator for Content Java beans are designed to ease development of end-user applications. The beans follow Java beans conventions, with default constructors, properties, and events. Also, the beans include associated BeanInfo classes, which aids their use in visual builder environments.

Table 5-2 describes the packages and JARs containing these classes:

Table 5-2 Classes and their locations

Class	Package	JAR
Non-visual beans	com.ibm.cmm.beans com.ibm.cmm.beans.util com.ibm.cmm.beans.workflow	cmb81.jar
Visual beans	com.ibm.mm.beans.gui	cmb81.jar
Java viewer toolkit	com.ibm.mm.viewer com.ibm.mm.viewer.annotation	cmbview81.jar

5.3.1 Non-visual beans

The non-visual beans are useful in building Web Applications, application environments where the application developer wants complete control over the user interface provided, and applications where no interface is provided, such as in a custom server application.

When used in Web applications, the beans typically serve as the model in a Model-View-Controller architecture. Servlets, written by the application developer or provided with Information Integrator for Content, act as the controller. Java Server Pages (JSPs) act as the view. This architecture provides separation of application logic form presentation logic; however, other architectures are possible when building Web Applications, such as JSP-only, which would both act as the controller and view, using the non-visual beans as the model.

Both of these types of Web applications are supported with the Information Integrator for Content non-visual beans. In Chapter 7, “Building a generic application” on page 155, we show how you can build Web applications using these non-visual beans.

The packages comprising the non-visual beans are com.ibm.mm.beans, com.ibm.mm.beans.util, and com.ibm.mm.beans.workflow. Here is a brief description of each of the beans:

CMBConnection	Maintains the connection to the federated database and content servers. By default, a connection is made to a federated database, but a direct connection can also be made to a content server.
CMBConnectionPool	Maintains a pool of CMBConnection instances. It provides a performance

	optimization for server applications in situations where the same user ID is used by multiple users by avoiding a disconnect and reconnect when the a CMBConnection instance for the same user ID is reused.
CMBSchemaManagement	Provides access to available search templates, entities and attributes. An instance of this bean can be obtained from CMBConnection, in which case it returns schema information about the server or database connected to by CMBConnection. Alternatively, you can create an instance separate from CMBConnection, to gain access to schema information for any content server.
CMBDataManagement	Provides services to retrieve data on documents and folders stored on content servers. You can obtain an instance of this bean from CMBConnection.
CMBUserManagement	Provides capabilities to view and modify the content server user ID mapping associated with an Information Integrator for Content federated user ID.
CMBQueryService	Provides query capabilities, either by search template or by using a query string.
CMBSearchResults	Maintains search results that are generated by searches performed using CMBQueryService or CMBSearchTemplate.
CMBDocumentServices	Provides rendering and conversion of documents for thick and thin clients (see Document Viewing Services below).
CMBWorkFlowDataManagement	Provides services to retrieve workflow data.
CMBWorkFlowQueryService	Provides workflow query capabilities.
CMBDocRoutingDataManagementICM	Provides services to manage Content Manager Version 8 Document Routing data.
CMBDocRoutingQueryServiceICM	Provides Content Manager Version 8 Document Routing query services.
CMBTraceLog	Provides common trace event handling for the non-visual beans. It can write the trace messages to a log file or display them in a

	window. For Version 8, this bean can write trace to a common log file with Java API classes, for ease in debugging problems.
CMBExceptionHandler	Provides common exception handling for exception events generated by other non-visual beans.

There are additional classes in the non-visual beans that are used in conjunction with the beans for data modeling:

CMBEntity	Represents a single entity definition. An entity is an item type, index class, application group, database table.
CMBAttribute	Represents a single attribute definition. This is an attribute on an item type, index class, application group. For relational database sources, it is a column on a data base table.
CMBSearchTemplate	Represents a search template defined on the Federated database, or an OnDemand folder when direct-connected to an OnDemand server.
CMBSTCriterion	Represents a single criterion on a search template.
CMBItem	Represents a single item on the server. This is typically a document or folder. For database sources, this will be a row in a database table. For CM Version 8, this can also be a resource or non-resource item, or a child component of an item.
CMBObject	Represents a single part of a document. For multipart documents, or documents with annotations and notes, each part on the server for the content of the document or annotations or note will be represented with a CMBObject.
CMBAnnotation	Annotation parts have additional information that is represented using this class.

Multithread support

A single instance of the CMBConnection bean does not support multithreaded use. Other beans, with the exception of CMBConnectionPool, contain a reference to the CMBConnection bean. This means that an entire set of beans, a CMBConnection bean and all beans that reference it, can only be used on a single thread at any time.

The lack of multithread support in CMBConnection does not imply that the beans cannot be used in a multithreaded environment. Quite the contrary — Web application environments are typically multithreaded, and the beans are used in this environment. But, they are managed in such a way as to only allow a single thread to use any set of beans (CMBConnection and associated beans) on any single thread at any point in time.

For Web applications, this is usually achieved by placing an instance of CMBConnection and associated beans in the session scope of the Web application. Also, it requires that only a single servlet be executing for a session at a time, which can be achieved by serializing on CMBConnection.

CMBConnectionPool can assist in managing connections in a multithreaded environment. When a connection is needed, it can be obtained from CMBConnectionPool, and then returned to CMBConnectionPool when it is no longer needed.

5.3.2 Visual beans

The Information Integrator for Content visual Java beans can be used to rapidly build Swing-based Java applications using the search capabilities of Information Integrator for Content. These classes work both with Federated and when connected directly to the native server, although some beans only provide function when connected to a particular server (because of features only available on that server). The classes comprising the visual beans are contained in the Java package com.ibm.mm.beans.gui.

Here is a brief description of each of these beans:

CMBLogonPanel	Displays a panel for logging onto the federated database. It also provides the capability to modify user IDs and passwords on the content servers.
CMBSearchPanel	Provides an interface for launching searches based on an entity and its attributes. The entity depends on the platform: Item Type (CMv8), Index Class (CMv7), Application Group (OnDemand).
CMBSearchTemplateList	Displays a list of available search templates and allows selection of a template.
CMBSearchTemplateViewer	Displays a search template. It provides fields for search criteria to be entered. It also performs a search.
CMBSearchResultsViewer	Displays the results of a search. In situations where folders are returned, use this bean to “drill-down” into the folder to see its contents. Items in the search results or folders can be selected and opened for viewing or attribute editing.

CMBFolderViewer	Displays the contents of one or more folders. You can use it in situations where a folder that is not returned from a search needs to be viewed.
CMBVersionsViewer	For versioned documents, this bean displays all versions of a document. Attributes related to each version: user ID and time created, ushered and time last updated are displayed. This bean works with documents from CM Version 8, though the connection may be to either Fed or CM.
CMBDocumentViewer	Provides display of document contents and display and update of annotations on those documents. Also, for documents from OnDemand, the OnDemand client (in view-only) mode is launched to view the document and provide view and addition of notes. By changing the cmbmime2app.properties, alternative viewers can be launched to display documents of particular content types.
CMBItemAttributesEditor	This bean displays the attributes of an item and provides the ability to reindex of the item and change attribute values. When it is not possible to update attributes or reindex, these fields will be disabled. However, even in this case, the bean is still useful for displaying the values of the attributes of an item.

Each of the visual Java beans has a Connection property. This property must reference an instance of CMBConnection, the non-visual bean that maintains the connection to the content servers. Therefore, any application built with the EIP visual beans must also contain an instance of the CMBConnection non-visual bean.

The visual beans can be used to build Java Applets. However, since the visual beans use the non-visual beans, which in turn use the Java API, the applet will either need to communicate directly with the content server, database, or to an RMI server. They cannot be used as part of a Web Application that also requires connection to content servers in the middle tier.

5.4 Java viewer toolkit

These classes are used by the non-visual and visual beans to provide document conversion, document rendering, and graphical annotations editing. They can also be used independently of the beans for standalone viewer applications and applets.

Package com.ibm.mm.viewer contains classes comprising the Java viewer toolkit. This toolkit provides both a Swing-based document viewer with annotations editing capabilities, and non-visual document conversion and rendering.

The Java viewer toolkit is used in the CMBDocumentServices non-visual bean and the CMBDocumentViewer visual bean, and provides most of the function of those two beans. The Java viewer classes are separated from the beans so that they can be used in situations where the beans would not be used, such as in a viewer applet or document conversion servlet that does not require connection to a content server, that is, it works with files or data streams.

These are some of the key classes in the Java viewer toolkit:

CMBStreamingDocServices	Provides functions for converting documents and pages of documents to browser displayable formats. It is very similar to the CMBDocumentServices non-visual bean, except that it works with streams rather than referring to other beans classes.
CMBAnnotationServices	Provides functions to parse, add, modify, delete, render, and save CM graphical annotations.
CMBGenericDocViewer	Provides the main panel of a Java Swing based document viewer, with capabilities to scale, rotate, and enhance document pages, create, modify, and save annotations. It also provides toolbars and a pages thumbnails view.
CMBDocument	Represents a document being processed by CMBStreamingDocServices or displayed by CMBGenericDocViewer.
CMBPage	Represents a page of a document.
CMBAnnotationSet	Represents a set of annotations associated with a document.
CMBPageAnnotation	Represents an annotation on a page of a document.

As mentioned earlier, in subsequent chapters, we show how to use specific Java APIs and non-visual beans to create custom Content Manager Web applications with sample code.

5.5 Content server and DDO concept

In order to develop Content Manager Web application using Information Integrator for Content, you need to be familiar with the II for Content Java API classes and beans as briefly discussed in the previous section. Most importantly, you need to know the concept of the content server and DDOs to understand how data can be accessed and manipulated.

5.5.1 Content server

As mentioned in 2.5.1, "Information access" on page 45, a content server is any system that stores multimedia, business forms, documents, and related data, along with metadata that allows users to process and work with the content. A content server is a data repository. It supports user sessions, connections, transactions, cursors, and queries.

Information Integrator for Content supports the following content servers:

- Content Manager Version 8
- Content Manager Version 7
- Domino.Doc
- Extended Search
- ImagePlus for OS/390
- Content Manager OnDemand
- VisualInfo™ for AS/400
- DB2
- DB2 DataJoiner®
- Information Catalog
- DB2 Warehouse Manager Information Catalog Manager
- JDBC/ODBC servers

Applications that use II for Content can create a federated content server, which acts as a common server. II for Content federated classes enable federated searching, retrieval, and updating across several different types of content servers.

The II for Content federated content server and each of the content servers have different schemas. Integrating multiple heterogeneous content servers into a federated system requires conversion and mapping.

The II for Content system administration client includes a schema mapping function. Schema mapping provides the schema information for each content server, and this information is stored in the II for Content administration database. Schema mapping is used during federated searches.

As mentioned earlier in 5.2, "Information Integrator for Content Java classes" on page 81, there are concrete implementations (connectors) of II for Content Java APIs for specific content servers. The implementations have a suffix to distinguish them. Refer to Table 5-3 for the associated suffix of the content servers. Refer to Table 5-1 on page 82 for sample class names.

You can also write your own implementations (connectors) that adhere to the standards using the native API set of the other repository. You can implement as many features as you want in the back-end, but a minimum set of functions would include logon, getting information about the index classes and attributes, search, and retrieve. Refer to the section, "Working with Other Content Servers", in the *IBM DB2 Content Manager Workstation Application Programming Guide*, SC27-1347 for details on creating custom connectors.

Table 5-3 Content servers and its associated class name suffix

Content Server	Suffix
Content Manager Version 8	ICM
Content Manager OnDemand	OD
Content Manager for AS/400 (VisualInfo for AS/400)	V4
Content Manager ImagePlus for OS/390	IP
Domino.Doc	DD
Extended Search	DES
Panagon Image Services (FileNet)	FN
Relational database classes	DB2 JDBC ODBC
Earlier version of Content Manager (Version 7 or earlier)	DL
Data Warehouse Manager Information Catalog Manager	IC

5.5.2 Dynamic data object (DDO) and extended data object (XDO)

As part of the Object Management Groups' (OMG) CORBA standard, the Persistent Object Service interacts with a datastore to get data in and out of an object. The Persistent Object Service specifies three protocols:

- A Direct Access Protocol (uses attributes to store the data)
- ODMG-93 Protocol: uses OMG ODL (Object Definition Language)
- Dynamic Data Object (DDO) Protocol, which is a datastore-neutral protocol

For OMG documentation, use the following link:

http://www.omg.org

In compliance with the OMG's CORBA Persistent Object Service and Object Query Service Specification, Information Integrator for Content provides an implementation of the *Dynamic Data Object* (DDO) and its extension, the *EXtended Data Object* (XDO).

Dynamic data object (DDO)

The *dynamic data object (DDO)* is an interface used to retrieve data from and load data into a content server. DDO represents the persistent data of an object that is not specific to a content server. An object can either be in a *dynamic state* which typically stay in memory and does not persist after the object life cycle, or in a *persistent state* which usually is in a storage media. DDO, representing persistent data, only exists in the application and does not exist after an application terminates.

A DDO has a single persistent ID (PID), an object type, and a set of data items whose cardinality is called the data count. Each data item can have a name, a value, an ID, one or more data properties, and data property count. Each data property can have an ID, a name, and a value. As an example, a DDO can represent a row in a database table. The DDO's data items and properties represent the columns in the row.

Using DDO, you can keep track of available data sources, have access to the query results, and obtain both metadata and data. Information Integrator for Content provides methods to add, retrieve, update, and delete DDOs.

In the Content Manager Version 8 document model, the document or folder is represented as an item, with a collection of resource items that represent the multiple parts. When programming, a DDO corresponds to the item. The DDO's object type corresponds to the item type. The DDO's data items correspond to the item's attributes.

In general, you need to go through the following programming steps to access and manage data in a content server:

1. Create and instantiate a content server.
2. Establish a connection to the content server.
3. Create and instantiate the DDOs to be operated on, and associate the content server with the DDOs.
4. Add, retrieve, update, and delete the DDOs using appropriate methods.
5. Close the connection and destroy the content server.

Extended data objects (XDO)

Extended data object (XDO) complements DDO by storing multimedia data of complex types and offering functions that implement the data type's behaviors in the application. An example of this is a resource item that is used to store an image or document in Content Manager.

XDO can be contained in, or owned by, DDO to represent a complex multimedia data object. It can also be a stand-alone dynamic object.

Similar to DDO, XDO has a set of properties to represent such information as data types and IDs.

Persistent ID (PID)

The persistent ID (PID) uniquely identifies a persistent object in any content server. A DDO's PID consists of an item ID, a content server name, and other related information. When a DDO is added to a content server, the system assigns a unique PID to the DDO.

Because a DDO is a dynamic interface to persistent data that is moved in or out of content servers, different DDOs can represent the same persistent data entities, and therefore different DDOs can have the same PID. For example, a DDO can be created to move a data entity into a content server to store data persistently, and another DDO can be created to hold the same data entity that is checked out from the same content server for modification. In this case, these two DDOs share the same PID value.

5.6 Mapping the terminology

In this redbook, we show you how to build sample applications using three types of content servers: Content Manager, Content Manager OnDemand, federated server, and UDB DB2.

Let us review some of the basic terminology used when programming with Information Integrator for Content:

Content server	The server where the data is stored.
Data source	The mechanism by which the data could be accessed.
Entity	Designates an object or data unit from which the data is retrieved.
Attributes	The characteristics that identify a stored information and which could be used to retrieve it from the back-end content server.

DDO	Corresponds to an item in Content Manager, which is an instance of some item type.
DDO's object type	The item type in Content Manager.
DDO's data items	The attributes that define an item type in Content Manager.

Table 5-4 maps the terminology to different content servers.

Table 5-4 Terminology mapping to servers

Content server	**Data source**	**Entity**	**Attribute**
Content Manager Version 8	Library Server	item type	attribute
Content Manager Version 7 or earlier	Library Server	index class	attribute, key attribute
Content Manager OnDemand	OnDemand Library Server	application group folder	field, criteria
relational database	DB2, JDBC, ODBC, DataJoiner	table	column
Federated content server	mapping server	mapped federated entity	mapped federated attribute

6

Quick start in Web application development

In this chapter, we focus on the programming details required when working with Information Integrator for Content (II for Content) Java OO APIs. We include a special section on how to set up WebSphere Application Developer environment in order to develop a Web application using Information Integrator for Content.

We cover the following topics:

- Introduction
- Development environment setup
- Developing sample code with the Java OOAPIs, including these topics:
  - Working with attributes
  - Working with item types
  - Working with items and importing documents

6.1 Introduction

As mentioned in 5.5.2, “Dynamic data object (DDO) and extended data object (XDO)” on page 90, Information Integrator for Content creates DDOs dynamically after it connects to a content server. The association between a DDO and a content server is established with the DDO’s PID. In general, an application using the II for Content APIs goes through the following five steps to manage data in a content server:

1. Create a content server.
2. Establish a connection to the content server.
3. Create the DDOs to represents the data while the application runs.
4. Add, retrieve, update, and delete the DDOs using appropriate methods.
5. Retrieve the corresponding XDO if needed
6. Close the connection and destroy the content server.

Information Integrator for Content comes with a set of Java OO APIs and Java beans. In this chapter, we show you how to get a quick start in development with the following topics:

- The setup and usage of WebSphere Studio Application Developer
- The usage of HTML, JSPs, and servlets

Several sample files (Figure 6-1) are involved that demonstrate the usage of Information Integrator for Content Java OOAPIs. They are as follows:

- index.html — Main HTML page.
- ConnectionController.java — Controller servlet.
- RetrievePlots.java — Retrieve image servlet.
- CMaddAttr.jsp — Create attributes for the building permit application.
- CMdelAttr.jsp — Delete attributes from Content Manager data model.
- CMaddItemType.jsp — Add item types for the building permit application.
- CMimportPlots.jsp — Import plot drawings as sample data.
- CMretrievePlots.jsp — Retrieve plot drawings.
- ViewObject.jsp — View images via container.
- InitClass.java — Initialization servlet.

We discuss what these files are, and how to build them, in later sections of this chapter.

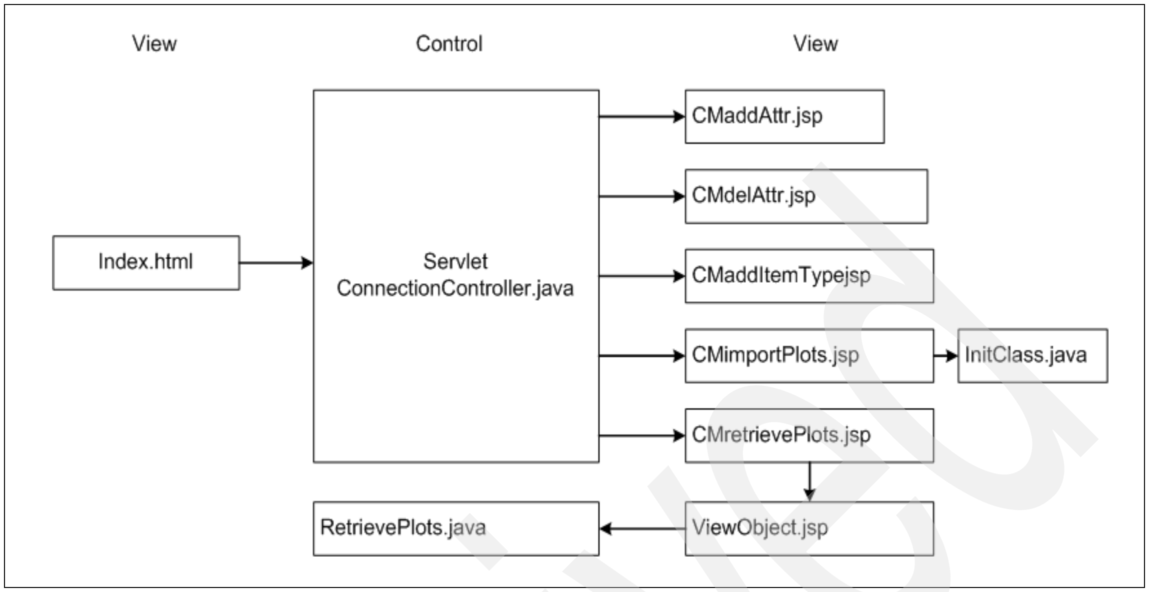


Figure 6-1 Sample codes using Information Integrator for Content Java OOAPIs

6.2 Development environment setup

Before you begin, make sure that Content Manager is installed correctly, and that your Client for Windows and eClient work properly.

To develop a Content Manager Web application using Information Integrator for Content, we need to set up a proper development environment. In this redbook, we use WebSphere Studio Application Developer as the development of choice.

If you do not have the software, you can obtain a trial version:

1. Go to:

   `http://www.ibm.com/software/awdtools/studioappdev`

2. Click **Trials and betas** from the left-hand navigator panel.
3. Scroll down the page, and you should find the trial version for WebSphere Studio Application Developer for Windows.
4. Click the link, and follow the instructions for download.

Although a newer version is available now, we recommend using Version 5.1.1 while developing the samples described in this redbook. You can always upgrade it to a later version once you are familiar with the tool.

Note: In this redbook, we develop all our applications using WebSphere Studio Application Developer v5.1. The specific commands we use cater to this version. If you use a different version, you may encounter slight differences when performing the steps we mention in this redbook; however, the general steps and ideas should be the same.

6.2.1 Setting up the development directory

We highly recommend that you set up a separate directory for WebSphere Studio Application Developer environment when developing and working with the sample codes provided in this redbook. This allows you to have a clean independent development environment, which does not interfere with any other development work that you maybe doing.

To set up a separate development environment, follow these steps:

1. Create a dedicate file system directory. In our scenario, we use c:\wsad_sg246338.
2. Create a new WebSphere Studio Application Developer shortcut to start with the following command:

   ```
   "<WSAD_HOME>\wsappdev.exe" -data c:\wsad_sg246338
   ```

   In this command, <WSAD_HOME> is the directory where you installed the WebSphere Studio Application Developer software. Note that if you did not create the directory, when you first start the WebSphere Studio Application Developer, a corresponding directory will be created for you based on the input data parameter value.

   Rename the shortcut to something similar to WSAD_CM to distinguish it from the generic shortcut.
3. Start the WebSphere Studio Application Developer. If you made the configuration as above, your workplace should be empty.

6.2.2 Setting up a new Web project

Before writing any code, we must create and configure a new Web project in the WebSphere Studio Application Developer environment.

There are several steps that we need to perform to accomplish this:

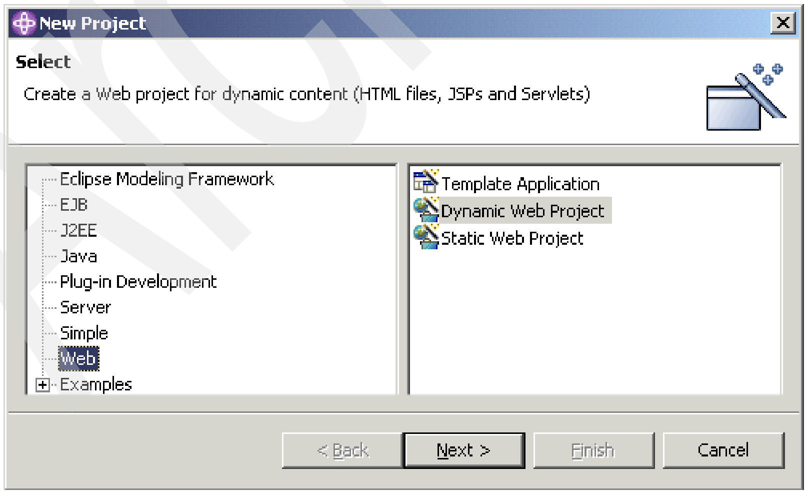
- Create a new Web project. We use II4CProject as our Web project name.
- Add JAR files to the Java build path. We include the following files in the libraries:
  - From the <CMBROOT> directory, cmb81.jar, cmbsdk81.jar, cmbview81.jar, and log4j.jar.
  - From the <DB2_HOME>\java directory, db2java.zip file.

In the following section, we provide the step-by-step procedures in setting up the new Web project.

Create a new Web project

To create a new Web project, do the following steps:

1. Start WebSphere Studio Application Developer. Remember to create a new shortcut that uses a dedicated directory for all the work you will be doing from this redbook.
2. Select **File** → **New** → **Project**.
3. From the New Project window (Figure 6-2), select **Web** from the left panel, and select **Dynamic Web Project** from the right panel. Click **Next**.

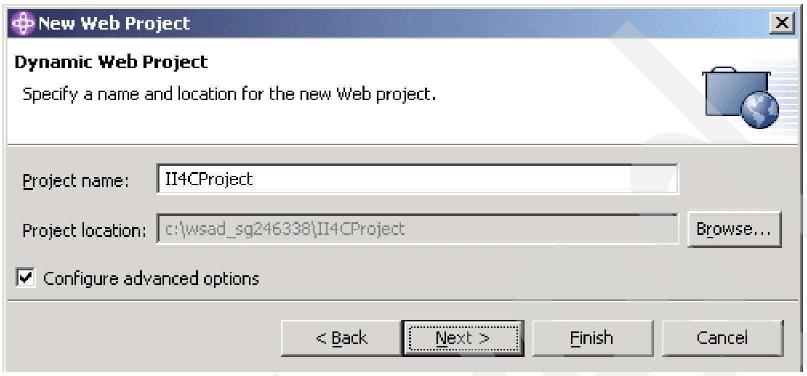

New Project

Select

Create a Web project for dynamic content (HTML files, JSPs and Servlets)

Figure 6-2 Create new Web project: select project type

4. From the New Web Project window (Figure 6-3), enter `II4CProject` for the Project name field. Note that you can always enter a different project name. However, by using the name we use here, it will be easier for you to follow the steps we present in this redbook.



Specify a name and location for the new Web project.

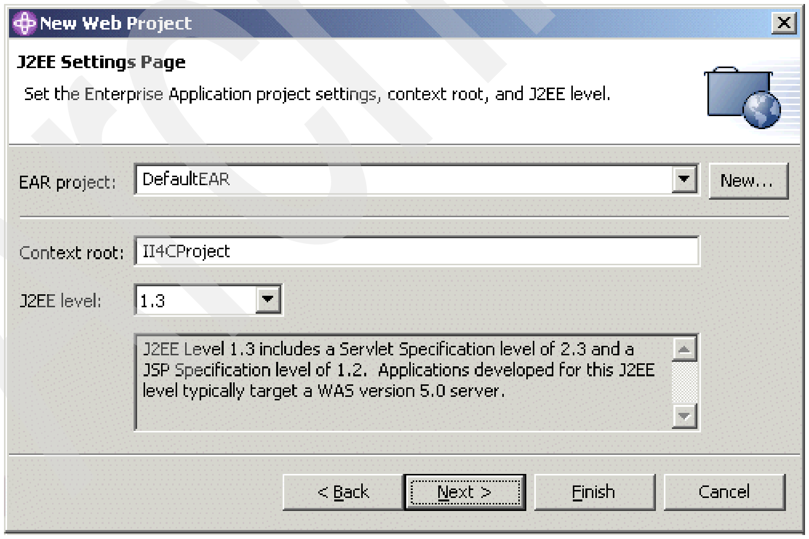
Project name: II4CProject

Project location: c:\wsad_sg246338\II4CProject

☑ Configure advanced options

Figure 6-3 Create new Web project - Enter project name

5. Select Configure advanced options. Click **Next**.
6. From the J2EE Settings Page window (Figure 6-4), enter `II4C_EAR` for the EAR project field.
7. For J2EE level, select `1.3` or higher is selected. Click **Finish**.



Set the Enterprise Application project settings, context root, and J2EE level.

EAR project: DefaultEAR

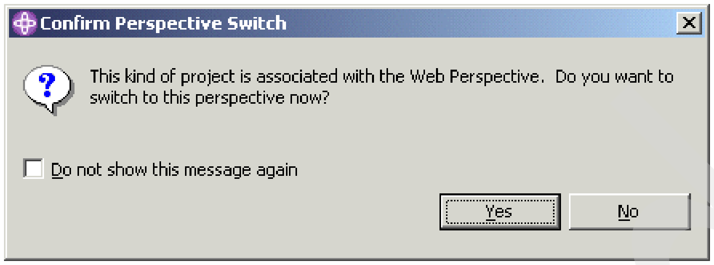
Context root: II4CProject

J2EE level: 1.3

J2EE Level 1.3 includes a Servlet Specification level of 2.3 and a JSP Specification level of 1.2. Applications developed for this J2EE level typically target a WAS version 5.0 server.

Figure 6-4 Create new Web project - Enter EAR project name

8. If the system prompts you to confirm the switch to the Web perspective (Figure 6-5), click **Yes**.


Confirm Perspective Switch

This kind of project is associated with the Web Perspective. Do you want to switch to this perspective now?

☐ Do not show this message again

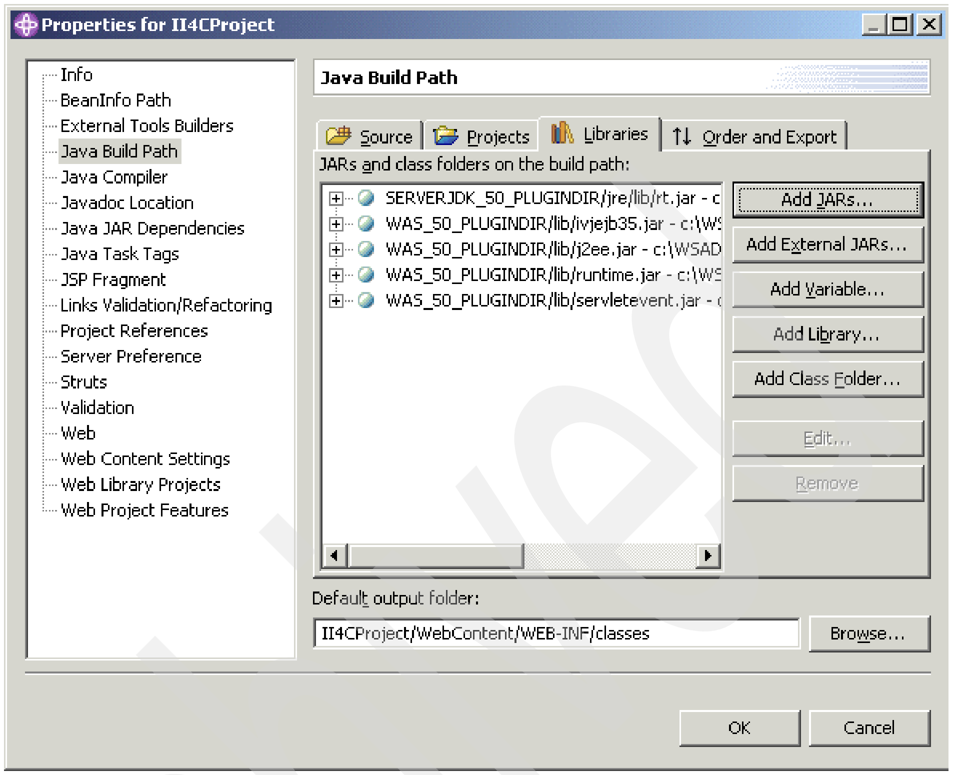
Figure 6-5 Confirm switch to Web perspective

Add JAR files to the Java build path

Following the steps in the previous section, you should see in the Project Navigator window two folders, II4CProject and II4C_EAR.

To add the necessary JAR files to the Java build path for II4CProject, do the following steps:

1. Right-click **II4CProject**, and select **Properties** from the context menu.
2. Select **Java Build Path** from the left panel, and then select the **Libraries** tab on the right panel (Figure 6-6).



JARs and class folders on the build path:

```
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - c
WAS_50_PLUGINDIR/lib/ivjejb35.jar - c:\WS
WAS_50_PLUGINDIR/lib/j2ee.jar - c:\WSAD
WAS_50_PLUGINDIR/lib/runtime.jar - c:\WS
WAS_50_PLUGINDIR/lib/servletevent.jar - c
```

Default output folder: II4CProject/WebContent/WEB-INF/classes

Figure 6-6 Set up Java build path - Initial

3. Add Information Integrator for Content JAR files to the Java build path:
   a. Click **Add External JARs**....
   b. Navigate to the directory <CMBROOT>\lib.

      <CMBROOT> is the location where Information Integrator for Content is installed.

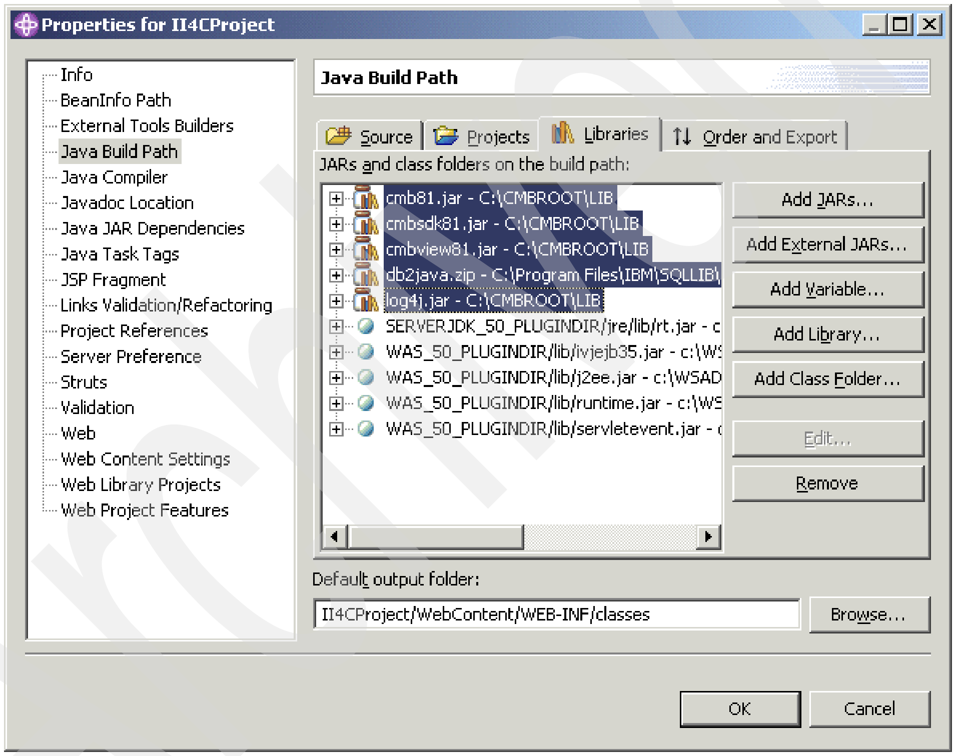
      By default, it is in `C:\CMBROOT`.
   c. Multi-select the following JAR files by holding down the **Ctrl** key on your keyboard:
      - `cmb81.jar`
      - `cmbsdk81.jar`
      - `cmbview81.jar`
      - `log4j.jar`

      Click **Open**.

4. Add the DB2 Java JAR file to the Java build path:
   a. While on the Java Build Settings, Libraries tab, click **Add External JARs...**
   b. Navigate to the directory <DB2_HOME>\java.

      <DB2_HOME> is the location where DB2 is installed.

      By default, it is in `C:\Program Files\IBM\SQLLIB`.
   c. Select db2java.zip file. Click **Open**.
5. The selected JAR files are now added to the build path (Figure 6-7). Click **OK**.


Properties for II4CProject

Info
BeanInfo Path
External Tools Builders
Java Build Path
Java Compiler
Javadoc Location
Java JAR Dependencies
Java Task Tags
JSP Fragment
Links Validation/Refactoring
Project References
Server Preference
Struts
Validation
Web
Web Content Settings
Web Library Projects
Web Project Features

Java Build Path

Source | Projects | Libraries | Order and Export

JARs and class folders on the build path:

```
cmb81.jar - C:\CMBROOT\LIB
cmbsdk81.jar - C:\CMBROOT\LIB
cmbview81.jar - C:\CMBROOT\LIB
db2java.zip - C:\Program Files\IBM\SQLLIB\
log4j.jar - C:\CMBROOT\LIB
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - c
WAS_50_PLUGINDIR/lib/ivjejb35.jar - c:\WS
WAS_50_PLUGINDIR/lib/j2ee.jar - c:\WSAD
WAS_50_PLUGINDIR/lib/runtime.jar - c:\WS
WAS_50_PLUGINDIR/lib/servletevent.jar - c
```

Default output folder: II4CProject/WebContent/WEB-INF/classes

Figure 6-7 Set up Java build path - With the necessary JAR files included

6.2.3 Setting up a new server project to run the sample code

To test the application within the application development environment, we create and configure a new server project.

There are several steps that we need to perform to accomplish this:

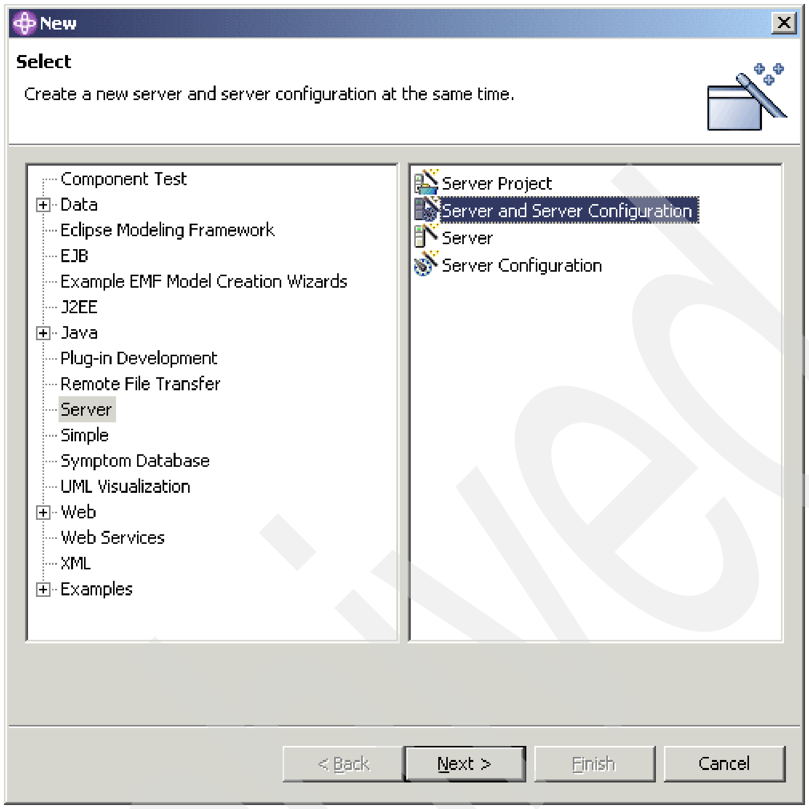
- Create a new server project. We use II4C_Server as our server project name.
- Add the JAR files for the runtime environment. We include the following in the libraries:
  - From the <CMBROOT> directory, cmb81.jar, cmbsdk81.jar, cmbview81.jar, log4j.jar, and xerces.jar.
  - From the <DB2_HOME>\java directory, db2java.zip file.
- Add the folder where cmbicmenv.properties is to the class path.
- Add II4C_EAR to the server configuration.

In the following section, we provide the step-by-step procedures in setting up the new server project.

Create a new server project

To create a new server project, do the following steps:

1. Within WebSphere Studio Application Developer, select **File** → **New** → **Other**.
2. Select **Server** from the left panel and select **Server and Server Configuration** (Figure 6-8) from the right panel. Click **Next**.



Select

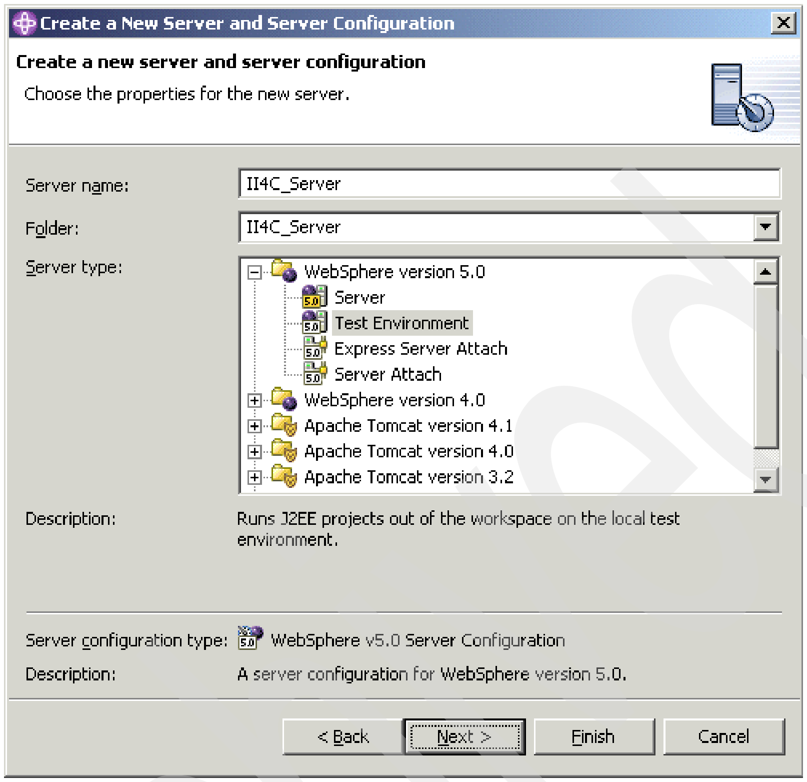
Create a new server and server configuration at the same time.

Figure 6-8 Create new server project - Initial

3. Enter `II4C_Server` for both Server name and Folder fields (Figure 6-9).

   For Server type field, expand **WebSphere version 5.0**, and select **Test Environment**.

   Click **Next**.


Create a New Server and Server Configuration

Create a new server and server configuration

Choose the properties for the new server.

Server name: II4C_Server

Folder: II4C_Server

Server type:
- WebSphere version 5.0
  - Server
  - Test Environment
  - Express Server Attach
  - Server Attach
- WebSphere version 4.0
- Apache Tomcat version 4.1
- Apache Tomcat version 4.0
- Apache Tomcat version 3.2

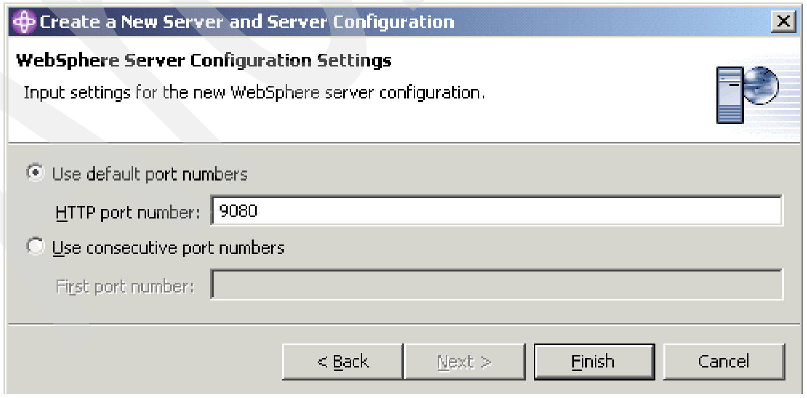
Description: Runs J2EE projects out of the workspace on the local test environment.

Server configuration type: WebSphere v5.0 Server Configuration

Description: A server configuration for WebSphere version 5.0.

Figure 6-9 Create new server project - Set up server name and folder name

4. Enter 9080 for Use default port numbers field (Figure 6-10). Click **Finish**.


Create a New Server and Server Configuration

WebSphere Server Configuration Settings

Input settings for the new WebSphere server configuration.

(●) Use default port numbers

HTTP port number: 9080

( ) Use consecutive port numbers

First port number:

Figure 6-10 Create new server project - Set up default port numbers

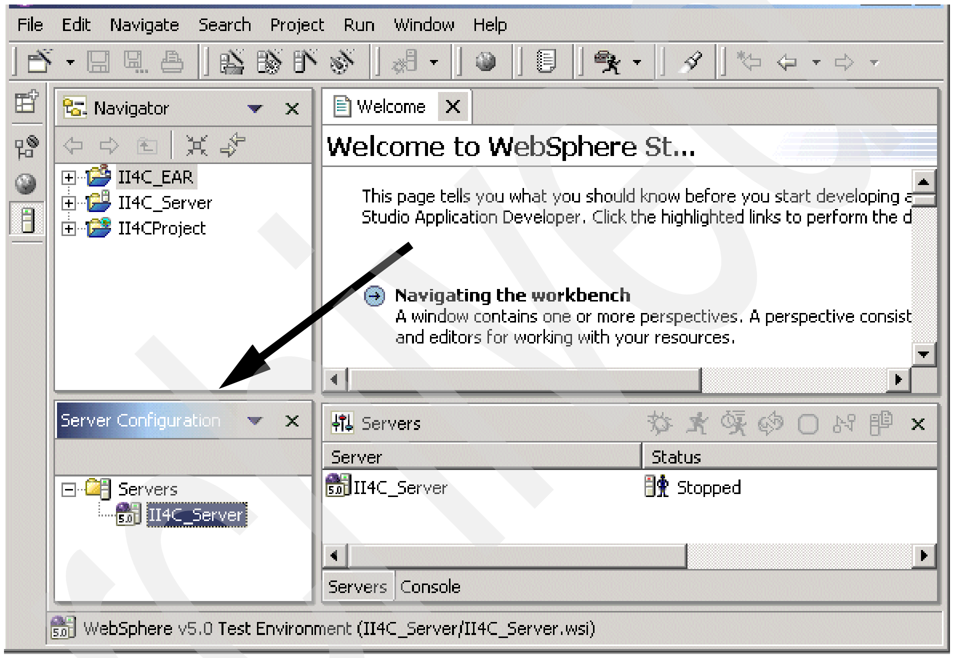
If prompted regarding whether you want to create a new server project with the name `II4C_Server`, click **Yes**.

Add the JAR files for the runtime environment

To add the JAR files for runtime environment, do the following steps:

1. From the file menu, select **Window** → **Open Perspective** → **Server**.

   In the lower left of the window, you should see Server Configuration panel (Figure 6-11).



This page tells you what you should know before you start developing a Studio Application Developer. Click the highlighted links to perform the d

Navigating the workbench
A window contains one or more perspectives. A perspective consist and editors for working with your resources.

Server	Status
II4C_Server	Stopped

WebSphere v5.0 Test Environment (II4C_Server/II4C_Server.wsi)

Figure 6-11 Server configuration

2. Expand the **Servers** folder, right-click **II4C_Server** and select **Open** from the context menu.
3. Click the **Environment** tab from the right panel (see Figure 6-12).
4. Add Information Integrator for Content JAR files to the Java build path:
   a. Click **Add External JARs...**
   b. Navigate to the directory <CMBROOT>\lib.

      <CMBROOT> is the location where Information Integrator for Content is installed.

      By default, it is in `C:\CMBROOT`.

c. Multi-select the following JAR files by holding down the **Ctrl** key on your keyboard:

- `cmb81.jar`
- `cmbsdk81.jar`
- `cmbview81.jar`
- `log4j.jar`
- xerces.jar

Click **Open**.

5. Add the DB2 Java JAR file to the Java build path:

a. While on the Java Build Settings, Libraries tab, click **Add External JARs...**

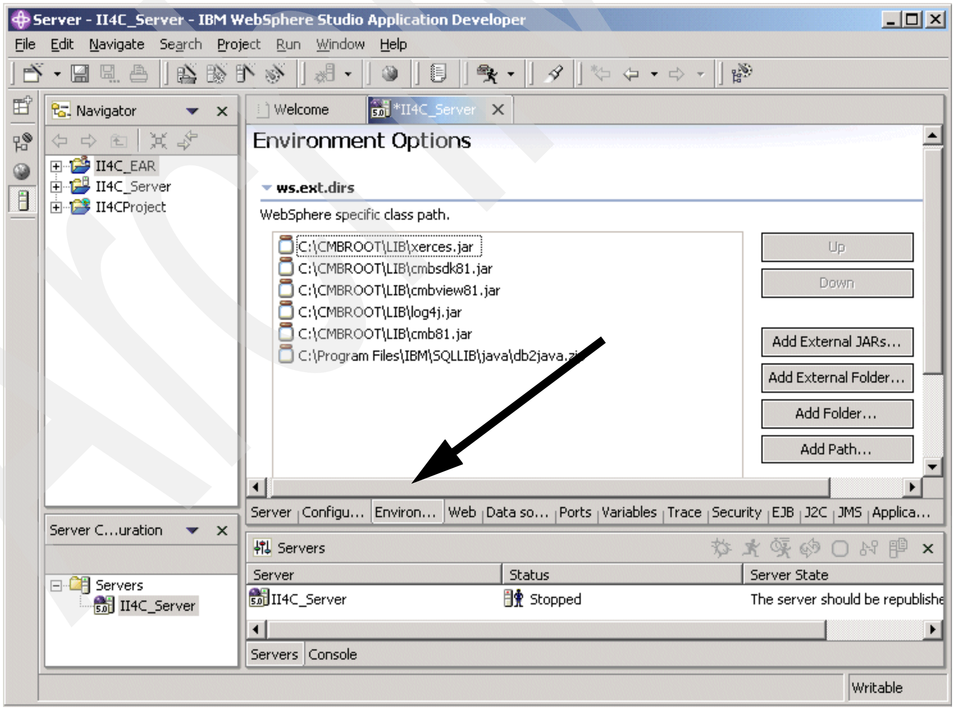
b. Navigate to the directory <DB2_HOME>\java.

<DB2_HOME> is the location where DB2 is installed.

By default, it is in `C:\Program Files\IBM\SQLLIB`.

c. Select db2java.zip file. Click **Open**.

6. The selected JAR files are added to the build path (Figure 6-12). Click **OK**.


Server - II4C_Server - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Run Window Help
Navigator
II4C_EAR
II4C_Server
II4CProject
Welcome
*II4C_Server
Server C...uration
Servers
II4C_Server
Up
Down
Add External JARs...
Add External Folder...
Add Folder...
Add Path...
Server Configu... Environ... Web Data so... Ports Variables Trace Security EJB J2C JMS Applica...
Servers Console
Writable

Environment Options

ws.ext.dirs

WebSphere specific class path.

C:\CMBROOT\LIB\xerces.jar
C:\CMBROOT\LIB\cmbsdk81.jar
C:\CMBROOT\LIB\cmbview81.jar
C:\CMBROOT\LIB\log4j.jar
C:\CMBROOT\LIB\cmb81.jar
C:\Program Files\IBM\SQLLIB\java\db2java.zi

Server	Status	Server State
II4C_Server	Stopped	The server should be republishe

Figure 6-12 Server configuration - with the necessary JAR files included

7. Going back to the Environment tab of the Server Configuration, select **File** → **Save** or press **Ctrl S** to save the changes.
8. If prompted with Do you want to load the changes, click **Yes** to confirm.

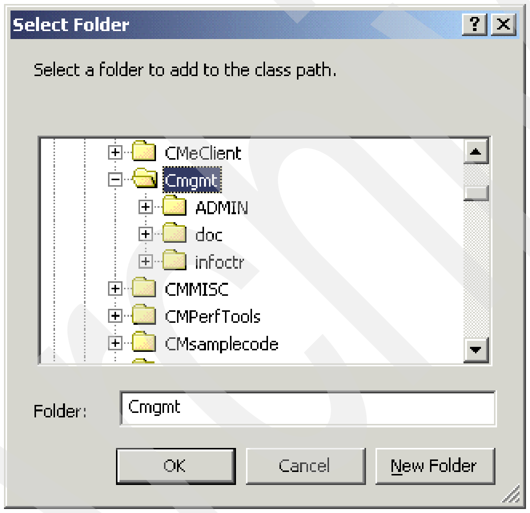
Add the folder where cmbicmenv.properties is to the class path

We need to set the folder that contains the `cmbicmenv.properties` file in the class path. To add this folder to the class path, do the following steps:

1. Within the Environment tab of the Server Configuration, click **Add External Folder...**
2. Navigate to the <CMGMT_DIR> folder.

   <CMGMT_DIR> is the location where the common files of Content Manager and Information Integrator for Content are installed.

   By default, it is in `C:\Cmgmt`.
3. Select the folder (Figure 6-13) and click **OK**.



Folder: Cmgmt

Figure 6-13 Server configuration - Select the property file folder to the class path

4. Select **File** → **Save** or press **Ctrl S** to save the changes to the server instances.
5. Select **File** → **Close** or click **X** on the **II4C_Server** tab to close the panel.

Add II4C_EAR to the server configuration

We need to add II4C_EAR to the server configuration as follows:

1. In Server Configuration panel, expand **Servers** folder if it is not already expanded, and right-click **II4C_Server**.

- Select **Add or remove projects....** Select **II4C_EAR** from the Available projects, and click **Add** (Figure 6-14).

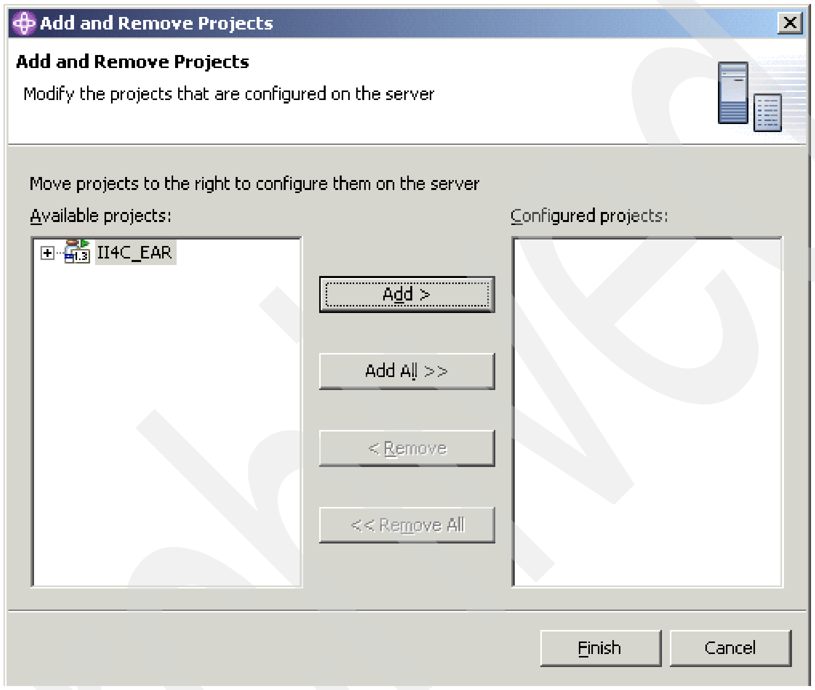


Figure 6-14 Server configuration - Add the project EAR file

6.3 Developing sample code with the Java OOAPIs

The development environment is now ready. In this section, we experiment with some sample code using Information Integrator for Content Java OOAPIs. We focus on three particular content servers: the relational database DB2, Content Manager, and OnDemand servers.

In order to illustrate the use of the Information Integrator for Content APIs, we work with a simple HTML page, a simple controller servlet (acting mainly as a control switch between JSP pages), and a number of JSP pages, each performing a specific set of functionality of the mentioned content servers.

6.3.1 Adding the common files to the Web project

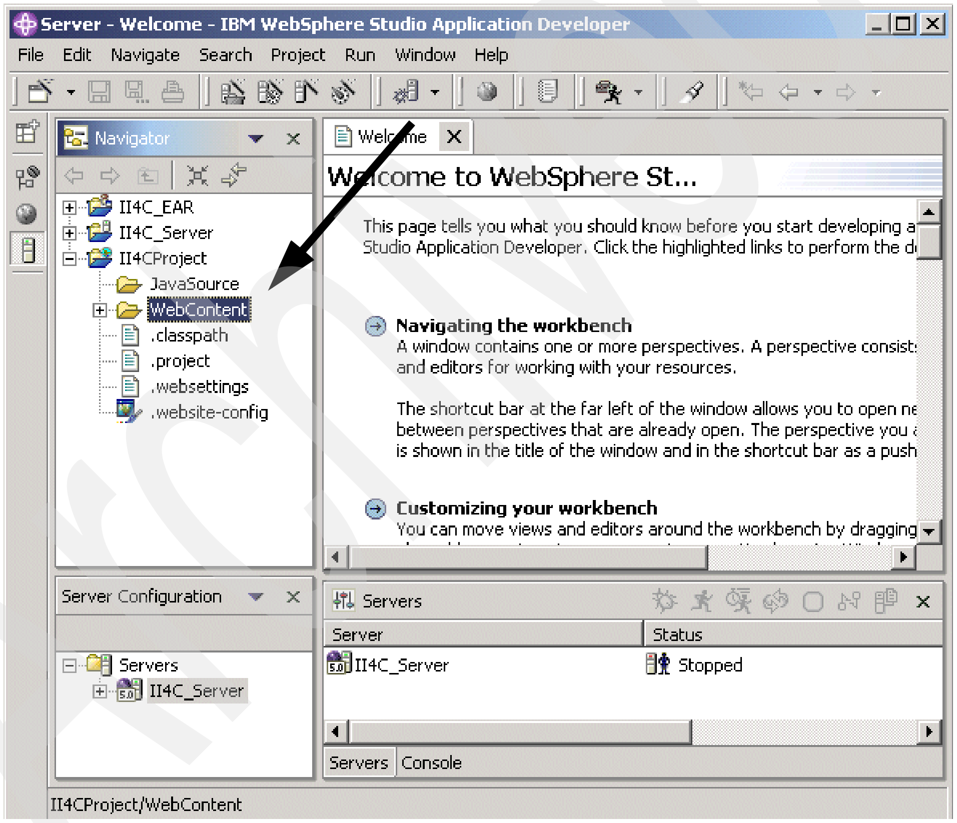
We start by adding some common files for the Web project:

- Creating an HTML file, `index.html`
- Adding static Web resources: images, theme
- A servlet, `ConnectionController.class`

Creating an HTML file, index.html

To add a new HTML page to a Web project:

1. From the Navigator panel, expand **II4CProject**, and select **WebContent** (Figure 6-15).



Welcome to WebSphere St...

This page tells you what you should know before you start developing a
Studio Application Developer. Click the highlighted links to perform the d

Navigating the workbench
A window contains one or more perspectives. A perspective consist
and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open ne
between perspectives that are already open. The perspective you a
is shown in the title of the window and in the shortcut bar as a push

Customizing your workbench
You can move views and editors around the workbench by dragging

Figure 6-15 Sample code - WebContent

2. Right-click WebContent, and select **New** → **Other...** from the context menu.
3. Select **Web** → **HTML/XHTML File** (Figure 6-16). Click **Next**.

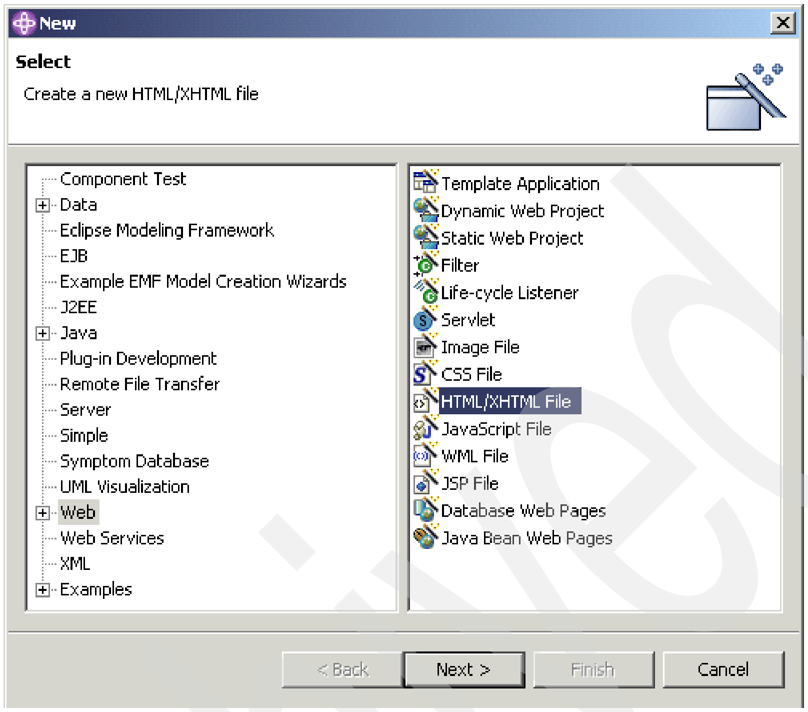
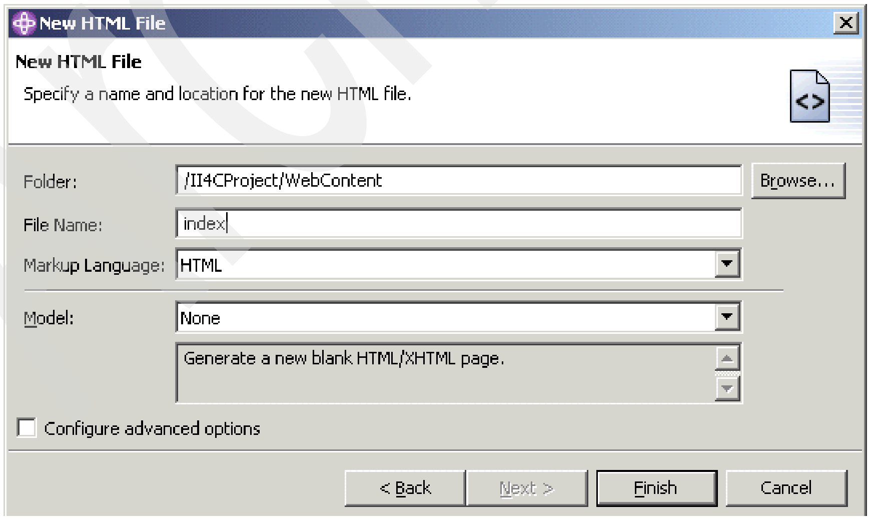


Figure 6-16 Sample code - Add HTML file

4. Enter `index` in the File Name field (Figure 6-17). Click **Finish**.


New HTML File
New HTML File
Specify a name and location for the new HTML file.
Browse...
Configure advanced options
< Back
Next >
Finish
Cancel

Folder: /II4CProject/WebContent
File Name: index
Markup Language: HTML
Model: None
Generate a new blank HTML/XHTML page.

Figure 6-17 Sample code - Set HTML file name

5. Click the **Source** tab, then copy and paste the sample code shown in Example 6-1 into the newly created HTML page. Alternatively, you can import the index.html from the additional material provided by this redbook.

   The HTML file shown contains a form with:

   – One field of type radio button, Content_Server
   – One submit button

   The HTML form submits the request to a servlet, ConnectionController.

Example 6-1 The project Welcome page, index.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>index.html</TITLE>
</HEAD>
<BODY>
<H1>Welcome to the redbrook county site</H1>
<HR>
<!--Form-->
<FORM action="servlet/ConnectionController">
<INPUT type="radio" name="Content_Server" value="DB2" checked>Relational
Database DB2<BR>
<INPUT type="radio" name="Content_Server" value="CM" >Content Manager<BR>
<INPUT type="radio" name="Content_Server" value="OD" >OnDemand Server<BR>
<BR>
<INPUT type="submit" name="submit" value="submit">
</FORM>
<!--Form-->
<HR>
<BR>
<IMG src="images/poweredby_WebSphere.gif" width="116" height="52"
border="0"></P>
</BODY>
</HTML>
```

6. To see what the HTML file produces, click the **Preview** tab (Figure 6-18).

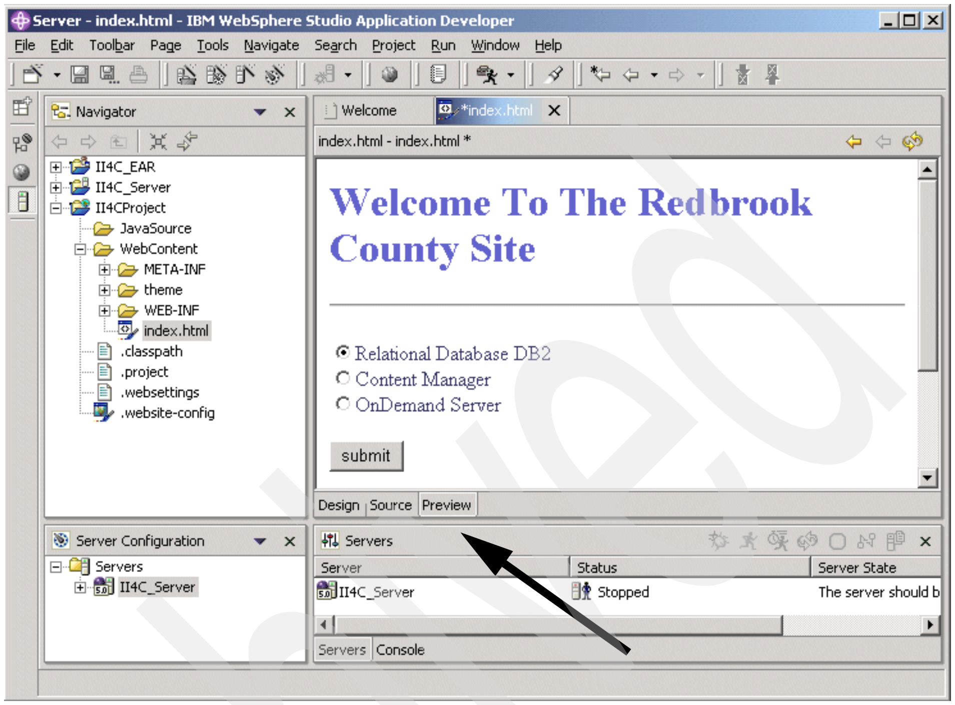


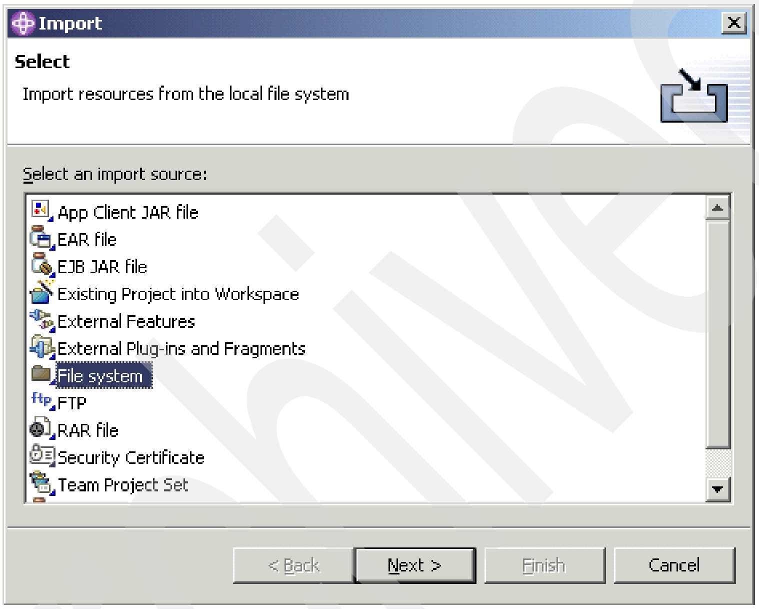
Figure 6-18 Sample code - HTML page preview

Adding static Web resources: images, theme

We want to static Web resources to the project. To do this, we must import two folders to the WebContent of the project. These folders and their content are in the additional material provided by this redbook.

To add static Web resources to the Web project:

1. Right-click **WebContent** and select **Import...** from the context menu.
2. From the Import dialog, select **File system** (Figure 6-19). Click **Next**.


Import

Select

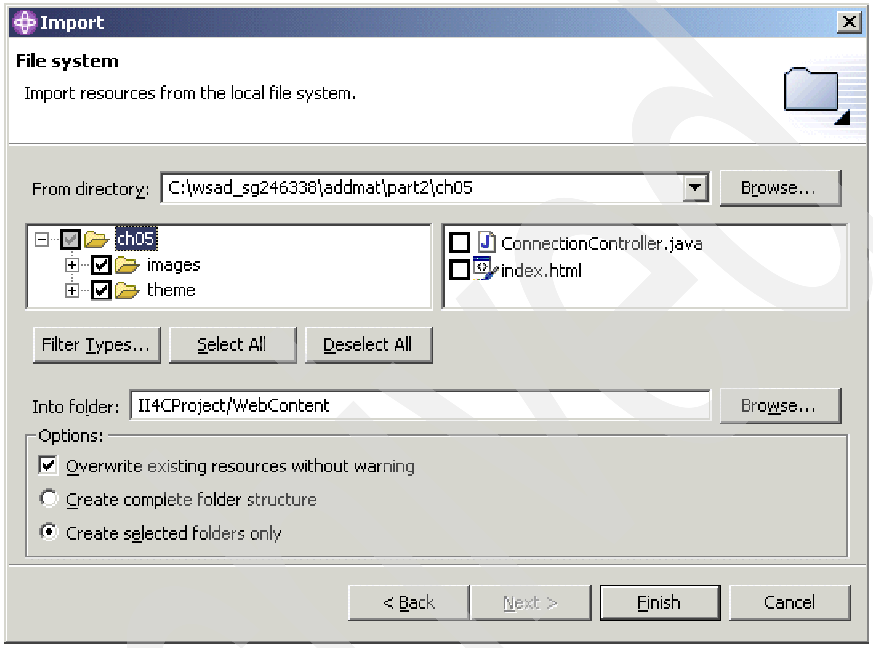
Import resources from the local file system

Select an import source:

App Client JAR file
EAR file
EJB JAR file
Existing Project into Workspace
External Features
External Plug-ins and Fragments
File system
FTP
RAR file
Security Certificate
Team Project Set

Figure 6-19 Sample code - Import resources from file system

3. Browse to the location where the two folders, images and theme, are located. Select the parent folder, and click **OK**.
4. Expand the folder. Check both **images** and **theme** folders. Check Overwrite existing resources without warning. Make sure Create selected folders only is checked (Figure 6-20). Click **Finish**.


Import

File system

Import resources from the local file system.

From directory: C:\wsad_sg246338\addmat\part2\ch05

- [✓] ch05
  - [✓] images
  - [✓] theme

- [ ] ConnectionController.java
- [ ] index.html

Into folder: II4CProject/WebContent

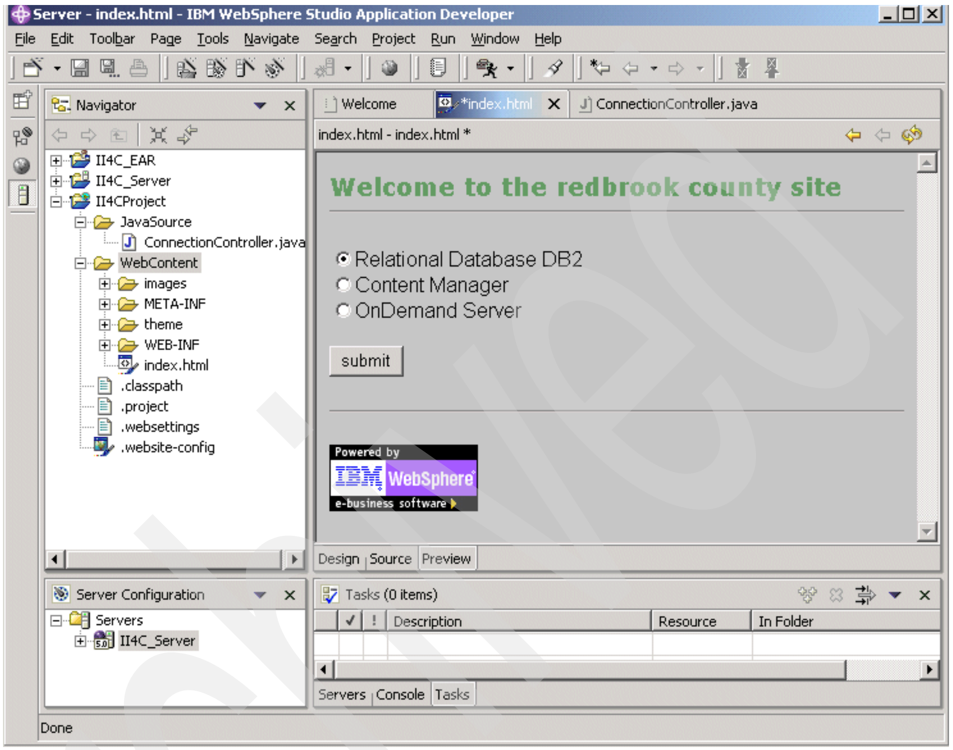
Options:

- [✓] Overwrite existing resources without warning
- ○ Create complete folder structure
- ● Create selected folders only

Figure 6-20 Sample code - Import images and theme resources

5. With the new image (poweredby_WebSphere.gif) and new theme resources, the look of the index.html is changed (Figure 6-21).

   To see the new look, select **index.html** under WebContent; click **Preview**.



Welcome to the redbrook county site

[✓ Relational Database DB2]
Content Manager
OnDemand Server

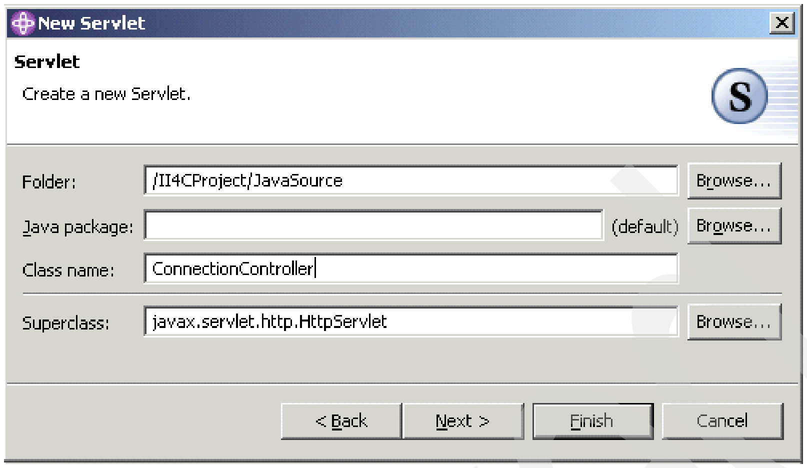
Figure 6-21 Sample code - index.html preview with new images and theme resources

Adding the control servlet

To open the add servlet:

1. From the Navigator panel, select **II4CProject**.
2. Select **File** → **New** → **Other...**
3. Select **Web** → **Servlet**, and click **Next**.
4. Enter `ConnectionController` in the Class name field.

   Leave Java package blank (Figure 6-22). Click **Finish**.


New Servlet

Servlet

Create a new Servlet.

Folder: /II4CProject/JavaSource

Java package: (default)

Class name: ConnectionController

Superclass: javax.servlet.http.HttpServlet

Figure 6-22 Sample code - Add new control servlet

The Control servlet is used to direct the user's request to the proper JSP.

It does the following actions:

a. It gets the action from the user request:

```
String action = request.getParameter("Content_Server");
```

b. Depending on this action, it calls various JSP files. For example, if user selected Relational Database DB2, which translates to a value of DB2 for Content_Server, then, the response JSP that needs to be called is connect.jsp.

```
if (action.equalsIgnoreCase("DB2") ){
    responseJSP="connect.jsp";
}
```

5. Copy and paste the code shown in Example 6-2, into your Web project. The servlet code is also found in the redbook additional material folder under part2\ch05\ConnectionController.java. You can optionally import the source code directly instead of cut and paste.

Example 6-2 Project controller servlet

```
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.mm.beans.*;
import java.io.*;
import java.lang.*;
import java.lang.String.*;
```

```
public class ConnectionController extends HttpServlet implements CMBBaseConstant {
  public void init(ServletConfig config) throws ServletException {
    //Call the parents init function. Should be done in a servlet
    super.init(config);
  }
  public synchronized void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
  public synchronized void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
     try {
      String responseJSP = null;
      String action = request.getParameter("Content_Server");
             if (action==null){
                responseJSP="error.jsp";
             }

             if (action.equalsIgnoreCase("DB2") ){
                responseJSP="connect.jsp";
             }

             if (action.equals("OD")){
                responseJSP="connectOD.jsp";
             }

             if (action.equals("CM")){
                responseJSP="connectCM.jsp";
             }

             // Call the presentation renderer
         getServletContext().getRequestDispatcher(responseJSP).forward(request,response);

      } //end try
    //-------------------------------------------------------
    catch (Exception exc) {
      exc.printStackTrace();
       } catch (Throwable exc) {
          exc.printStackTrace();
      }
  }
}
```

Once we have added and created the common files needed for the Web project, we can create attributes, item types, and items, as described in the following sections.

6.3.2 Working with attributes

Part of the Content Manager System Administrator's task is to create the necessary data model for the Content Manager application. This can be done manually, or it can be done with a program. The advantage of doing it via a program is that it will be easier to change and port to another system.

In this section, we show you how you can create the data model for the business case we use in this redbook.

For simplicity, we set all of the attributes to variable character type. The length of the attribute depends on the data to be held in the attribute. If it is an identifier, the length is limited to the number of digits that represent it. If it is a description field, the length should be set to the maximum possible number of characters that are to be stored.

For example, a document description attribute, BP_DOC_DESC, might be of variable length. For simplicity, we assume it never exceeds 200 characters. We therefore set the maximum size to 200. The social security number, on the other hand, is generally in the `###-##-####` format. We can strip the two dashes (-) in the middle and specify the maximum size to be 9. For simplicity of the sample code, we decide to use 11 characters long, without doing additional conversion. Table 6-1 shows the information for both of these two attributes.

Table 6-1 Sample attribute information for BP_DOC_DESC and BP_TAX_ID

Attribute name	BP_DOC_DESC	BP_TAX_ID
Attribute description	Document description	Social security number
Attribute type	Variable character type	Variable character type
Maximum size	200	11

Basics of creating and deleting attributes

In order to create new attributes, we have to connect to Content Server. This requires two steps:

1. Create a datastore object.

   ```
   DKDatastoreICM dsICM = new DKDatastoreICM();
   ```

2. Connect to the datastore using the function.

   ```
   dsICM.connect("LSDB","LSADMIN","lspasswd","");
   ```

   To connect to the datastore, you must supply the Library Server database name, the user ID, and the password.

Note: We hard-coded these values for simplicity of the sample code. Modify these values if your information is different than our development environment. In later chapters, we show you how to get input from the user to use for log in purposes, and alternatively, how to log in via a configuration file.

After connecting the datastore, we can create attributes as follow:

1. Create a new Attribute Definition object:

   ```
   DKAttrDefICM attr = new DKAttrDefICM(dsICM);
   ```

2. Set the attribute's name, description, type, and size:

   ```
   attr.setName("BP_DOC_DESC");
   attr.setDescription("Document Description ");
   attr.setType(DKConstant.DK_CM_VARCHAR);
   attr.setSize(200);
   ```

3. Add the new definition to the persistent datastore:

   ```
   attr.add();
   ```

We can also delete attributes as follows:

1. Get the datastore definition objects from the connected datastore:

   ```
   DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
   ```

2. Retrieve the attribute definitions:

   ```
   DKAttrDefICM attrDef = (DKAttrDefICM)
   ```

3. Retrieve specific attribute:

   ```
   dsDefICM.retrieveAttr("BP_CODEBOOK_ID");
   ```

4. If the attribute exists, delete the attribute:

   ```
   if(attrDef!=null)
      attrDef.del(); // Delete the attribute definition
   ```

At the end, we need to close the connection and destroy the datastore:

1. Disconnect from your datastore object:

   ```
   dsICM.disconnect();
   ```

2. Destroy the object:

Example 6-3 shows a code snippet of the steps required to create one attribute.

Example 6-3 Steps required to create an attribute

```
<%
// Create new datastore object.
   DKDatastoreICM dsICM = new DKDatastoreICM();
// Connect to the datastore.
   dsICM.connect("LSDB","LSADMIN","lspasswd","");
// Create an attribute definition Object
   DKAttrDefICM attr = new DKAttrDefICM(dsICM);
// set the attribute name
   attr.setName("BP_DOC_DESC");
// Give it a description
   attr.setDescription("Document Description ");
// Set the data type that this attribute should hold
   attr.setType(DKConstant.DK_CM_VARCHAR);
// Set its maximum size
   attr.setSize(200);
// Add the attribute to the datastore
   attr.add();
// Important step
dsICM.disconnect();
dsICM.destroy();
%>
```

Note: The attribute type is set using `attr.setType(DKConstant.DK_CM_VARCHAR)`. This method has a constant as an argument. Most Information Integrator for Content constants are found in the class DKConstant. Additional constants are found in DKConstant of a specific datastore (ICM, DB2, OD).

Creating the attribute definition code

When editing and testing attribute definition code, do the following steps:

1. Create a new JSP file:
   a. Under II4CProject, right-click **WebContent**.
   b. Select **New** → **Other** from the context menu.
   c. Select **Web** → **JSP File** (Figure 6-23). Click **Next**.

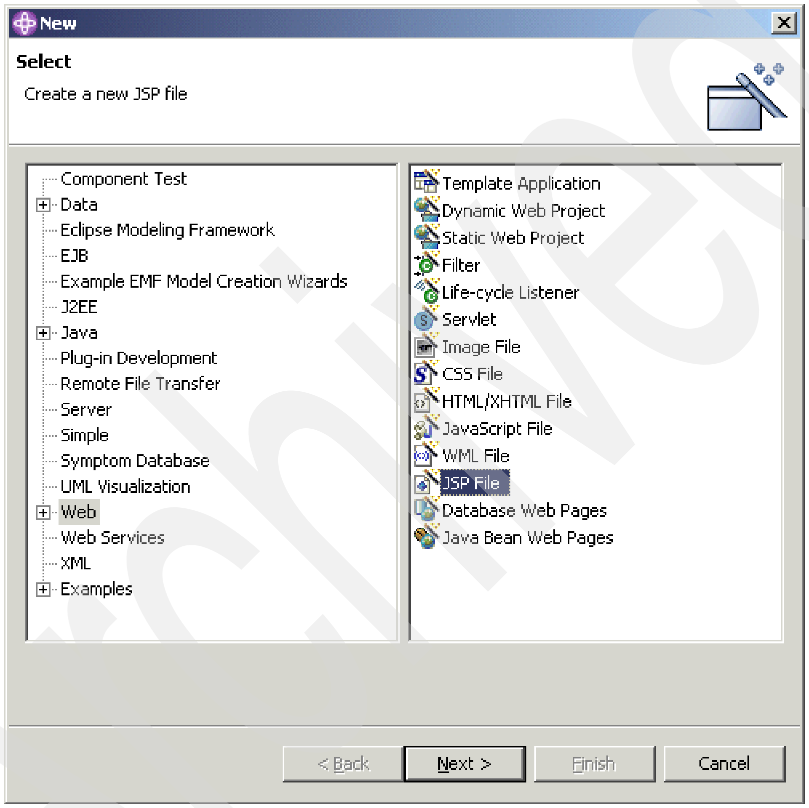
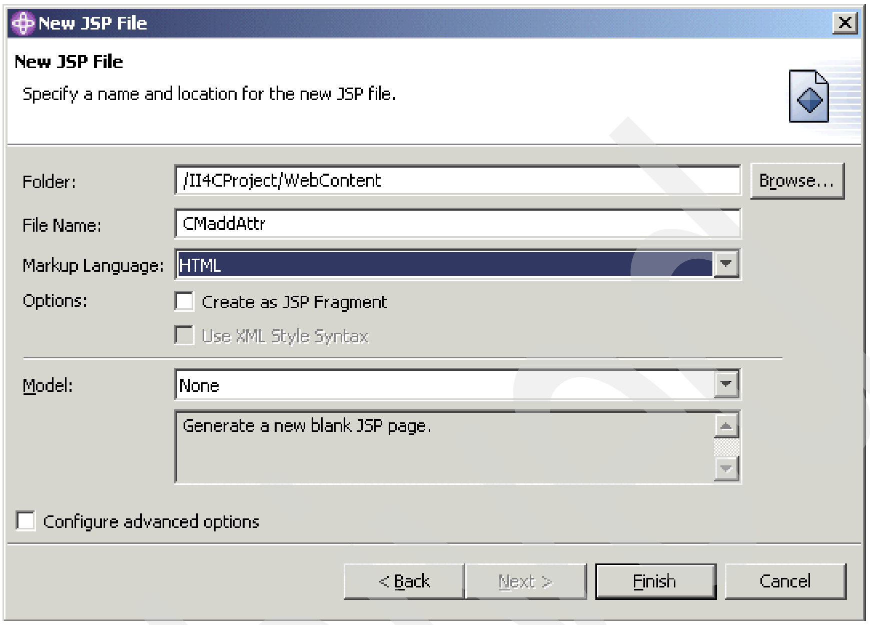


Figure 6-23 Sample code - Create new JSP file

2. Enter `CMaddAttr` in the File Name field (Figure 6-24). Click **Finish**.



Folder: /II4CProject/WebContent
File Name: CMaddAttr
Markup Language: HTML
Model: None
Generate a new blank JSP page.

Figure 6-24 Sample code - Enter new JSP file name

3. An editor with the newly created CMaddAttr.jsp file appears. Click the **Source** tab.

   The `<TITLE>` tag is filled with the file name by default.

4. Search for `pageEncoding="ISO-8859-1"` text. Add the following import statement after this line.

```
import="com.ibm.mm.sdk.server.*, com.ibm.mm.sdk.common.*"
```

> **Note:** These two packages com.ibm.mm.sdk.server.* and com.ibm.mm.sdk.common.* contain all the Information Integrator for Content classes that we use in the sample code. You must include them in *all* your JSP pages if you want to test each code separately.

5. Now copy the code in Example 6-4 before the closing `</BODY>` tag.

   This sample code creates all the required attributes for our data model.

Example 6-4 CMaddAttr.jsp - Create attributes definition sample code

```
<%
DKDatastoreICM dsICM = new DKDatastoreICM();  // Create new datastore object.
dsICM.connect("LSDB","LSADMIN","lspasswd",""); // Connect to the datastore.

   DKAttrDefICM attr = new DKAttrDefICM(dsICM);
   attr.setName("BP_DOC_DESC");
   attr.setDescription("BP_DOC_DESC");
   attr.setType(DKConstant.DK_CM_VARCHAR);
   attr.setSize(200);
   attr.add();
   %>
   Attribute <%=attr.getName()%> has been added <br>
   <%
   attr = new DKAttrDefICM(dsICM);
   attr.setName("BP_CODEBOOK_SUB");
   attr.setDescription("BP_CODEBOOK_SUBJECT");
   attr.setType(DKConstant.DK_CM_VARCHAR);
   attr.setSize(200);
   attr.add();
   %>
   Attribute <%=attr.getName()%> has been added <br>
   <%
   attr = new DKAttrDefICM(dsICM);
   attr.setName("BP_DOC_VERSION");
   attr.setDescription("BP_DOC_VERSION");
   attr.setType(DKConstant.DK_CM_VARCHAR);
   attr.setSize(20);
   attr.add();
   %>
   Attribute <%=attr.getName()%> has been added <br>
   <%
   attr = new DKAttrDefICM(dsICM);
   attr.setName("BP_SUB_DATE");
   attr.setDescription("BP_SUBMISSION_DATE");
   attr.setType(DKConstant.DK_CM_DATE);
   attr.add();
   %>
   Attribute <%=attr.getName()%> has been added <br>
   <%
   attr = new DKAttrDefICM(dsICM);
   attr.setName("BP_NUMBER");
   attr.setDescription("BP_NUMBER");
   attr.setType(DKConstant.DK_CM_VARCHAR);
   attr.setSize(12);
   attr.add();
   %>
   Attribute <%=attr.getName()%> has been added <br>
```

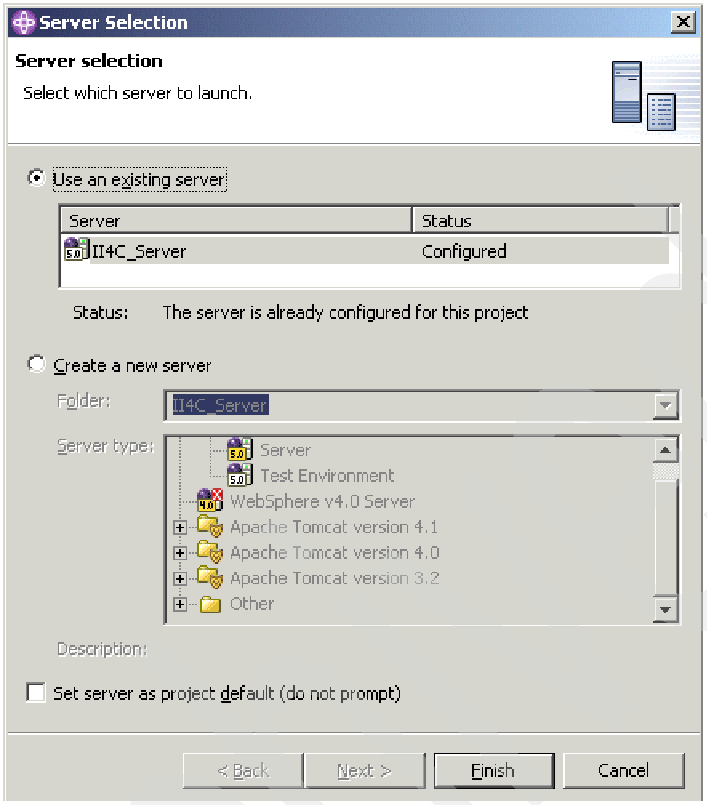
```
    <%
    attr = new DKAttrDefICM(dsICM);
    attr.setName("BP_TAX_ID");
    attr.setDescription("BP_TAX_ID");
    attr.setType(DKConstant.DK_CM_VARCHAR);
    attr.setSize(12);
    attr.add();
    %>
    Attribute <%=attr.getName()%> has been added <br>
    <%
    attr = new DKAttrDefICM(dsICM);
    attr.setName("BP_CODEBOOK_ID");
    attr.setDescription("BP_CODEBOOK_ID");
    attr.setType(DKConstant.DK_CM_VARCHAR);
    attr.setSize(12);
    attr.add();
    %>
     Attribute <%=attr.getName()%> has been added <br>
     <%

dsICM.disconnect();
dsICM.destroy();
%>
```

6. Save the file, by selecting **File** → **Save** or **Ctrl S**.

Running the attribute definition code

Now you have the code for creating attributes in your CMaddAttr.jsp file. You need to run the code:

1. Under II4CProject, WebContent folder, right-click **CMaddAttr.jsp**.
2. Select **Run on Server...**
3. If prompted to save resources, click **OK**.
4. Make sure **II4CServer** is selected under Use an existing server part (Figure 6-25). Click **Finish**.
5. Figure 6-26 displays the results of running the sample code.



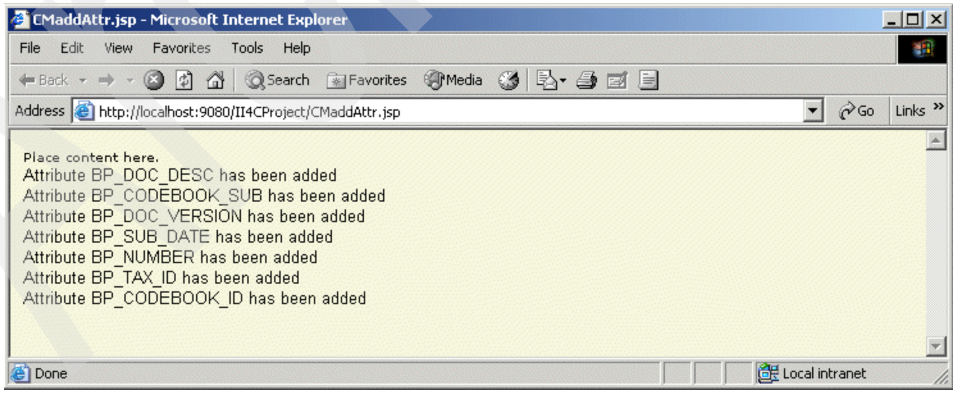
Server	Status
II4C_Server	Configured

Status: The server is already configured for this project

Folder: II4C_Server

Server type:
- Server
- Test Environment
- WebSphere v4.0 Server
- Apache Tomcat version 4.1
- Apache Tomcat version 4.0
- Apache Tomcat version 3.2
- Other

Figure 6-25 Test sample code - Select server to run on


CMaddAttr.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links
Done
Local intranet

Address: http://localhost:9080/II4CProject/CMaddAttr.jsp

```
Place content here.
Attribute BP_DOC_DESC has been added
Attribute BP_CODEBOOK_SUB has been added
Attribute BP_DOC_VERSION has been added
Attribute BP_SUB_DATE has been added
Attribute BP_NUMBER has been added
Attribute BP_TAX_ID has been added
Attribute BP_CODEBOOK_ID has been added
```

Figure 6-26 Result of running the CMaddAttr.jsp in the browser

6. To test that attributes were properly created, we must go to the Content Manager System Administration Client to test this function.

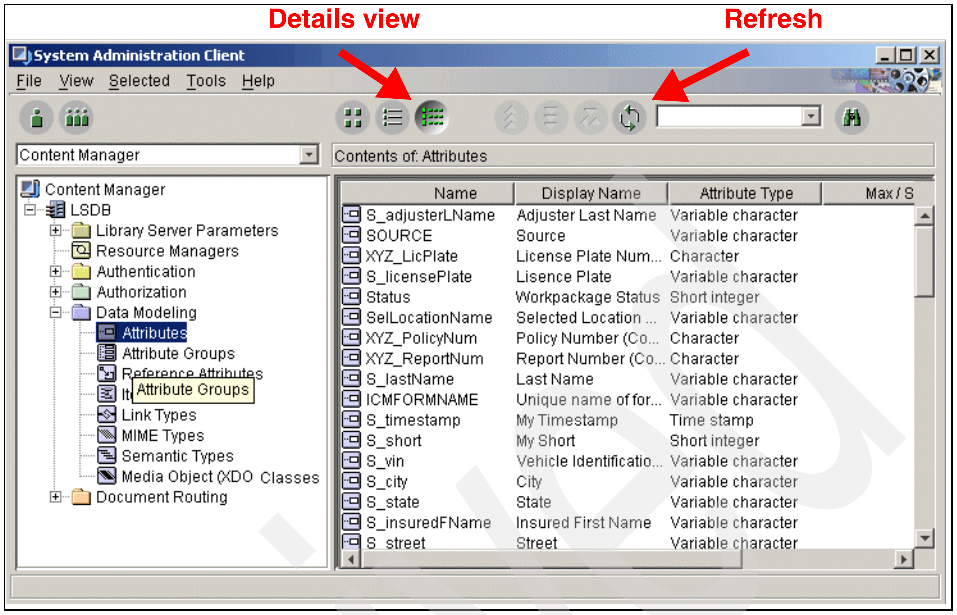
Attention: If at this step, you failed, go back to check your Library Server database name, user ID, and password in CMaddAttr.jsp. It must match the values in your environment.

Testing the code

When the code runs smoothly without any errors, you need to test that the attributes were properly created. To do this, we use Content Manager System Administration Client.

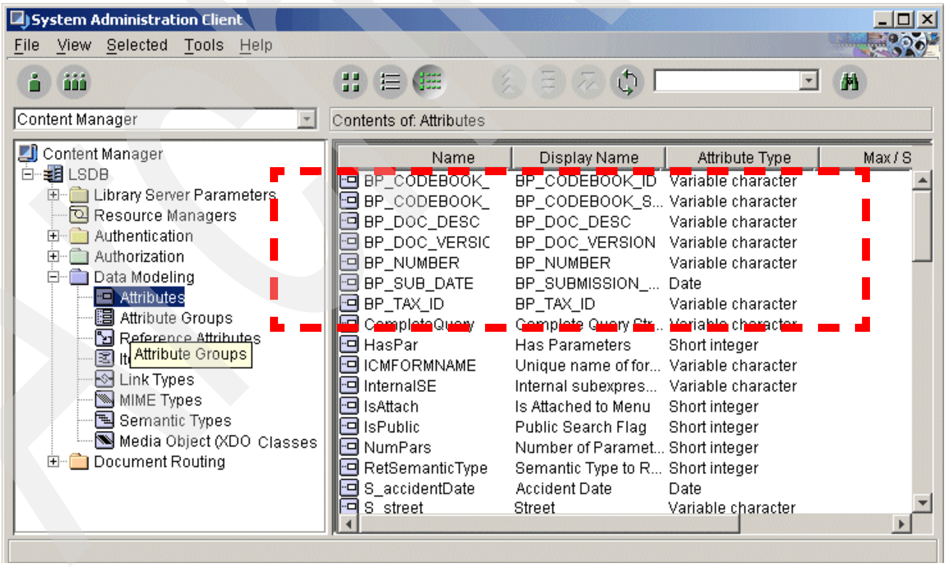
1. Start Content Manager System Administration Client by selecting **Programs** → **IBM Content Manager for MultiPlatforms V8.2** → **System Administration Client**.
2. Enter the Library Server User ID and password, then click **OK**.

   We use `LSDB, lsadmin, lspasswd`.
3. Once log in, expand your Library Server database.
4. Select **Data Modeling** →**Attributes**.
5. It is better to select the Details view in order to display attributes in alphabetic order. All our attributes names from the sample code start with the postfix “BP_” so you should be able to see them on top of the list:
   a. Click the **Details view** icon. Click **Refresh** (see Figure 6-27).
   b. Click the **Name** column to sort attributes by their names (see Figure 6-28).



Name	Display Name	Attribute Type	Max / S
S_adjusterLName	Adjuster Last Name	Variable character	
SOURCE	Source	Variable character	
XYZ_LicPlate	License Plate Num...	Character	
S_licensePlate	Lisence Plate	Variable character	
Status	Workpackage Status	Short integer	
SelLocationName	Selected Location ...	Variable character	
XYZ_PolicyNum	Policy Number (Co...	Character	
XYZ_ReportNum	Report Number (Co...	Character	
S_lastName	Last Name	Variable character	
ICMFORMNAME	Unique name of for...	Variable character	
S_timestamp	My Timestamp	Time stamp	
S_short	My Short	Short integer	
S_vin	Vehicle Identificatio...	Variable character	
S_city	City	Variable character	
S_state	State	Variable character	
S_insuredFName	Insured First Name	Variable character	
S_street	Street	Variable character	

Figure 6-27 System Administration Client - Details view and refresh icons



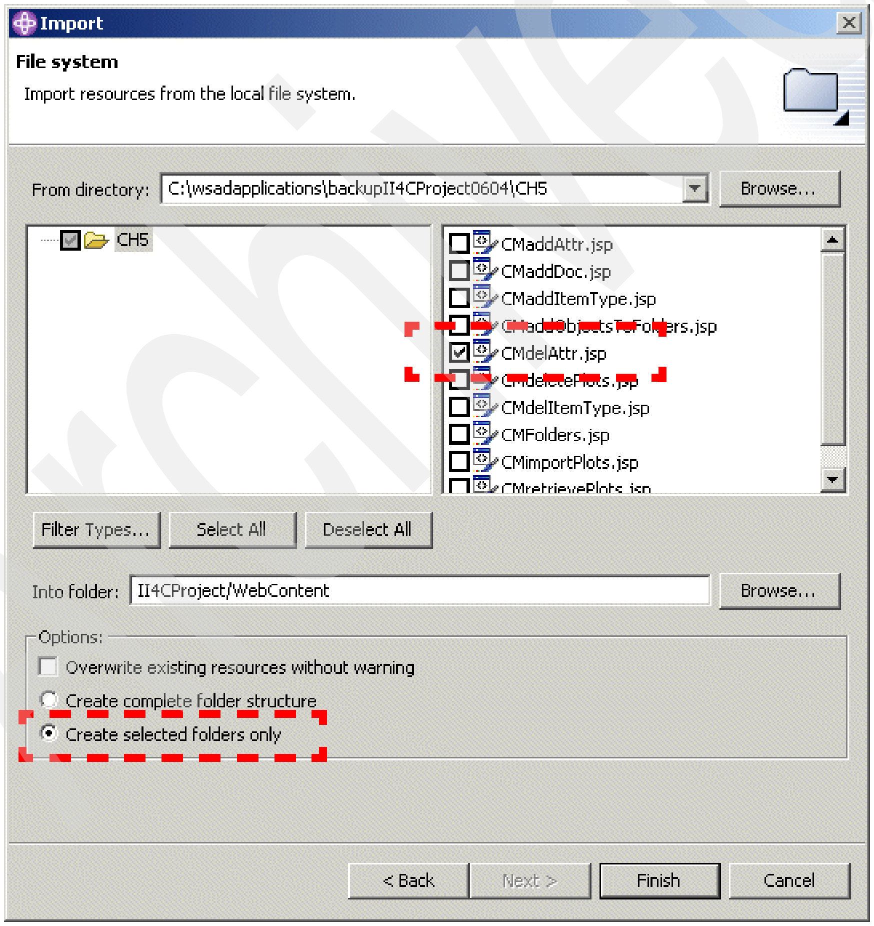
Name	Display Name	Attribute Type	Max / S
BP_CODEBOOK_	BP_CODEBOOK_ID	Variable character	
BP_CODEBOOK_	BP_CODEBOOK_S...	Variable character	
BP_DOC_DESC	BP_DOC_DESC	Variable character	
BP_DOC_VERSIC	BP_DOC_VERSION	Variable character	
BP_NUMBER	BP_NUMBER	Variable character	
BP_SUB_DATE	BP_SUBMISSION_...	Date	
BP_TAX_ID	BP_TAX_ID	Variable character	
CompleteQuery	Complete Query Str...	Variable character	
HasPar	Has Parameters	Short integer	
ICMFORMNAME	Unique name of for...	Variable character	
InternalSE	Internal subexpres...	Variable character	
IsAttach	Is Attached to Menu	Short integer	
IsPublic	Public Search Flag	Short integer	
NumPars	Number of Paramet...	Short integer	
RetSemanticType	Semantic Type to R...	Short integer	
S_accidentDate	Accident Date	Date	
S_street	Street	Variable character	

Figure 6-28 Newly created BP_* attributes

Re-running the sample code

If you want to run the code again, make sure you delete the attributes first. You can delete the attributes from the System Administration Client manually, or via programming:

1. Import the CMdelAttr.jsp file from the additional material provided:
   a. Under II4CProject, right-click **WebContent**.
   b. Select **Import...** → **File system**. Click **Next**.
   c. Click **Browse** to go to the directory where the additional material is located. Select CH6 folder, then click **OK**.
   d. Check the file **CMdelAttr.jsp** (Figure 6-29). Make sure Create selected folders only is selected. Click **Finish**.



From directory: C:\wsadapplications\backupII4CProject0604\CH5

☑ CH5

☐ CMaddAttr.jsp
☐ CMaddDoc.jsp
☐ CMaddItemType.jsp
☐ CMaddObjectsToFolders.jsp
☑ CMdelAttr.jsp
☐ CMdeletePlots.jsp
☐ CMdelItemType.jsp
☐ CMFolders.jsp
☐ CMimportPlots.jsp
☐ CMretrievePlots.jsp

Into folder: II4CProject/WebContent

Options:
☐ Overwrite existing resources without warning
○ Create complete folder structure
● Create selected folders only

Figure 6-29 Import CMdelAttr.jsp sample code

2. Run CMdelAttr.jsp on server.
   a. Right-click **CMdelAttr.jsp**, and select **Run on Server**.
   b. Take the default setting, and click **Finish**.
   c. Check the results from the System Administration Client for verification.

Note: If it is not running, take a look at the dsICM.connect line in the code. Make sure you change the parameters to accommodate your Library Server database name, user ID, and password!

3. Run CMaddAttr.jsp on server to re-add the attributes back to the system.

6.3.3 Working with item types

After creating the attributes, we now show you how to create the item types. In Content Manager, item type is the grouping of common metadata for a set of objects having common characteristics. It is a key element in Content Manager data modeling. For more information on this topic, see 2.2, “Data modeling” on page 29.

When we create an item type, we add attributes to it. This is why we show you how to create attributes first. There are also design-time properties for the attributes to be added when an item type is created. For example, when we create an attribute, we set its type and size. When we add it to an item type, we set additional properties which are only applicable to that particular item type. An example of the additional properties included are whether this attribute is unique, text searchable, or nullable.

Content Manager allows the creation of hierarchical item types and versioning of items. Since the purpose of this redbook is to get you up and going in developing a Web Content Manager application, we choose not to explore these topics in this redbook; however, you should be aware of the options and learn more about them if your business requires these features and functionalities.

Item type classifications

An item is an instance of an item type. Depending on the item type classification, an item can have certain functionality and specific behavior.

An item type is classified as one of the following possibilities:

- **Item:** A simple item that contains only metadata and does not reference to any resource object. It is represented by DDO.

- **Resource item:** A resource object that references to content and is stored in the Resource Manager. This can be a video, audio, or document. It is represented by XDO.
- **ICM Document or Document Part:** The item follows the ICM Document Model, and can optionally have parts. In this model, the item is constructed of a DDO (metadata) and object (resource object).

  There are several system defined part types:

  – ICMANNOTATION — A LOB resource that holds an annotation. The constant used when creating this part is DK_ICM_PART_ANNOTATION.

  – ICMBASELOB — A resource that holds part of the document model. The constant used when creating this part is DK_ICM_PART_BASE.

  – ICMBASETEXT — A text resource that holds part of the document model. The constant used when creating this part is DK_ICM_PART_BASETEXT.

  – ICMBASESTREAM — A stream resource that holds part of the document model. The constant used when creating this part is DK_ICM_PART_BASESTREAM.

  – ICMNOTELOG — A text resource that holds any notes and comments. The constant used when creating this part is DK_ICM_PART_NOTELOG.

In our data model we create item types that follow the Content Manager Document Model. The process of creating an item type involves the following steps:

1. Create a new DKITemTypeDefICM, passing it a reference to the datastore.
2. Give the item type a name that will be used to create these items of this item type later.
3. Add the desired attributes to the item type, setting qualifiers such as nullable, text searchable, and whether it is unique or not.
4. Add the item type to the persistent datastore.

Additional steps are added accordingly if we need an item type whose classification is not the default item/DDO. This might include the two steps below before adding the data item to the persistent datastore:

1. Specify the item classification.
2. Link the data type to a resource object type, and set properties if required.

Using Redbrook County's building permit process as our case study, we build five item types with their corresponding attributes as shown in Table 6-2.

Table 6-2 Item types and attributes for Redbrook County building permit application

Item type names	Attributes
BP_FOLDER	BP_NUMBER BP_TAX_ID BP_STATUS BP_REJ_DESC
BP_APPLICATION	BP_NUMBER BP_DOC_DESC BP_SUB_DATE
BP_DRAWING	BP_NUMBER BP_DOC_DESC BP_SUB_DATE BP_DOC_VERSION
BP_PLOT	BP_NUMBER BP_DOC_DESC BP_SUB_DATE BP_DOC_VERSION
BP_CODE_BOOK	BP_CODEBOOK_ID BP_CDEBOOK_SUB

Basics of creating an item type

Using BP_PLOT as an example, we need to do the following steps:

1. Create a new DKITemTypeDefICM, passing it a reference to the datastore:

   ```
   itemType = new DKItemTypeDefICM(dsICM);
   ```

2. Give the item type a name that will be used to create these items of this item type later:

   ```
   itemType.setName("BP_PLOT");
   itemType.setDescription("BP_PLOT document");
   ```

3. Add the desired attributes to the item type, setting qualifiers such as nullable, text searchable, and whether it is unique or not. The following code snippet shows how to add to attributes, BP_NUMBER and BP_DOC_DESC to the BP_PLOT item type:

   ```
   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_NUMBER");
   attr.setUnique (false);
   itemType.addAttr(attr);

   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_DESC");
   attr.setUnique (false);
   itemType.addAttr(attr);
   ```

4. Set the item type classification, and add the item type to the persistent datastore:

   ```
   itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
   itemType.add();
   itemTypeRel = new DKItemTypeRelationDefICM(dsICM);
   itemTypeRel.setSourceItemTypeID(dsDefICM.getEntityIdByName("BP_PLOT"));
   itemTypeRel.setTargetItemTypeID(dsDefICM.getEntityIdByName("ICMBASE"));
   itemTypeRel.setDefaultACLCode(3);
   dsDefICM.add(itemTypeRel);
   ```

Repeat the coding above for all item types defined in Table 6-2 on page 133.

Implementing to a JSP file

You need to do the following steps to put the code together and run it in your application development environment:

1. Create a new JSP file by:
   a. Right-click **Web content folder** under the II4C Project.
   b. Select **New** → **other** from its context menu.
   c. Name the new file as CMaddItemType.
2. Copy and paste the code before the closing </BODY> tag.
3. Add the import statement at the beginning of your JSP file.

   ```
   <%@ page import="com.ibm.mm.sdk.server.*, com.ibm.mm.sdk.common.*"%>
   ```

4. Add the code to connect to the datastore in the beginning of creating the itemType:

   ```
   DKDatastoreICM dsICM = new DKDatastoreICM();
   dsICM.connect ("LSDB", "LSADMIN", "lspasswd","");
   ```

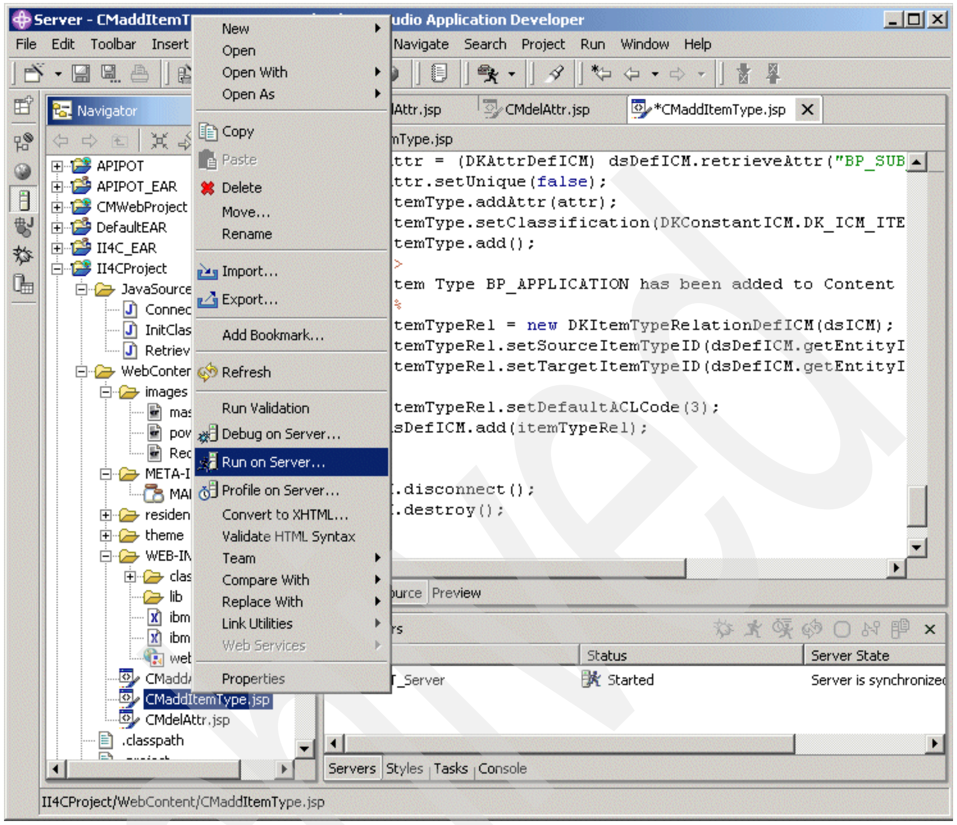
Alternatively, you can import the entire JSP file provided along with this redbook to your project, change the appropriate settings such as Library Server name, user ID, and password, and go from there. See Example B-1 on page 540 for the complete JSP file.

Testing the code

To test the code, execute the JSP file on the server:

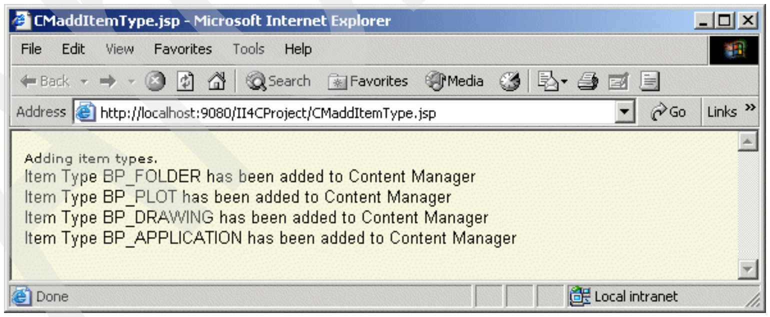
1. Right-click **CMaddItemType.jsp**.
2. Select **Run on Server** from the context menu (see Figure 6-30).

A browser should open up (see Figure 6-31).



```
ttr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_SUB
ttr.setUnique(false);
temType.addAttr(attr);
temType.setClassification(DKConstantICM.DK_ICM_ITE
temType.add();
>
tem Type BP_APPLICATION has been added to Content
%
temTypeRel = new DKItemTypeRelationDefICM(dsICM);
temTypeRel.setSourceItemTypeID(dsDefICM.getEntityI
temTypeRel.setTargetItemTypeID(dsDefICM.getEntityI

temTypeRel.setDefaultACLCode(3);
sDefICM.add(itemTypeRel);

.disconnect();
.destroy();
```

Figure 6-30 Run CMaddItemType.jsp on the server

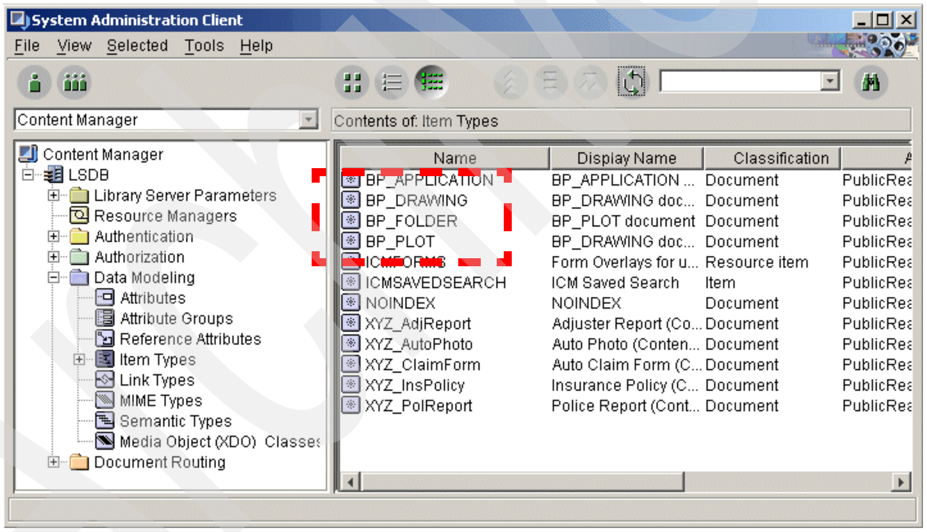

CMaddItemType.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:9080/II4CProject/CMaddItemType.jsp
Go
Links
Done
Local intranet

```
Adding item types.
Item Type BP_FOLDER has been added to Content Manager
Item Type BP_PLOT has been added to Content Manager
Item Type BP_DRAWING has been added to Content Manager
Item Type BP_APPLICATION has been added to Content Manager
```

Figure 6-31 Result of running CMaddItemType.jsp

There is one additional manual step you must perform to set the default Resource Manager. But before doing that, let us make sure that the code runs successfully by verifying the item type creation through the System Administration Client:

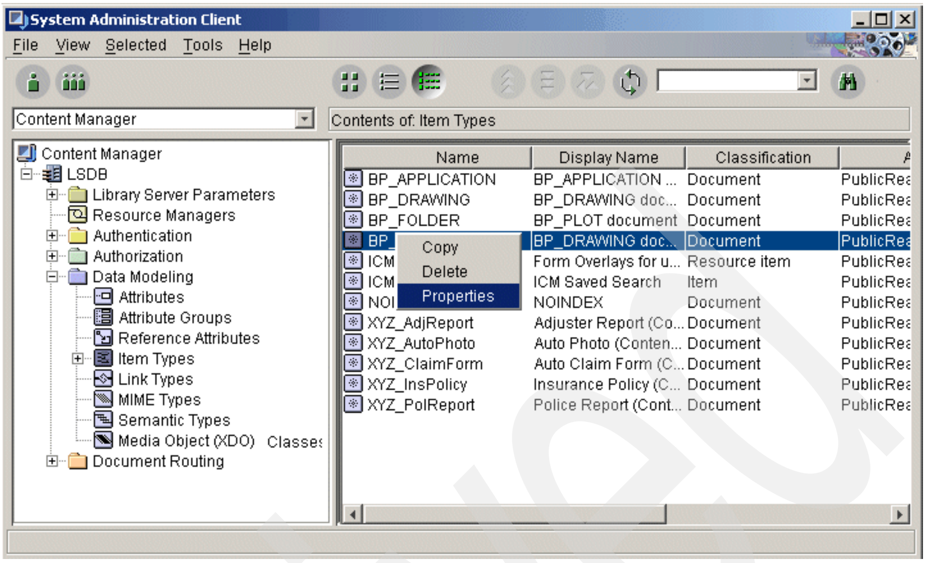
1. Start System Administration Client:
   a. Select **Start** → **Programs** → **IBM Content Manager for MultiPlatforms V8.2** → **System Administration Client**.
   b. Enter your Library Server user ID and password, then click **OK** to login to the System Administration main screen.
2. Refresh your screen by clicking the Refresh icon after you run your code. Refer to Figure 6-27 on page 129 to see where the Refresh icon is located.
3. Select **Item Types** from the left panel and select the **details view**.
4. Click the **Name** column to bring up the item type names in alphabetical order. The newly created item types should show up on top of the list with a "BP_" postfix (see Figure 6-32).



Name	Display Name	Classification	A
BP_APPLICATION	BP_APPLICATION ...	Document	PublicRea
BP_DRAWING	BP_DRAWING doc...	Document	PublicRea
BP_FOLDER	BP_PLOT document	Document	PublicRea
BP_PLOT	BP_DRAWING doc...	Document	PublicRea
ICMFORMS	Form Overlays for u...	Resource item	PublicRea
ICMSAVEDSEARCH	ICM Saved Search	Item	PublicRea
NOINDEX	NOINDEX	Document	PublicRea
XYZ_AdjReport	Adjuster Report (Co...	Document	PublicRea
XYZ_AutoPhoto	Auto Photo (Conten...	Document	PublicRea
XYZ_ClaimForm	Auto Claim Form (C...	Document	PublicRea
XYZ_InsPolicy	Insurance Policy (C...	Document	PublicRea
XYZ_PolReport	Police Report (Cont...	Document	PublicRea

Figure 6-32 Item Types added to Content Manager

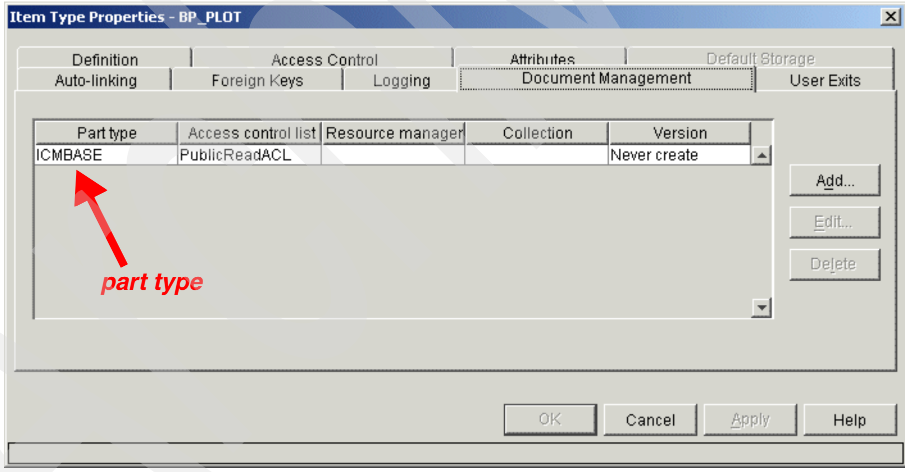
5. Right-click any of the item types, for example, BP_PLOT, and select **Properties** (see Figure 6-32) to view the property of the item type.
6. Check the different properties of the item type. For example, under Document Management tab, check that the part type is ICMBase (see Figure 6-33). Also check for Access Control for the item.



Content Manager
- LSDB
  - Library Server Parameters
  - Resource Managers
  - Authentication
  - Authorization
  - Data Modeling
    - Attributes
    - Attribute Groups
    - Reference Attributes
    - Item Types
    - Link Types
    - MIME Types
    - Semantic Types
    - Media Object (XDO) Classes
  - Document Routing

Name	Display Name	Classification	A
BP_APPLICATION	BP_APPLICATION ...	Document	PublicRea
BP_DRAWING	BP_DRAWING doc...	Document	PublicRea
BP_FOLDER	BP_PLOT document	Document	PublicRea
BP_	BP_DRAWING doc...	Document	PublicRea
ICM	Form Overlays for u...	Resource item	PublicRea
ICM	ICM Saved Search	Item	PublicRea
NOI	NOINDEX	Document	PublicRea
XYZ_AdjReport	Adjuster Report (Co...	Document	PublicRea
XYZ_AutoPhoto	Auto Photo (Conten...	Document	PublicRea
XYZ_ClaimForm	Auto Claim Form (C...	Document	PublicRea
XYZ_InsPolicy	Insurance Policy (C...	Document	PublicRea
XYZ_PolReport	Police Report (Cont...	Document	PublicRea

Figure 6-33 Check properties of the created item type



Part type	Access control list	Resource manager	Collection	Version
ICMBASE	PublicReadACL			Never create

Figure 6-34 Document Management Properties

Setting the default Resource Manager for item types

After you have created your item type, there is one more step to do in order to make the item type ready to store resources in Content Manager.

This step is to assign the item a default Resource Manager. The sample code is supplied with the product. The choices are as follows:

- SResourceMgrDefSetDefaultICM.java
- SResourceMgrDefRetrievalICM.java

Read the instruction carefully to be able to run the code successfully.

For simplicity, we supply a Java class BP_ResourceMgrDefSetDefaultICM to do this task. We combine the code of these two classes in the Java class. You can find the source code in the additional material provided by this redbook.

If your environment does not match our sample application environment (Item types names are different), edit the following line of the code that assigns the item type to a specific Resource Manager and recompile your Java class:

```
setDefaultResourceManagerForItemType(dsICM, "BP_PLOT",
rmUserDefinedReferenceName);
```

To compile your code:

1. Start Information Integrator for Content Development environment:

   Select **Start** → **Programs** → **Enterprise Information Portal for Multiplatforms 8.2** → **Development Window**.

2. At the command line, type cd <DIR> where DIR is the directory where the BP_ResourceMgrDefSetDefaultICM is located.

3. Compile the code:

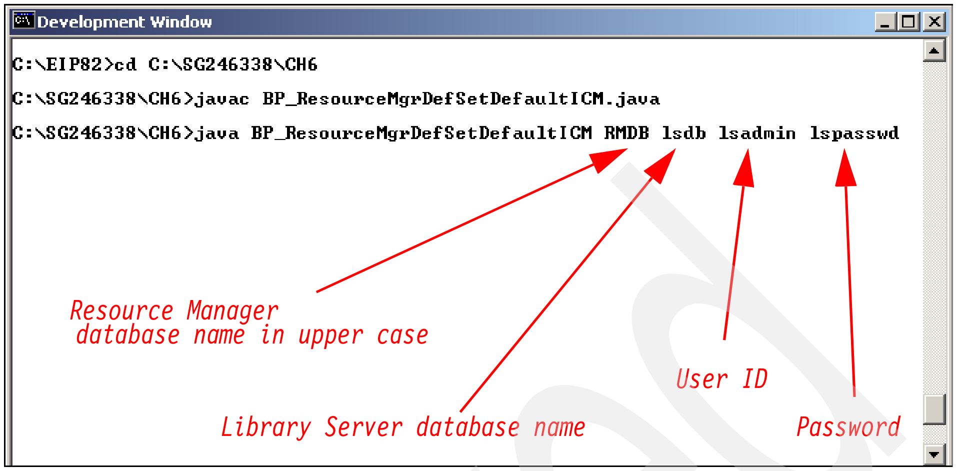
   `javac BP_ResourceMgrDefSetDefaultICM.java`.

4. To run the command (see Figure 6-35):

   `java BP_ResourceMgrDefSetDefaultICM RMDB lsdb lsadmin lspasswd`

   Where BP_ResourceMgrDefSetDefaultICM takes four arguments:

   – The Resource Manager database name (*must be in upper case*)
   – The Library Server database name
   – The Library Server user ID
   – The Library Server password



```
C:\EIP82>cd C:\SG246338\CH6

C:\SG246338\CH6>javac BP_ResourceMgrDefSetDefaultICM.java

C:\SG246338\CH6>java BP_ResourceMgrDefSetDefaultICM RMDB lsdb lsadmin lspasswd
```

Development Window
Resource Manager database name in upper case
Library Server database name
User ID
Password

Figure 6-35 Compile and run the BP_ResourceMgrDefSetDefaultICM

6.3.4 Working with items

An item is an instance of an item type. Each item may have components. Components are what make the hierarchical tree of data. There is one root component, and zero to many child components. In this section, we show how to create items.

Before you begin, check Resource Manager

Before you begin creating items, there are two things you need to check:

- ► Verify that the Resource Manager server is up and running.
- ► Verify that the Resource Manager is installed correctly and configured properly.

Attention: This is a very important step. You must make sure the Resource Manager runs properly before you run and test your code.

Manually import a text document as NOINDEX via the Content Manager Windows client. Once imported, retrieve this document. If this works, then, your Resource Manager is ready to go.

Start the Resource Manager server:

1. Go to the DOS prompt.

2. Change change the directory to <WAS_HOME>\AppServer\bin where <WAS_HOME> is where the WebSphere Application Server is installed. The default is c:\Program Files\IBM\WebSphere.
3. Run the following command to check the status of the application servers:

   ```
   serverstatus -all
   ```

4. If the Resource Manager application server is not started, start the Resource Manager:

   ```
   startserver rmas
   ```

   You need to substitute rmas with the value in your system setup. This is the Resource Manager server name. The default is usually icmrm.

Alternatively, you can add the Resource Manager server as a Windows services and start it from there:

1. Run the following command to add Resource Manager to the Windows service:

   ```
   wasservice -add “Resource Manager” -servername rmas -userid rmadmin
   -password password
   ```

   You need to substitute rmas, rmadmin, and password, with the values in your system setup.
2. Go to the Windows services control panel.
3. Start “IBM WebSphere Application Server V5 - Resource Manager” service.

Test to see if the Resource Manager works:

1. Open the browser and go to:

   ```
   https://<host name>/rmas/ICMResourceManager
   ```

2. You should get the following response:

   ```
   IBM Content Manager V8. Your request:null. No order found to process.
   ```

Test to see if the Resource Manager is working accordingly:

1. Start Windows Client.
2. Import some text documents as NOINDEX.
3. Retrieve these documents and review.

If you are able to import and retrieve these documents, the Resource Manager is working properly. We now proceed to create some items for the case study.

Basics of item creation

In our case study, we work with a simple item. For information about more complex items creation, refer to the sample code located under <CMBROOT_DIR>\samples\java\icm, where <CMBROOT_DIR> is the installation drive and directory where Information Integrator for Content is installed. The default is C:\CMBROOT.

An item is created as a *DKDDO* object. It should always be created using the createDDO() method from DKDatastoreICM class, which allows for an automatic setup of important information in the DKDDO structure.

To create an item:

1. Use DKDatastoreICM.createDDO() to create item. Two parameters must be given to the createDDO() method:
   - Item type
   - Semantic type which could be a Document, a Folder, an Item, or any user-defined type. This value is stored in a property called "DK_CM_PROPERTY_ITEM_TYPE".
2. With the new DKDDO object, do the following actions:
   - Setting or modifying its attributes
   - Creating child components if required
3. Add the item to the Persistent Datastore using the add() method.

Our data model was created using the code in the previous examples in this chapter. We created the attributes in 6.3.2, “Working with attributes” on page 120 and the item types in 6.3.3, “Working with item types” on page 131. You should complete these sections before you continue.

Creating items of Document Model type

In your WebSphere Studio Application Developer window, select the WebContent folder under II4CProject project. From the context menu, select **New** → **Other**, then select **Web** from the left panel and **JSP file** from the right panel and click **Next**. In the next screen you will be asked to enter a name for the JSP file; type `CMimportPlots` and click **Finish**.

Earlier, we created our item types that conform to the Content Manager Document Model. This extends the default item capabilities by including document parts. We designed our item types to include one type of document parts, which is the ICMBASE, a LOB resource that holds part of the document, in order to add our plots that come as GIF files. The mime type, such as JPEG and TIFF, will be set during the item creation process, so that any type of objects could be added as part of the document.

To create an item and add to the datastore, follow these steps, which must be done in the same order as given:

1. Connect to the datastore.
2. Create a DDO of an item type, and give it the required parameter to be able to act as a document.
3. Create a DKLobICM object to be added to the DDO.
4. Set Parts MIME Type/Type of Contents.
5. Load content into parts (for example, from a file in the file system).
6. Retrieve the part from the DDO.
7. Add the parts to the document.
8. Add the DDO's attributes.
9. Add the DDO to the persistent datastore.

The code in Example 6-7 shows the implementation of those steps.

We first have to import a class `InitClass.java` in the II4CProject project (see Example 6-5). This class is very simple, it declares an object of type Vector. Elements of the Vector are two-dimensional arrays that hold the building permit Number BP_NUMBER and the file name associated with the plot.

Select the JavaSource folder under the project and from its context menu select **import**. Then select File System and click **Next**. Browse to the location where the additional material provided by the redbook is located, select the appropriate folder, check the file `InitClass.java` and click **Finish**.

Alternatively, you can create a new Java class in the JavaSource folder and copy the code in Example 6-5 and paste it in your Java class.

Example 6-5 InitClass.java

```
/*
 * Created on Mar 31, 2004
 *
 * To change the template for this generated file go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */

/**
* To change the template for this generated type comment go to
 * Window&gt;Preferences&gt;Java&gt;Code Generation&gt;Code and Comments
 */
public class InitClass {
   public InitClass()
   {
   }

   public java.util.Vector getData()
   {
      java.util.Vector myVector= new java.util.Vector();
      String dataDefinition[][] = new String[10][2];
      myVector.addElement(dataDefinition);

      dataDefinition[0][0]="111111";
      dataDefinition[0][1]="apartment_image.gif";
      dataDefinition[1][0]="222222";
      dataDefinition[1][1]="building_elevation_med.gif";
      dataDefinition[2][0]="333333";
      dataDefinition[2][1]="building_plan_med.gif";
      dataDefinition[3][0]="444444";
      dataDefinition[3][1]="cool_building_full.gif";
      dataDefinition[4][0]="555555";
      dataDefinition[4][1]="cool_deck_image.gif";
      dataDefinition[5][0]="666666";
      dataDefinition[5][1]="houseplan1_med.gif";
      dataDefinition[6][0]="777777";
      dataDefinition[6][1]="houseplan2_med.gif";
      dataDefinition[7][0]="888888";
      dataDefinition[7][1]="landscape_design_med.gif";
      dataDefinition[8][0]="999888";
      dataDefinition[8][1]="restaurant_med.gif";
      dataDefinition[9][0]="999999";
      dataDefinition[9][1]="studio_med.gif";

      return myVector;
   }

   public java.util.Vector getSSNData()
      {
      java.util.Vector SSNVector= new java.util.Vector();
      SSNVector.addElement("111-11-1100");
      SSNVector.addElement("111-11-1200");
      SSNVector.addElement("111-11-1300");
      SSNVector.addElement("111-11-1400");
      SSNVector.addElement("111-11-1500");
      SSNVector.addElement("111-11-1600");
      return SSNVector;
      }
}
```

When you edit your `CMimportPlots.jsp` file, please insert the import directive at the beginning of the file, this time we have to also import the InitClass (see Example 6-6).

Example 6-6 import statement: must include the InitClass

```
<%@ page import="com.ibm.mm.sdk.server.*,
         com.ibm.mm.sdk.common.*,
         java.io.*,
         InitClass"
%>
```

Add the import code before the ending </BODY> tag. Example 6-7 shows the source code for creating the BP_PLOT item type and importing plot documents.

Example 6-7 CMimportPlots: create BP_PLOT item type and import plot documents

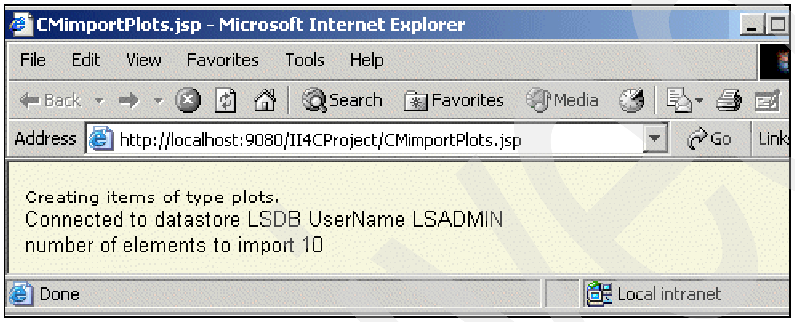
```
<!--  BEGINNING OF CODE -->
<%
DKDatastoreICM dsICM = new DKDatastoreICM();
dsICM.connect("LSDB","LSADMIN","lspasswd","");
%>
Connected to datastore <%=dsICM.datastoreName()%> UserName
<%=dsICM.userName()%><br>
<%
System.out.println("Get Vector"  );
InitClass initclass = new InitClass();
System.out.println("Get Vector after initclass"  );
java.util.Vector retrievedData = (java.util.Vector)initclass.getData();
System.out.println("Retrieved vector " +  retrievedData  );
String dataArray[][] = (String[][])retrievedData.elementAt(0);
System.out.print("array size = " + dataArray.length  );
String imgDir = "C:\\SG246338\\CH6\\CMObjects\\";
String imgFile = "";
String fileName = "";

%>
number of elements to import <%=dataArray.length%>
<%


DKDDO ddoDocument;
// Loop through the array and retrieve the data to make up the item
for (int i = 0 ; i < dataArray.length ; i++)
{
   ddoDocument = dsICM.createDDO("BP_PLOT", DKConstant.DK_CM_DOCUMENT);
   System.out.println("Created a Document.");
    //-------------------------------------------------------------
    // Create Parts
    //-------------------------------------------------------------
   System.out.println("Creating Parts...");
   DKLobICM   base      = (DKLobICM)  dsICM.createDDO("ICMBASE",
DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
   System.out.println("Setting Parts' MIME Type / Type of Contents...");
    base.setMimeType("image/gif");
   System.out.println("Loading Content into Parts...");

   imgFile = dataArray[i][1];
   fileName = imgDir+imgFile;
   System.out.println("  fileName ..." + fileName);
   base.setContentFromClientFile(fileName);
   System.out.println("Created Parts.");
   System.out.println("Accessing the DKParts Attribute...");
   DKParts dkParts = (DKParts)
ddoDocument.getData(ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConsta
nt.DK_CM_DKPARTS));

   short dataid =
ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
   if(dataid==0)
      throw new Exception("No DKParts Attribute Found!  DDO is either not a CM
Document Model Item or Document has not been explicitly retrieved.");
   dkParts = (DKParts) ddoDocument.getData(dataid);
   System.out.println("Adding Parts to Document...");
   dkParts.addElement(base);
   System.out.println("Added Parts to Document.");
                // Adding New Document to Persistent Datastore


ddoDocument.setData(ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_NUMB
ER"),dataArray[i][0]);

ddoDocument.setData(ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_DOC_
DESC"),"Engineering Drawings: "+dataArray[i][1]);

ddoDocument.setData(ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_SUB_
DATE"),    java.sql.Date.valueOf("2004-03-16"));

ddoDocument.setData(ddoDocument.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_DOC_
VERSION"),"Version 1 ");
      ddoDocument.add();
}

dsICM.disconnect();
dsICM.destroy();
%>
<!--  END OF CODE -->
```

Running and testing the creation code for your items

You need to run the code and test whether items have been created via Windows Client.

To run the code:

1. Right-click **CMImportPlots.jsp**.
2. Select **Run on Server** from its context menu.
3. The output on the browser is shown in Figure 6-36.


CMimportPlots.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/II4CProject/CMimportPlots.jsp
Go Links
Done
Local intranet

```
Creating items of type plots.
Connected to datastore LSDB UserName LSADMIN
number of elements to import 10
```

Figure 6-36 Output of CMimportPlots.jsp on the browser

Use Windows Client to test that the code runs successfully:

1. Start Windows Client:
   a. Select **Start** → **Programs** → **IBM Content Manager V8** → **Client for Windows**.
   b. Enter your Library Server user ID and password, then click **OK** to login to the Windows Client main screen (see Figure 6-37).
2. Click **Search**.
3. From the screen in Figure 6-38, select **BP_PLOT** item type.
4. You do not have to enter any values in the search criteria, and just click **OK**.
5. You should be able to see the list of items that your code has just imported (see Figure 6-39).

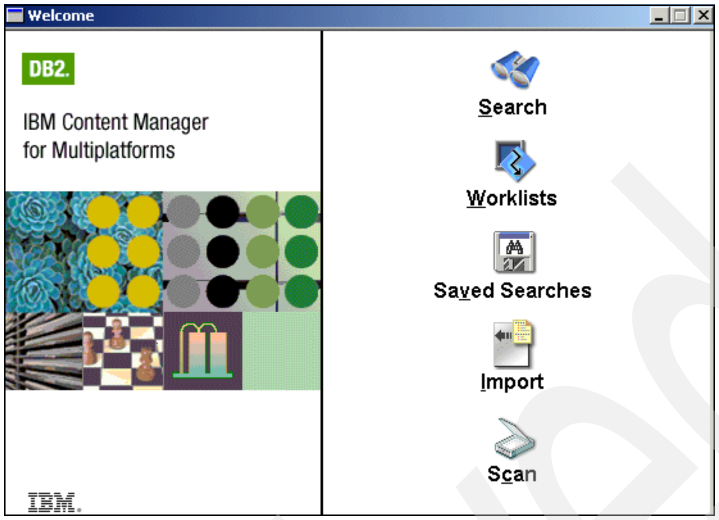
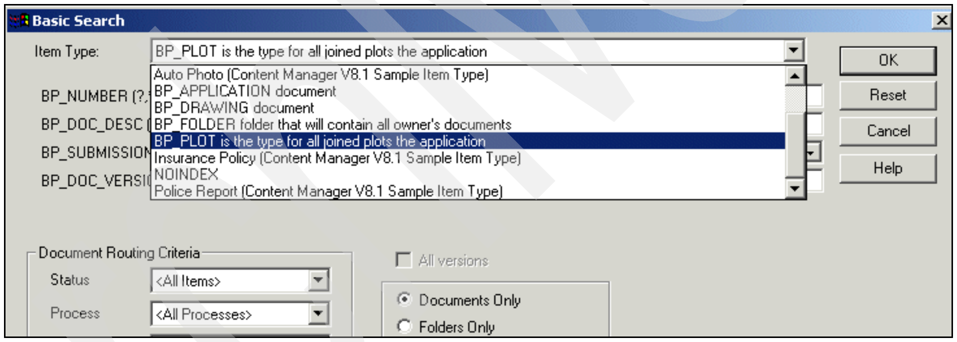


Figure 6-37 Content Manager Windows Client



Item Type: BP_PLOT is the type for all joined plots the application

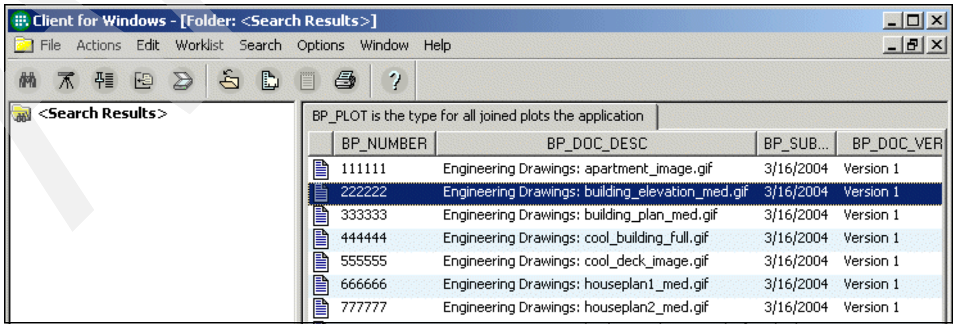
Auto Photo (Content Manager V8.1 Sample Item Type)
BP_APPLICATION document
BP_DRAWING document
BP_FOLDER folder that will contain all owner's documents
BP_PLOT is the type for all joined plots the application
Insurance Policy (Content Manager V8.1 Sample Item Type)
NOINDEX
Police Report (Content Manager V8.1 Sample Item Type)

BP_NUMBER (?,
BP_DOC_DESC (
BP_SUBMISSION
BP_DOC_VERSI(

Status: <All Items>
Process: <All Processes>

(●) Documents Only
( ) Folders Only

Figure 6-38 Search BP_Plot Item type from Content Manager Windows Client


Client for Windows - [Folder: <Search Results>]
File Actions Edit Worklist Search Options Window Help
<Search Results>

BP_PLOT is the type for all joined plots the application

BP_NUMBER	BP_DOC_DESC	BP_SUB...	BP_DOC_VER
111111	Engineering Drawings: apartment_image.gif	3/16/2004	Version 1
222222	Engineering Drawings: building_elevation_med.gif	3/16/2004	Version 1
333333	Engineering Drawings: building_plan_med.gif	3/16/2004	Version 1
444444	Engineering Drawings: cool_building_full.gif	3/16/2004	Version 1
555555	Engineering Drawings: cool_deck_image.gif	3/16/2004	Version 1
666666	Engineering Drawings: houseplan1_med.gif	3/16/2004	Version 1
777777	Engineering Drawings: houseplan2_med.gif	3/16/2004	Version 1

Figure 6-39 View items in Content Manager Windows Client

Viewing Content Manager items on the Web

In this section we build a simple code to retrieve the items imported previously into the Content Manager and view them in the Web page in a very simple way. In the later chapter, we use the Content Manager Document Viewer and use its advanced features.

To retrieve an object from Content Server:

1. Connect to the Content Server by creating a datastore object and then connect to the datastore:

   ```
   DKDatastoreICM dsICM = new DKDatastoreICM();
   dsICM.connect("LSDB","LSADMIN","lspasswd","");
   ```

   Substitute the Library Server name, user ID, and password with the values in your system setup.

2. Create a search query and specify query options:

   ```
   String query = "( /BP_PLOT )";
   DKNVPair options[] = new DKNVPair[3];

   // Set max using default: “0”. This means no Maximum.
   options[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0");

   // Specify any retrieval options desired. Default is ATTRONLY.
   options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,    new
   Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));

   // Mark the end of the NVPair
   options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
   ```

3. Execute the query and retrieve result set in dkResultSetCursor:

   ```
   dkResultSetCursor cursor = dsICM.execute(query,
   DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
   ```

4. Iterate the results using the fetchNext() function of the dkResultSetCursor:

   ```
   while((ddo = cursor.fetchNext())!=null)
   {.... }
   ```

5. Retrieve DDO with the proper option, DKConstant.DK_CM_CONTENT_YES, in order to retrieve contents also:

   ```
   itemId = ((DKPidICM)ddo.getPidObject()).getItemId();
   attnum =ddo.dataCount();
   ddo.retrieve(DKConstant.DK_CM_CONTENT_YES);
   ```

6. Retrieve the attributes and their values:

   ```
   for (short i = 1;i <= attnum-1; i++)
   {
   ```

```
   Short type   = (Short)
ddo.getDataPropertyByName(i,DKConstant.DK_CM_PROPERTY_TYPE);
   ddo.getDataName
   ddo.getData(i)
}// end of for loop
```

7. Retrieve parts, and retrieve parts IDs:

```
short dataid =
ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
if(dataid > 0)
{
   // if a DKParts collection exists, print contents
   DKParts dkParts = (DKParts) ddo.getData(dataid);

   // obtain the DKParts collection
   dkIterator iter = dkParts.createIterator();

   // Create an iterator to go through Collection
   while(iter.more())
   {
      // While there are children, print list
       DKDDO part = (DKDDO)iter.next();
       part.retrieve();
       DKPidICM PIDx =(DKPidICM) part.getPidObject();
       String pidxdo = PIDx.pidString();
       // ....
   }//while iter
}//if dataid>0
```

8. Display parts using their parts IDs in a simple servlet:

```
%><td><a style="cursor:hand"
onClick="ViewObject('<%=java.net.URLEncoder.encode(pidxdo)%>');">View this
plot</a><%
```

9. Destroy the dkResultSetCursor object:

```
cursor.destroy();
```

In case of a search query, *it is very important to destroy the cursor/iterator.*

10. Disconnect from your datastore and destroy the object:

```
dsICM.disconnect();
dsICM.destroy();
```

Do not forget to close the connection and destroy the datastore DKDatastoreICM object that you have just created.

To put it together in a JSP file, follow these steps:

1. Create a new JSP file. See the detailed steps of creating a JSP file from “Implementing to a JSP file” on page 134.
2. Name the file CMretrievePlots.jsp.
3. Add the import directive in the beginning of the file:

   ```
   <%@ page import="com.ibm.mm.sdk.server.*,
                      com.ibm.mm.sdk.common.*"%>
   ```

4. Add the following Java script in the <HEAD> tag, just before the ending </HEAD> tag.

   ```
   <Script Language="JavaScript">
      function ViewObject(fpid)
         {
   window.open("/II4CProject/ViewObject.jsp?PID="+fpid,"_blank","toolbar=0,scr
   ollbars=0,location=0,statusbar=1,menubar=0,resizable=1,width=500,height=670
   ");
      return false;
         }
   </Script>
   ```

5. Within the JavaScript, it calls another JSP file `ViewObject.jsp` with a parameter which is string representation of the DKPidICM that represents that part. The part will be retrieved using this information and viewed using a simple servlet.

For the complete code of CMretrievePlots.jsp, you can find it in the additional material provided by this redbook. We also provide the code in Example B-2 on page 544.

The ViewObject.jsp is called from CMretrievePlots.jsp file. It is used as container for the image/object to display. It does the following:

1. Retrieves the part ID passed as request parameter.

   ```
   String pidxdo = request.getParameter("PID");
   ```

2. Calls a servlet to returns the array of bytes that form that object.

   ```
   <img src="/II4CProject/servlet/RetrievePlots?PID=<%=pidxdo%>">
   ```

You can download the complete ViewObject.jsp code from the additional material provided by this redbook or copy and paste it from Example B-3 on page 548.

The servlet, RetrievePlots.java, is called from ViewObject.jsp file. It retrieves the part using its part ID and gets its contents in an array of bytes that will be returned to the calling JSP and displayed inside an <IMG> tag. It does the following steps:

1. In the doGet method, the servlet retrieves the PID:

   ```
   String s = httpservletrequest.getParameter("PID");
   DKPidICM pidx = new DKPidICM(s);
   ```

2. It gets the part:

   ```
   DKLobICM blob =new DKLobICM(dsICM);
   blob.setDatastore(dsICM);
   blob.setPidObject(pidx);
   blob.retrieve();
   ```

3. It gets the content in an array of bytes:

   ```
   long nl =blob.getSize();
   int n= (new Long(nl)).intValue();
   byte abyte0[] =new byte[n];
   abyte0 = blob.getContent();
   ```

4. It writes to the output string:

   ```
   servletoutputstream.write(abyte0);
   ```

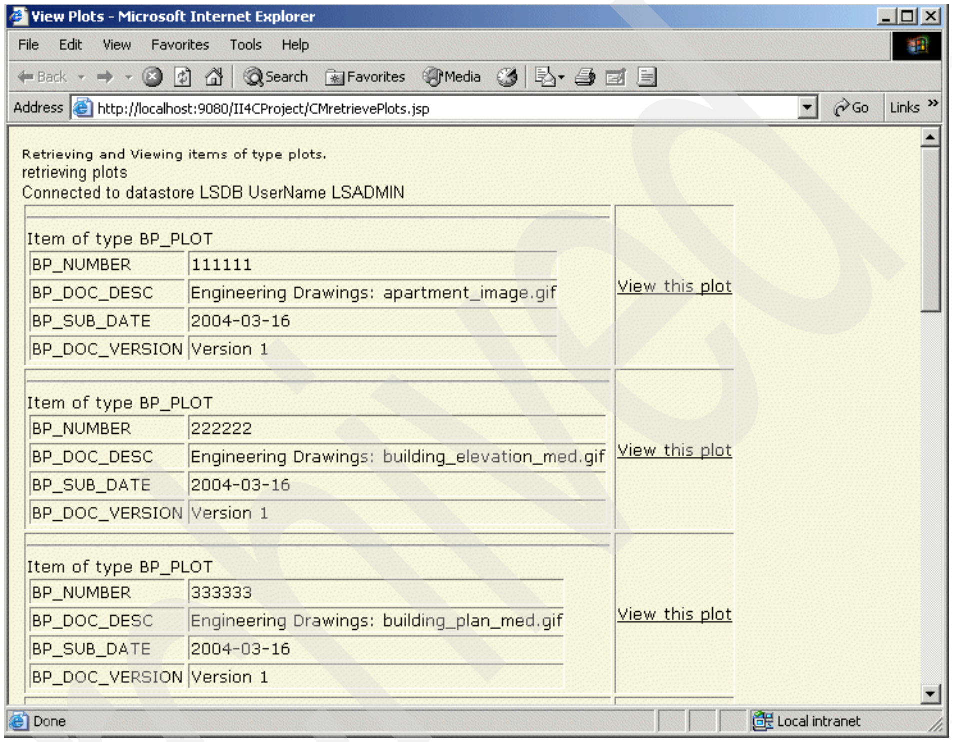
You can download the complete RetrievePlots.java code from the additional material provided by this redbook or copy and paste it from Example B-4 on page 549.

Test and run the code

To test and run the code, do the following steps:

1. Run CMretrievePlots.jsp on server.

   You should get a result similar to the output shown in Figure 6-40.


View Plots - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/II4CProject/CMretrievePlots.jsp
Go Links
Done
Local intranet

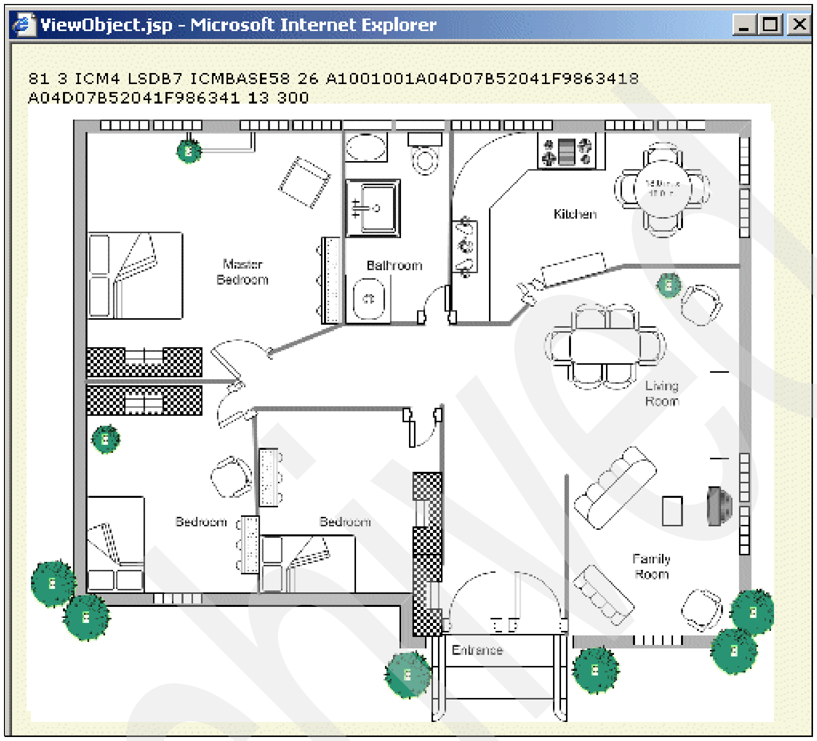
```
Retrieving and Viewing items of type plots.
retrieving plots
Connected to datastore LSDB UserName LSADMIN
```

Item of type BP_PLOT		View this plot
BP_NUMBER	111111	
BP_DOC_DESC	Engineering Drawings: apartment_image.gif	
BP_SUB_DATE	2004-03-16	
BP_DOC_VERSION	Version 1	
Item of type BP_PLOT		View this plot
BP_NUMBER	222222	
BP_DOC_DESC	Engineering Drawings: building_elevation_med.gif	
BP_SUB_DATE	2004-03-16	
BP_DOC_VERSION	Version 1	
Item of type BP_PLOT		View this plot
BP_NUMBER	333333	
BP_DOC_DESC	Engineering Drawings: building_plan_med.gif	
BP_SUB_DATE	2004-03-16	
BP_DOC_VERSION	Version 1	

Figure 6-40 Results in the browser

2. Click the link, **View this plot**.

   You should get a screen similar to Figure 6-41.



```
81 3 ICM4 LSDB7 ICMBASE58 26 A1001001A04D07B52041F9863418
A04D07B52041F986341 13 300
```

Figure 6-41 View object contents

6.4 Conclusion

In this chapter, we provided step-by-step procedures on setting up your development environment using WebSphere Studio Application Developer. We showed you how to quickly get started with developing Web applications using Information Integrator for Content Java OO APIs. Some of the samples we showed include the creation of attributes, item types, and item. In the following chapters, we show you how to develop a generic sample application and a customized Web application for Content Manager.

For a complete Java OO API reference, refer to the on-line Information Center for Content Manager or Information Integrator for Content.

7

Building a generic application

In the previous chapter, we showed you how to set up your development environment and develop some simple codes to create Content Manager data models and items using the Java OO APIs from the Information Integrator for Content (II for Content).

In this chapter, we show you how to build a generic application using the non-visual Java beans from II for Content. Using the non-visual Java beans is the preferred way of developing Content Manager Web-based applications.

We cover the following topics:

- Introduction
- Development environment setup
- Sample application implementation including:
  - Listing available servers and connecting to servers
  - Listing entities, search templates, attributes, and search criterias
  - Conducting search, processing search results
  - Displaying item information
  - Document Routing processes: Selecting a process and starting an item in a process
  - Document Routing work lists: Listing, opening, listing actions for advancing work item, advancing a work item

7.1 Introduction

Information Integrator for Content provides a set of Java beans that you can use to build Web applications. The beans are a set of Java classes that follow beans conventions as stated by Sun Microsystems. The set incorporates all the connectors allowing one program to access different types of back-end repositories. Unlike what we described in Chapter 6, “Quick start in Web application development” on page 95, you program to a higher level of abstraction and avoid some of the details and complexities of the individual connector. The disadvantage is you give up some of the more specialized functions of the Java OO APIs for a given connector.

As described in 5.3.1, “Non-visual beans” on page 83, when developing Web applications using the Java beans, the beans typically serve as the model in a Model-View-Controller architecture. Servlets act as the controller. Java Server Pages (JSPs) act as the view. This architecture provides separation of application logic from presentation logic.

Other architectures are possible when building Web Applications, such as JSPs-only, which would both act as the controller and view, using the non-visual beans as the model. This is what we presented in Chapter 6, “Quick start in Web application development” on page 95. There is also a set of visual Java beans shipped with Information Integrator for Content that uses the non-visual Java beans along with Java swing objects for creating Java Applet programs.

A good set of samples is provided when you install Information Integrator for Content. These can be found in the SAMPLES\java\beans directory under where the Information Integrator for Content is installed (the default install directory is C:\CMBROOT). These samples can be compiled and run from a command line, and demonstrate how to use the beans to logon, list information, search, retrieve, and perform other functions. There is a readme.html file in this directory that documents how to run them, and what each samples does.

In this chapter, we show you how to develop a sample Web applications using the non-visual Java beans.

For detailed reference of the usage of the beans including all the properties and their methods, refer to Information Center of Information Integrator for Content along with OO APIs. Another good source of reference is *IBM Information Integrator for Content Workstation Application Programming Guide*, SC27-1347.

7.2 Development environment setup

Before you start building a generic Web application using Information Integrator for Content non-visual Java beans, you must first set up your development environment. If you have not already set up an environment based on the previous chapter and you are not familiar with WebSphere Studio Application Developer, we recommend that you go back to 6.2, “Development environment setup” on page 97 and follow the procedures provided there before you continue, noting the different Web project name and EAR file name.

In summary, you need to do the following tasks to set up your environment:

1. “Setting up a new Web project” on page 99.
   - Enter `Redbook` for Project Name.(II4CProject2.
   - Enter for `Redbook_EAR` for EAR project. (II4C2_EAR).
   - Add external JAR files to the Libraries of the Java Build Path:
     - cmb81.jar
     - cmbviewer81.jar
     - cmbod81.jar
     - cmbsdk81.jar
     - db2java.zip

   Use Add External JARs options to add them. The external cmb*.jar files can be found in the <CMBROOT_DIR>\lib directory, in which <CMBROOT_DIR> is where II for Content is installed. The db2java.zip can be found in <DB2_DIR>\java, in which <DB2_DIR> is where DB2 is installed.
2. “Setting up a new server project to run the sample code” on page 104.
   - Enter `II4C_Server` for both Server name and Folder name.
   - Enter `9080` for HTTP port number.
   - Add external JAR files to the class path of the server environment:
     - cmb81.jar
     - cmbviewer81.jar
     - db2java.zip
   - Add an external folder where cmbicmenv.properties file is located. The default is <CMGMT_DIR>.
   - Add RedbookEAR to II4C_Server.

Note: If you have followed the previous chapter and set up a developer environment already with a testing server, you do not need to set up another server project. You can use the testing server you created earlier for testing the code developed in this chapter.

7.3 Sample application

In this chapter, we build a general purpose Web application capable of logging onto Content Manager Version 8, OnDemand, or a Federated system using a controller servlet and JSPs. We use the Model-Viewer-Controller method with the model being the Information Integrator for Content non-visual Java beans, the viewer being a set of JSPs, and the controller being a custom servlet.

Using this sample application, a user can do the following tasks:

1. Select a server and log in.
2. Get a list of entities or search templates.
3. Choose an entity or a search template, enter search values, and submit a search.
4. Get a list of search results. Process the search result:
   a. Select the entry.
   b. If it is a document, the user can retrieve and view its content.
   c. If it is a folder, the user can open the folder.
5. If working with CM V8 backend, the user can:
   a. Start a document in a Document Routing process.
   b. View and work with worklists.

There are 10 Java files, 7 JSP files, and 1 CSS file developed for this application:

- RBConnectionPool.java — Handles connection pool.
- RBController.java — Controller servlet.
- RBDocRouting.java — Handles Document Routing.
- RBFolderOpen.java — Opens folder.
- RBListEntities.java — Lists entities.
- RBSearch.java — Handles search.
- RBSearchEntry.java — Handles search entries.
- RBServers.java — Obtains a list of servers.
- RBSortArray.java — Sorts arrays.
- RBView.java — Handles view.
- RBAdvanceEntry.jsp — Displays advance entries.
- RBItemList.jsp — Displays items.
- RBListEntities.jsp — Displays entities.
- RBLogin.jsp — Displays log in.
- RBProcessEntry.jsp — Displays process entry screen.
- RBSearchEntry.jsp — Displays search entry.
- RBWorklist.jsp — Displays worklist.
- Master.css

7.3.1 How to get the most benefit from this chapter

In order to get the most out of this chapter, you should be familiar with Content Manager's Windows Client or eClient first. If you are not, use them and work with them. Make sure that you are familiar with how they work, and that you understand Content Manager item types (entities), items, search, and view capability. If you skipped the previous chapter, you should populate load sample data using Content Manager's First Steps. With the sample data, you can check out how to retrieve and view data that is in Content Manager using the provided client.

When you are familiar with how the out-of-the-box Content Manager application works, download the source code from the additional material provided by this redbook. Run the generic Web application we provided. Make sure that they run under your environment and understand how it works.

For the remainder of this chapter, we go into details as how each piece is put together. We start with the controller servlet and go through each function that the application is doing along with the associated non-visual Java bean that is involved. For each bean, we briefly explain what it is, how it is obtained, and some of the methods that the bean has. The purpose of listing some of these methods is to familiar you with what the bean can do. For a complete list of all the methods, refer to the on-line Information Center, API Reference, Java beans.

Note that it is not our intention to show you how to build the entire application piece by piece, rather we intend to show you how it works. From this, we want you to get a general idea of how the non-visual Java beans work to produce the Web application. Depending on your business scenario, you can modify the source code to fit your requirement.

Important: These sample programs are provided as educational material and should be used *as-is*. It is not the responsibility of the IBM Technical Support Team to provide any support or troubleshooting on these codes. When you encounter problems with them, however, we welcome you to submit them to the IBM ITSO Redbook team. We will try our best to answer them.

In the next chapter, Chapter 8, "Building a case study application" on page 207, we develop a specific building permit application for Redbrook County. In that chapter, we will go into more details and use a step-by-step approach to show you how you can develop the building permit application piece by piece.

7.3.2 Set up and run the sample application

To set up the generic Web application, do the following steps:

1. Go to J2EE Web Perspective:

   Select **Window** → **Open Perspective** → **J2EE**.

2. Import Java files:

   a. Expand **Redbook** project, if it is not already expanded. Right-click **JavaSource** under Redbook project file.

   b. Click **Import**.

   c. Select **File system** and click **Next**.

   d. At From directory, click **Browse**. Go to the directory where you downloaded the source code associated with this chapter. Select that folder. Click **OK**.

   e. Select all the Java files. (There should be 10 of them.) Click **Finish**.

3. Import JSP files:

   a. Right-click **WebContent** under Redbook project file.

   b. Click **Import**.

   c. Select **File system** and click **Next**.

   d. At From directory, click **Browse**. Go to the directory where you downloaded the source code associated with this chapter. Select that folder. Click **OK**.

   e. Select all the JSP files. (There should be 7 of them.) Click **Finish**.

4. Replace existing master.css file:

   a. Under Redbook project, WebContent, right-click **theme**.

   b. Click **Import**.

   c. Select **File system** and click **Next**.

   d. At From directory, click **Browse**. Go to the directory where you downloaded the source code associated with this chapter. Select that folder. Click **OK**.

   e. Select master.css file. Click **Finish**.

   f. Click **Yes** when prompt to whether to overwrite master.css file or not.

Figure 7-1 shows where and how they should be imported to your project. Run RBController.associated with this chapter, and import them into your newly created project.

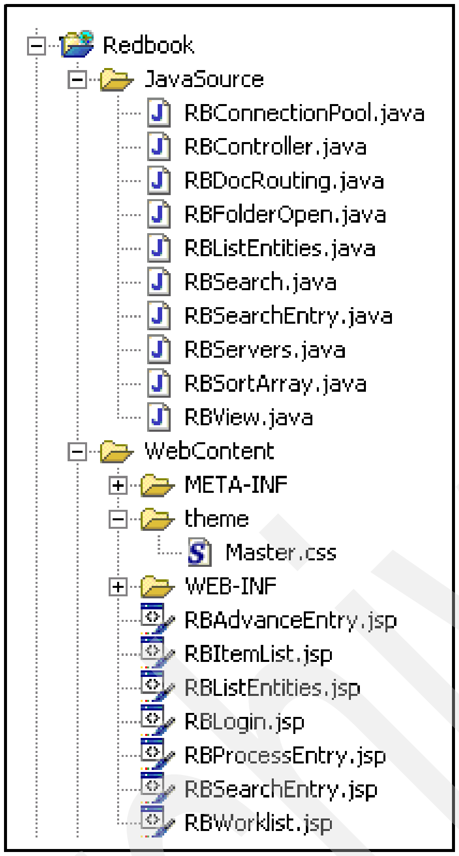

Redbook
JavaSource
RBConnectionPool.java
RBController.java
RBDocRouting.java
RBFolderOpen.java
RBListEntities.java
RBSearch.java
RBSearchEntry.java
RBServers.java
RBSortArray.java
RBView.java
WebContent
META-INF
theme
Master.css
WEB-INF
RBAdvanceEntry.jsp
RBItemList.jsp
RBListEntities.jsp
RBLogin.jsp
RBProcessEntry.jsp
RBSearchEntry.jsp
RBWorklist.jsp

Figure 7-1 WebSphere Studio Application Developer setup for the Redbook project

To run the sample application:

1. Stop WebSphere Application Server 1.

   You can either stop it through Windows services, or manually from the command line:

   a. Go to <WAS_HOME>\AppServer\bin directory, in which <WAS_HOME> is where you installed WebSphere.

   b. Run **serverstatus -all** to check whether server 1 is up and running.

   c. If so, use **`stopserver server1`** to stop it.

2. Under Redbook project, JavaSource, right-click RBController.java.
3. Select **Run on Server**.
4. Select **II4C_Server**, and click **Finish**.

7.3.3 Import the beans into the code

The following import statements are typically needed for establishing a reference to servlets using the Java beans.

```
import javax.servlet.*;      // Needed for being a servlet
import javax.servlet.http.*; // Needed for HttpServletRequest
import java.io.*;  // Needed for direct writing to ServletOutputStream
import java.net.URLDecoder; // Needed for passing PID and other parameters

import com.ibm.mm.beans.*;   // Needed for all CMB beans
import com.ibm.mm.beans.workflow.*; // Needed for CM V8 Doc Routing
```

7.3.4 The Controller program: RBController

You can start the sample application from a browser by going to `http://hostname/Redbook/servlet/RBController` without any parameters. This invokes the RBController, and has it present a panel to logon to a selected server.

The RBController servlet, invoked from a browser, follows normal Java servlet coding practices, with the doGet method performing all the work. The doGet method retrieves a parameter named *action* from the request block. Based on the action value, the controller servlet performs the proper action. It sets the name of a response JSP, and redirects the control to the JSP.

Example 7-1 shows a code snippet of RBController.java.

Specifically, when you first invoke the controller, the action value is null. The servlet calls the RBServers class (which we will explain later) to list all the servers and pass the control to RBLogin.jsp.

When the action value is ListEntities, the controller calls the RBListEntities class (which we will explain later) to list all the entities and then pass the control to RBListEntities.jsp.

Example 7-1 RBController.java

```
import javax.servlet.*;      // Needed for being a servlet
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*;   // Needed for all CMB beans
import java.io.*; // Needed for direct writing to ServletOutputStream

public class RBController extends HttpServlet {
    ...

public synchronized void doGet(HttpServletRequest request, HttpServletResponse
response)
```

```
    throws ServletException, IOException {

    try {
      String responseJSP = null;
      String action = request.getParameter("action");
      if (action == null) {
        RBServers.list(request);
        responseJSP = "RBLogin.jsp";

      } else if (action.equals("ListEntities")) {
        RBListEntities.list(request);
        responseJSP = "RBListEntities.jsp";

      } else if (action.equals("SearchEntry")) {
        RBSearchEntry.list(request);
        responseJSP = "RBSearchEntry.jsp";

...

      } // end if (action == null)

      RequestDispatcher rd =
getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
      rd.forward(request, response);
    } //end try

...
```

Review the complete source code you imported or see Example B-5 on page 551 for the source code. Make sure you understand it before you continue.

7.3.5 Listing available servers: CMBConnection

The CMBConnection bean's primary purpose is to connect to the back-end repositories with a valid user ID, password, and server name. To connect, you have to set a valid server name, server type, user ID, and password.

RBServer.java

This class is used to list the available servers by setting the CMBConnection's dsType property, and issuing the listDataSources() method. What is returned is an array of String[] that contains the server names of that type that you may want to put into a list box for the user to select. You have to issue the method for each type of back-end repository you want to list.

As mentioned in the previous section, the method RBServer.list() gets control when no “action” parameter is passed to the controller servlet. It does the following tasks:

1. Creates a String array dsType with the three values of server types for connecting.

   ```
   String[] dsType = new String[3];
   dsType[0] = "ICM";
   dsType[1] = "OD";
   dsType[2] = "Fed";
   ```

2. Instantiates a new CMBConnection bean

   ```
   CMBConnection connection = new CMBConnection();
   ```

3. In a nested loop, set the dsType of the bean, and use the CMBConnection.listDataSources() method to list all the data sources of that type. It then loops through the sourceNames, and adds a string with the type and name to the vector:

   ```
   for (int i = 0; i < dsType.length; i++) {
         // Set the connection's type to the dsType element
         connection.setDsType(dsType[i]);
         System.out.println("Listing " + dsType[i] + " servers");
         String[] sourceNames = connection.listDataSources();

           // Loop through the array and add the dsType and name to the vector
           for (int j = 0; j < sourceNames.length; j++) {
             dsList.add(dsType[i] + " " + sourceNames[j]);
           }
         }
   ```

4. Put the vector in the request object to be picked up by the JSP.

   ```
   request.setAttribute("list", dsList);
   ```

Review the complete source code you imported or see Example B-6 on page 554 for the source code. Make sure you understand it before you continue.

RBLogin.jsp

Once a list of available servers is obtained by RBServer class, RBController passes the control to RBLogin.jsp. This JSP file generates a login form where the user can enter user ID and password, and select the back-end repository server. Upon submission, it executes RBController, with a hidden action value of “ListEntities” and user ID and password values.

Specifically, RBLogin.jsp does the following tasks:

1. Generates a form to execute RBController, with a hidden action of value “ListEntities,” the user ID and user password entered by the user:

   ```
   <form method=POST action=RBController target=_top>
   <input type=hidden name=action value="ListEntities">

   <tr><td>Userid </td><td><INPUT type=text name=userid   value=""></td></tr>
   <tr><td>Password </td><td><INPUT type=password name=password value="">
   ```

2. Gets a list of available servers from the “list” attribute:

   ```
   Vector vStat = (Vector) request.getAttribute("list");
   ```

3. If the list is not empty, populates a drop-down box with the server type and name of the vector:

   ```
   if (!(vStat == null)) {
     vSize = vStat.size();
   }
   for (int i = 0; i < vSize; i++) { %>
   <OPTION value="<%=vStat.elementAt(i) %>"><%=vStat.elementAt(i) %>
   ```

4. Enables the user to login with a submit button:

   ```
   <tr><td>&nbsp</td><td><input TYPE="submit" VALUE="Logon"
   tabindex="7"></table>
   ```

Review the complete source code you downloaded from the additional material or see Example B-7 on page 555 for the source code.

RBLogin.jsp generated HTML and screen

The sample generated HTML file from RBLogin.jsp is shown in Example 7-2.

Example 7-2   RBLogin.jsp generated HTML file

```
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="ListEntities">
<tr><td>Userid </td><td><INPUT type=text name=userid   value=""></td></tr>
<tr><td>Password </td><td><INPUT type=password name=password value="">
</td></tr>
<tr><td>Server </td><td>
<SELECT NAME=server SIZE="1" >
<OPTION value="ICM LSDB">ICM LSDB
<OPTION value="ICM ICCDEMO">ICM ICCDEMO
<OPTION value="ICM CMDEMO">ICM CMDEMO
<OPTION value="ICM ICMNLSDB">ICM ICMNLSDB
<OPTION value="OD CMDEMO">OD CMDEMO
<OPTION value="OD cs8demo">OD cs8demo
<OPTION value="Fed ICCDEMO">Fed ICCDEMO
<OPTION value="Fed CMDEMO">Fed CMDEMO
</SELECT> </td></tr>
```

```
<tr><td>&nbsp</td><td><input TYPE="submit" VALUE="Logon" tabindex="7"></table>
```

The sample output of RBServer.java and RBLogin.jsp is shown in Figure 7-2.

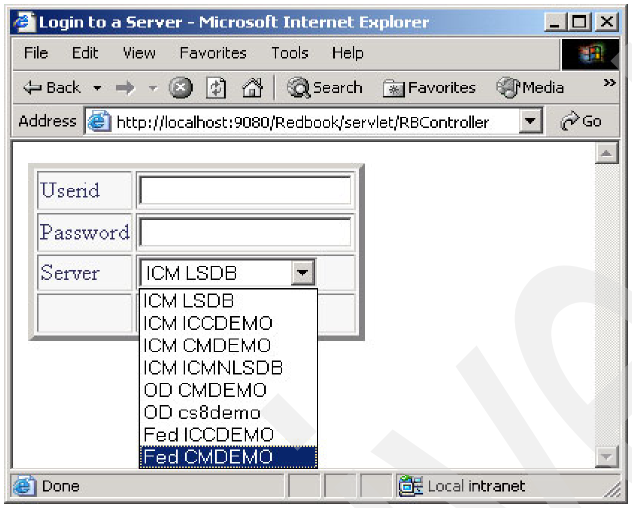

Login to a Server - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/Redbook/servlet/RBController
Go
Userid
Password
Server
ICM LSDB
ICM LSDB
ICM ICCDEMO
ICM CMDEMO
ICM ICMNLSDB
OD CMDEMO
OD cs8demo
Fed ICCDEMO
Fed CMDEMO
Done
Local intranet

Figure 7-2 Sample output of RBServers.java and RBLogin.jsp

7.3.6 Connecting to server: CMBConnection, CMBConnectionPool

The CMBConnection bean maintains the connection to a back-end server which could be a native content server or a federated server. It has the server name, server type, user ID, and password as properties. Using the property values, the connect() method physically goes out and tries to do a valid logon onto the named server.

A connected CMBConnection bean is needed to access any information on the connected content server. There is a set of system wide beans to access and manage the various parts of the content server.

The primary Java beans are as follows:

CMBSchemaManagement	Access information about entities, attributes, search templates and search criterion
CMBQueryServices	Perform searches, and interrogate results.
CMBDataManagement	Manipulate content stored on the server.
CMBDocRoutingQueryServiceICM	Manipulate CM V8 content through a Document Routing Process.

These are some of the CMBConnection bean methods:

- void connect(String server, int portNo, int servicePortNo, String userid, String passwd, String newPasswd) — Log on the server.
- void disconnect() — Log off from the server.
- CMBDataManagement getDataManagement() — Get the reference to the CMBDataManagement bean.
- dkDatastore getDatastore() — Get the datastore used by this bean.
- CMBDocRoutingDataManagementICM getDocRoutingDataManagementICM() — Get the reference to the CMBDocRoutingDataManagementICM bean.
- CMBDocRoutingQueryServiceICM getDocRoutingQueryServiceICM() — Get the reference to CMBDocRoutingQueryServiceICM bean.
- String listDataSources() — Get a list of all available servers.
- void setDsType(String dsType) — Set the dsType property.

Special note about OnDemand

OnDemand uses both the terms Folders and Application Groups in their data model to denote entities to search on. By default, the CMBConnection bean, when connected to an OnDemand server, uses Application Groups as the primary entity. To use Folders, you must set the CMBConnection bean's configuration string with:

```
connection.setConfigurationString("ENTITY_TYPE=TEMPLATES");
```

This allows you to use the OnDemand Folders in the same way as the Federated Search templates.

CMBConnectionPool

The CMBConnectionPool bean allows an application to make multiple connections to a back-end server, and then allocate the connections as needed. This can improve the response time to the end users by not having them logon and logoff each time they visit the site. They still must provide a valid user ID and password each time they logon to retrieve a connection from the pool.

The properties for the CMBConnectionPool bean are similar to the properties of CMBConnection, plus others to manage the maximum and minimum number of concurrent connections.

Some of the CMBConnectionPool bean methods are as follows:

- void destroy() — Destroy the connection pool.

- void freeConnection(CMBConnection connection) — Free a connection previously obtained using getConnection().
- CMBConnection getConnection(String userid, String password) — Get a connected instance of CMBConnection.
- CMBConnection getConnection(String userid, String password, int sessionID) — Get a connected instance of CMBConnection from the same session.
- int getMaxConnections() — Return the maximum number of total connections, both free and in use, that may be created.
- int getMinFreeConnections() — Get a minimum number of free connections that will be pooled.
- void setMaxConnections(int maxConnections) — Set the maximum number of connections.

Our sample application uses CMBConnectionPool to maintain the connection across sessions and requests to the application from the same user.

To request a connection from the pool, you need to call:

```
CMBConnection connection = RBConnectionPool.getConnection(request);
```

To return the connection to the pool, you need to call:

```
RBConnectionPool.freeConnection(connection);
```

On the first call to getConnection, RBConnectionPool will use the entry fields for user ID and password, and parse the selected item in the drop-down box for server type and server name. This and subsequent calls will return a CMBConnection to the application. See Example 7-3 for a code snippet of RBConnectionPool.getConnection() method. See Example B-8 on page 556 for the complete source code.

Example 7-3 RBConnectionPool.java code snippet for getConnection() method

```
public static synchronized CMBConnection getConnection(HttpServletRequest 
request)
    throws
      IllegalAccessException, MalformedURLException, InstantiationException,
      ClassNotFoundException, PropertyVetoException, CMBException, Exception
      {

    HttpSession session = null;

if (connectionPool == null) {
     System.out.print("Creating new bean connection Pool ... ");
      connectionPool = new CMBConnectionPool();
      connectionPool.setConnectionType(CMBBaseConstant.CMB_CONNTYPE_LOCAL);
connectionPool.setServiceConnectionType(CMBBaseConstant.CMB_CONNTYPE_LOCAL);
      connectionPool.setClientURLString(null);
      connectionPool.setCsURLString(null);
      connectionPool.setServiceClientURLString(null);
      connectionPool.setServiceCsURLString(null);

      // Parse the string server into the server name and server type
     String server = request.getParameter("server");
     int space = server.indexOf(" ");
     String dsType = server.substring(0, space);
     String dsName = server.substring(space + 1);

     if(dsType.equals("OD"))
      connectionPool.setConfigurationString("ENTITY_TYPE=TEMPLATES");

      connectionPool.setDsType(dsType);
      connectionPool.setServerName(dsName);
      // Set up a new ssession object.
     session = request.getSession(true);
     session.setAttribute("userid", request.getParameter("userid"));
     session.setAttribute("password", request.getParameter("password"));
     session.setAttribute("server", dsName);
     session.setAttribute("dsType", dsType);
    }
    else {
      session = request.getSession(false);
    }

    System.out.println("Returning CMBConnection");
    CMBConnection connection = connectionPool.getConnection((String)
session.getAttribute("userid"),
     (String) session.getAttribute("password"));

    return connection;
  }
```

7.3.7 Schema: CMBSchemaManagement

The CMBSchemaManagement bean accesses all the system information about entities, attributes, search templates, and search criteria. Each of the following four objects has a corresponding bean (CMBEntity, CMBAttribute, CMBSearchTemplate, and CMBSTCriterion) that can be created from the CMBSchemaManagement bean; either a list of all, or a particular one by name.

In order to create a CMBSchemaManagement bean, use the *getSchemaManagement()* method of the CMBConnection bean, or use the default bean constructor and set its connection property:

```
CMBSchemaManagement schema = connection.getSchemaManagement();
```

-or-

```
CMBSchemaManagement schema = new CMBSchemaManagement();
schema.setConnection(connection);
```

These are some of the methods of the CMBSchemaManagement bean:

- CMBConnection getConnection() — Get the connection bean reference.
- String getCurrentServerName() — Get the current server name.
- String getCurrentServerType() — Get the current server type.
- CMBEntity getEntities() — Get a list of top-level entity objects.
- CMBEntity getEntity(String entityName) — Get the entity object for the given name.
- String getEntityName(int index) — Get the entity name for the given index.
- CMBSearchTemplate String getEntityNames() — Get a list of all entity names.
- CMBSearchTemplate CMBSearchTemplate(int index) — Get a search template object given the index.
- CMBSearchTemplate CMBSearchTemplate(String name) — Get a search template object given the search template's name.
- String getSearchTemplateName(int index) — Get the search template name given the index.
- String[] getSearchTemplates() — Get a list of all search template names.
- CMBSearchTemplate[] getSearchTemplates() — Get a list of all search template objects.

7.3.8 Listing entities: CMBEntity

Entity is the common term used to denote the various terms for document type by the different backend servers. For Content Manager Version 8, this is the item type. For DB2, this is the table. For OnDemand, it is the application group. The federated repository has federated entities. These entities all have attributes as labels for the metadata of the content stored, and are used in searches to find particular records.

As mentioned in the earlier section, the CMBSchemaManagement has a method *getEntities()* that returns an array of CMBEntity beans, one bean for each entity with all the system information in the connected system.

The CMBEntity bean is a helper class for the CMBSchemaManagement bean. It provides functions for retrieval of entity metadata, including attribute definitions, entity type, and sub-entity definitions.

These are some of the CMBEntity bean methods:

- CMBAttribute getAttribute(String attrName) — Get a single attribute in this entity, given the name of the attribute.
- String getAttributeNames() — Get the names of this entity's attributes.
- CMAttribute[] getAttributes() — Get the attribute objects for this entity.
- String getDisplayName() — Return the display name for the entity.
- String getName() — Return the name of the entity.
- String getParentEntityName() — Return the name of the parent entity, or an empty string if there is none.
- String[] getPartTypes() — Determine the type of parts that documents of this entity type may contain.
- CMEntity[] getSubEntites() — Get all the sub entities in this entity.
- short getType() — Return the entity type, 0 by default.

7.3.9 Listing search templates: CMBSearchTemplate

Search templates provide a search function based on a pre-defined template. They are used in both the federated system and the OnDemand system. In the federated system, they are associated with federated entities, and can further define the attributes to be for searching or "display only." A subset of the allowed Boolean operators (=, !=, <, >, etc.) can be defined to each attribute. Also specific users or user groups can be granted or denied use of the search template.

In OnDemand, SearchTemplates are the same as OnDemand folders when the CMBConnection was instantiated with the connection string:

```
setConfigurationString("ENTITY_TYPE=TEMPLATES");
```

With the CMBSchemaManagement bean, the method *getSearchTemplates()* returns an array of CMBSearchTemplates beans. These beans have all of the system information about the search template, so you may use the information in your program. The method getSearchTemplate(String name) returns one bean by name from the connected system.

These are some of the CMBSearchTemplate bean methods:

- Vector getAllCriteria() — Get a list of all criteria, both search criteria and display criteria.
- short getCriterionDefaultOp (String critName) — Get the each criterion's default search operator.
- String[] getCriterionDisplayName() — Get a list of display name for criterion.
- String getCriterionDisplayName(String critName) — Get a criterion's display name given a search criterion.
- String[] getCriterionName() — Get a list of search and display criteria names.
- String[] getCriterionPredefinedValues (String critName) — Get a list of predefined values for a search criterion.
- short getCriterionValidOps(String critName) — Get a list of valid search operators for a given search criterion.
- String getEntityName() — Get the name of the federated entity associated with this search template.
- String getItemAttrName() — Get the default attribute name to be used as the item name.
- Object getResults() — Get the collection of result items.
- Vector getSearchCriteria() — Get a list of search criteria in order.
- short getSearchCriterionOperator(String critName) — Get the search operator.

RBListEntities.java

In our sample application, RBListEntities.java lists either the entities if the connection is for Content Manager Version 8, or the search templates if it is for OnDemand or Federated server.

The RBListEntities.list() method does the following tasks:

1. Establish the connection to the server and create a new schema management from the connection:

```
CMBConnection connection = RBConnectionPool.getConnection(request);
CMBSchemaManagement schema = connection.getSchemaManagement();
```

2. Decide if it should list search templates or entities based on the dsType:

```
if (connection.getDsType().equals("OD") ||
connection.getDsType().equals("Fed")) {
      // OD and FED use searchTemplates
      CMBSearchTemplate searchTemplates[] = schema.getSearchTemplates();
...
} else {
      // ICM and most other dsTypes use entity and attributes.
      CMBEntity entities[] = schema.getEntities();
```

3. Call RBSortArray to sort the results of listing:

```
RBSortArray sort = new RBSortArray();
      sort.sort(searchTemplates, true)
```

4. Loop through the array and add a line to the vector that has the name and the display name separated by a “;”. Only Content Manager Version 8 has the display name. For search templates that do not have a display name, add the name twice separated by a “;”:

```
for (int i = 0; i < searchTemplates.length; ++i) {
        vector.add(
java.net.URLEncoder.encode(searchTemplates[i].getName())
            + ";" + searchTemplates[i].getName());
```

   If it is for Content Manager Version 8, use the following statement:

```
        vector.add(
          java.net.URLEncoder.encode(entities[i].getName()) + ";" +
entities[i].getDisplayName());
      }
```

5. Put the vector in the request object to be picked up by the JSP:

```
session.setAttribute("list", vector);
```

For the complete source code of RBListEntities.java, see the downloaded file from the additional material provided by this redbook or view the source in Example B-9 on page 557.

RBListEntities.jsp

RBListEntities.jsp generates a table that has the links to start a search on the selected entity. It does the following tasks:

1. Get the list of entities from the session attribute:

```
Vector vStat = (Vector) session.getAttribute("list");
```

2. Loop through the list and parse the element in the vector into the entity's (or search template's) name and display name:

```
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) {
    String element = (String) vStat.elementAt(i);
   // Parse the string in the vector into entity 'Name' and 'DisplayName'
   String name = element.substring(0, element.indexOf(";"));
   String displayName = element.substring(element.indexOf(";") + 1);
```

3. For each element, a row in the table would be similar to:

```
<tr><td>
<a href="RBController?Entity=NOINDEX&action=SearchEntry">NOINDEX
</td></tr>
```

For the complete source code of RBListEntities.jsp, see the downloaded file from the additional material provided by this redbook or view the source in Example B-10 on page 558.

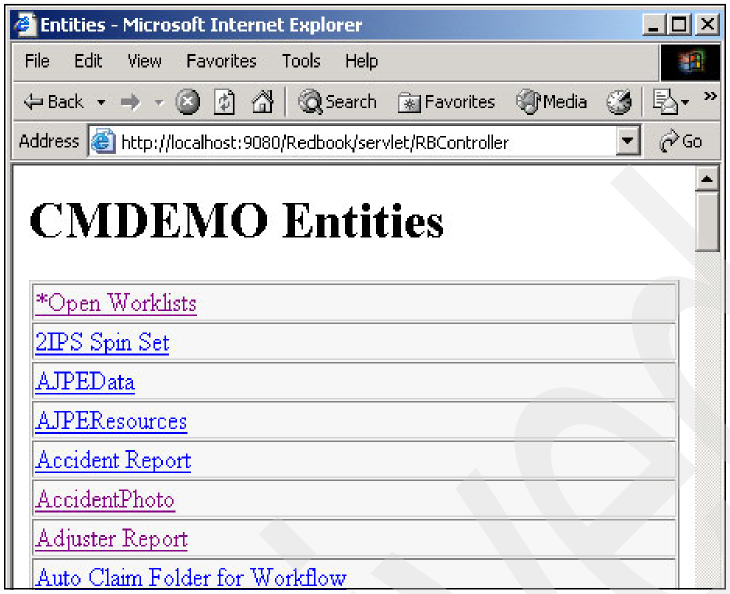
RBListEntities.jsp generated HTML

The sample generated HTML file from RBListEntities.jsp is shown in Example 7-4.

Example 7-4 RBListentities.jsp generated HTML

```
<TABLE cellpadding="1" cellspacing="1" border="1" bgcolor="#80ffff">
<tr><td><a href="RBController?action=Worklist">*Open Worklists</TD></tr>
<tr><td>
<a href="RBController?Entity=BP_APPLICATION&action=SearchEntry">BP_APPLICATION
document
</TD></tr>

   (One table row for each item in the vector)

<tr><td><a href="RBController?Entity=ns_test2&action=SearchEntry">ns_test2
</TD></tr>
</table>
```

The sample output of RBListEntities.java and RBListEntities.jsp is shown in Figure 7-3.


Entities - Microsoft Internet Explorer

Address http://localhost:9080/Redbook/servlet/RBController

CMDEMO Entities

*Open Worklists
2IPS Spin Set
AJPEData
AJPEResources
Accident Report
AccidentPhoto
Adjuster Report
Auto Claim Folder for Workflow

Figure 7-3 Sample output of RBListEntities.java and RBListEntities.jsp

7.3.10 Listing attributes: CMBAttribute

With an instantiated CMBEntity bean, use the *getAttributes()* method to get an array of CMBAttribute. These will be only the attributes in the given entity.

The CMBAttribute bean provides methods to query the properties of an attribute definition. This is a helper class of the CMBSchemaManagement bean.

These are some of the CMBAttribute bean methods:

- CMBAttribute getAttribute(String attrName) — Get the CMBAttribute object for the named attribute of the child component or for the named nested child component of this child component.
- CMBAttribute[] getAttributes() — Get a list of CMBAttribute objects for all attributes of the child component and for nested child components.
- String getDefaultValue() — Get the default value for the attribute.
- String getDescription() — Get the attribute description.
- String getDisplayName() — Get the display name of the attribute.
- String getEntityName() — Get the name of the entity that the attribute is defined in.
- String getName() — Get the name of the attribute.

- int getSize() — Get the display size of the attribute.
- int getStringType() — Get the string type value of the attribute.
- boolean isChildComponent() — Check if the attribute is a child component.
- boolean isNullable() — Check if the attribute is nullable.
- boolean isTextSearchable() — Check if the attribute is text searchable.

7.3.11 Listing search criteria: CMBSTCriterion

With an instantiated CMBSearchTemplate bean, use the *getSearchCriteria()* method to get a Vector of CMBSTCriterion beans. You may need to do so to solicit values from the user for searching or for reindexing

The CMBSTCriterion bean represents information for a particular search criterion of a search template. This is a helper class for the CMBSearchTemplate bean.

Here are some of the CMBSTCriterion bean methods:

- String getAttrName() — Get the associated attribute name.
- short getAttrType() — Get the associated attribute type.
- short getCriteriaPosition() — Get the order of the search criterion to be displayed in the search template.
- short getDefaultOperator() — Get the default search operator.
- String getDefaultValue() — Get the default search value.
- String getDisplayName() — Get the display name of the search criterion.
- short getDisplayPosition() — Get the column display order within the search result.
- short getDisplayWidth() — Get the display column initial width.
- String getEntityName() — Get the associated entity name.
- String getName() — Get the name of this search criterion.
- short getOperator() — Get the operator for the criterion.
- String[] getPredefinedValues() — Get a list of predefined values.
- int getType — Get search criterion type.
- short[] getValidOperators() — Get a list of valid search operators.
- String getValue() — Get the search value for the search criterion.

Notice that for each get method, there is usually a corresponding set method that we do not list here.

RBSearchEntry.java

In our sample application, we list the attributes of entities and the search criterion of search templates in RBSearchEntry.java code.

Specifically, RBSearchEntry.java does the following tasks:

1. Get the entity name:

   ```
   String entityName = request.getParameter("Entity");
   session.setAttribute("Entity", entityName);
   ```

2. Create a CMBSchema Management bean:

   ```
   CMBSchemaManagement schema = connection.getSchemaManagement();
   ```

3. If the connection type is OnDemand or Federated, with the entity name, create a single CMBSearchTemplate bean. Loop through the search criterion, and put the names of the search criteria names in a vector:

   ```
   if (connection.getDsType().equals("OD") ||
   connection.getDsType().equals("Fed")) {
       Vector searchCrits =
   schema.getSearchTemplate(entityName).getSearchCriteria();

         for (int i = 0; i < searchCrits.size(); i++) {
          CMBSTCriterion sCrit = (CMBSTCriterion) searchCrits.elementAt(i);
           vector.add(java.net.URLEncoder.encode(sCrit.getName()) + ";"
             + sCrit.getDisplayName());
          }
   ```

4. Else, for Content Manager, with the entity name, create a single CMEntity bean. Loop through the list of attributes and put the real names and descriptive names of the attributes into a vector:

   ```
   } else {
         CMBEntity entity = schema.getEntity(entityName);
         CMBAttribute[] attrs = entity.getAttributes();

         RBSortArray sort = new RBSortArray();
         sort.sort(attrs, true);

         for (int i = 0; i < attrs.length; ++i) {
           vector.add(
             java.net.URLEncoder.encode(attrs[i].getName()) + ";" +
   attrs[i].getDisplayName());
         }
       }
   ```

   Note: For Content Manager Version 8, the end users are probably accustomed to seeing the display names of attributes, but the system needs the real names to perform searches and updates.

5. Set the vector (either a list of search criteria names or a list of attribute names (including display names) into attrlist of the session:

   ```
   session.setAttribute("attrlist", vector);
   ```

For the complete source code of RBSearchEntry.java, refer to the downloaded file from the additional material provided by this redbook, or view the source in Example B-11 on page 559.

RBSearchEntry.jsp

The RBSearchEntry.jsp displays a single drop down box to select the Boolean operator for the search, and a table with attribute names and entry fields for values of the search.

Specifically, RBSearchEntry.jsp does the following tasks:

1. To simplify our sample application code, hard coded a list of available operators that can be used for search:

   ```
   <TR><TH>Comparison<br>Operator</TH><td>
   <SELECT NAME="_compop" SIZE=1 >
   <OPTION  SELECTED value=<%=CMBBaseConstant.CMB_OP_EQUAL%>> Equals
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_NOT_EQUAL%>> Not Equals
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_GREATER%>> Greater Than
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_LESS%>> Less Than
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_GREATER_EQUAL%>> Greater or Equal
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_LESS_EQUAL%>> Less or Equal
   <OPTION  value=<%=CMBBaseConstant.CMB_OP_LIKE%>> Like
   </SELECT></td></tr>
   ```

2. Get a list of attributes:

   ```
   Vector vStat = (Vector) session.getAttribute("attrlist");
   ```

3. List the attribute names and entry fields for values of search:

   ```
   <TR><TH>Attribute</TH><TH>Search Value</TH></TR>
   <%
   int vSize = 0;
   if (!(vStat == null)) {
     vSize = vStat.size();
   }
   for (int i = 0; i < vSize; i++) {
     String element = (String) vStat.elementAt(i);
     int parse = element.indexOf(";");
     String name = element.substring(0, parse);
     String displayName = element.substring(parse + 1);
     %>
     <tr><td><%=displayName%></TD><TD><input type=text size=45 name=<%=name%>
   Value=""></TD></TR>
     <%}%>
   ```

For the complete source code of RBSearchEntry.jsp, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-12 on page 561.

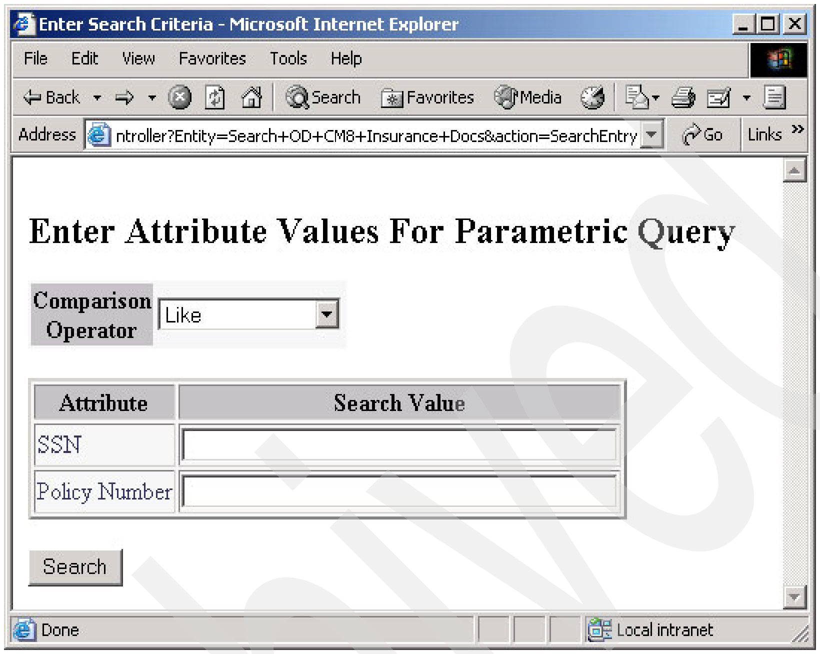
RBSearchEntry generated HTML

The sample generated HTML file from RBSearchEntry.jsp is shown in Figure 7-5.

Example 7-5 RBSearchEntry.jsp generated HTML

```
<form method="POST" action="RBController" target=_top>
<input type=hidden name=action value="Search">
<TABLE bgcolor="#80ffff">
<TR><TH>Comparison<br>Operator</TH><td>
<SELECT NAME="_compop" SIZE=1 >
<OPTION  SELECTED value=1> Equals
<OPTION  value=2> Not Equals
<OPTION  value=3> Greater Than
<OPTION  value=4> Less Than
<OPTION  value=13> Greater or Equal
<OPTION  value=14> Less or Equal
<OPTION  value=5> Like
</SELECT></td></tr>
</table><br>
<TABLE border=1 bgcolor="#80ffff">
<TR><TH>Attribute</TH><TH>Search Value</TH></TR>
<tr><td>Building permit number</TD><TD><input type=text name=BP_NUMBER
Value=""></TD></TR>
<tr><td>Document Description</TD><TD><input type=text name=BP_DOC_DESC
Value=""></TD></TR>
<tr><td>Submission date</TD><TD><input type=text name=BP_SUB_DATE
Value=""></TD></TR>
</table> <br>
<input type=submit value=Search></form></body>
```

The sample output of RBSearchEntry.java and RBSearchEntry.jsp is shown in Figure 7-4.


Enter Search Criteria - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address ntroller?Entity=Search+OD+CM8+Insurance+Docs&action=SearchEntry
Go Links

Enter Attribute Values For Parametric Query

Comparison Operator	Like

Attribute	Search Value
SSN	
Policy Number	

Search

Figure 7-4 Sample output of RBSearchEntry.java and RBSearchEntry.jsp

7.3.12 Sorting arrays of beans

Many methods in the Information Integrator for Content beans return arrays of beans for the information you want. From the system, these arrays come back in a random order. You should sort them before you present the results to the end user.

RBSortArray.java

RBSortArray.java implements the dkSort abstract class but overrides the *getKey* method to return the value of the property used to sort on.

The RBSortArray.java.getKey() method gets the following key based on the result set:

- The entity's description if it is dealing with CMEntity object
- The attribute's description if it is dealing with CMAttribute object
- The search template name if it is dealing with CMSearchTemplate object
- The search criterion name if it is dealing with CMSTCriterion object

Example 7-6 shows a code snippet of the RBSortArray.getKey() method.

Example 7-6 RBSortArray.getKey() method code snippet

```
public Object getKey(Object anObject) throws DKUsageError
{
   if (anObject != null && (anObject instanceof CMBEntity))
      return ((CMBEntity) anObject).getDescription();
   if (anObject != null && (anObject instanceof CMBAttribute))
      return ((CMBAttribute) anObject).getDescription();
   if (anObject != null && (anObject instanceof CMBSearchTemplate))
      return ((CMBSearchTemplate) anObject).getName();
   if (anObject != null && (anObject instanceof CMBSTCriterion))
      return ((CMBSTCriterion) anObject).getName();
```

The application calls the RBSortArray class's sort method. The RBSortArray.sort() method calls quicksort to actually sort the array.

For the complete source code of RBSortArray.java, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-13 on page 562.

7.3.13 Conducting a search: CMBQueryServices

CMBQueryServices provides the ability to conduct searches on the back-end server based on entities and attribute values. There are three ways in which this can be accomplished:

- **Query string:** This type of search takes a query string and an optional set of query parameters. The syntax of the query string and possible parameters are documented in the Java API documentation. Using this style of search, any query that is supported by the server can be performed; however, the building up of the query is dependent on the type of server. We use this way to search for Content Manager Version 8 documents.
- **Template based queries:** For servers that support search templates, which includes Federated and OnDemand (folders), a template-based search uses an administrator-defined set of search criteria, with allowed operators and values. This style of query takes a CMBSearchTemplate instance whose criteria have been filled in with selected operators and values and performs a search. We use this way for searching Federated and OnDemand.
- **Entity/attribute based query:** This type of query takes the name of an entity, and the names, operators, and values of attributes. A method on CMBQueryService, generateQueryExpression, converts this into a query string. This style of query supports the building of basic, search-template-like, searches to servers that do not support search templates. It also allows the building of a query in a server independent way.

Searches can also be run in one of three modes:

- **Synchronous:** This type of search sends the query to the server and obtains all results before returning. This is the simplest form of search, but it may require a long time to execute depending on the number of results obtained.
- **Asynchronous:** This type of search sends the query to the server and then immediately returns. A secondary thread is created internally, and callbacks are made to a callback class provided to indicate that results have arrived. More than one call to the callback may occur, providing partial results.
- **Cursored:** This type of search sends the query to the server and obtains a cursor which can be used to retrieve the results from the server when needed. This type of search is similar to asynchronous but it requires the application to "pull" the results from the server, rather than having them "pushed" by the server via a callback. This means that additional resources are required on the server for maintaining the search results.

Some of the CMBQueryService methods are as follows:

- String generateQueryExpression(String entityName, CMBQueryService.QueryCriterion[] criteria, short queryType, boolean matchAllCriteria) — Generate a query expression from the given datastore type.
- CMBQueryService.QueryParameter[] generateQueryParams(String ODentityName) — Generate basic query parameters for the current datastore type.
- generateQueryString() — Obtain a query string for an entity with operators and attribute values. These are entity/attribute style searches.
- int getCallbackThreshold() — Get the callback threshold value.
- CMBQueryService.QueryParameter[] getQueryParameters() — Get the additional query parameters (for query string only).
- String getQueryString() — Get the query string currently being used for query service.
- Object getResults() — Get a collection of result items from synchronous searches. Note that for asynchronous search, CMBSearchReplyEvents are also generated as results are obtained from the server. The frequency of these events is set using the setCallbackThreshold method.
- void runQuery() — Start a search.
- void setAsynchSearch(boolean asynch) — Set the query option to be either synchronous or asynchronous.
- void setMaxResults(int max) — Set the maximum number of hits returned from each query.
- setQueryString — This is a query string-based search.

7.3.14 Processing search results: CMBSearchResults

CMBSearchResults manages the results generated by the CMBQueryService and CMBSearchTemplate beans. It has methods for returning arrays of CMBItem beans, each bean representing an item that matched the search criteria. It has its own sort method that you may supply a particular attribute to sort on.

As mentioned in the previous section, for asynchronous searches, CMBSearchResults listens to CMBSearchReplyEvents from CMBQueryService or CMBSearchTemplate. Alternatively, for synchronous searches, results can be obtained from getResults() method of CMBQueryService.

Here are some of the CMBSearchResults methods:

- int getCount() — Get the total hit count.
- CMBHitItem getCurrentHitItem() — Get the current hit item from the search result.
- CMBHitItem[] getHitItem() — Get a list of all hit items from the search result.
- CMBItem getItem(String pidString) — Get an item with a hit item's ID.
- CMBItem[] getItems() — Get a list of all items from the search result.
- String getPidString() — Get the current hit item's ID.
- void sort(String attribute, boolean ascending) — Sort the search results.

RBSearch.java

RBSearch.search is called from the RBSearchEntry JSP with the name of the entity or search template to use, the boolean operator to use, (=, >, <, etc.) and the attribute names and values to use.

RBSearch.java does the following tasks:

1. Decide, based on the connection's server type, whether to use an entity and attribute search (for Content Manager Version 8) or a search template search (for Federated and OnDemand).
   a. For Content Manager Version 8, construct a valid XPATH query string and perform a search.
   b. For OnDemand and the Federated, set the search criteria of the selected search template and perform the search.
2. Conduct a search.
3. For either case, use the CMBSearchResult bean to sort the results based on an attribute, and add the hit list to a session parameter "ItemList" to be displayed by the RBItemList.jsp.

Example 7-7 shows a code snippet for building an XPATH query string for Content Manager Version 8 entity searches.

Example 7-7 RBSearch.java - Building CM V8 XPATH query string code snippet

```
if (connection.getDsType().equals("ICM")) {
      // booleanOperator will take the short opCode and return a character
      // string for the boolean operator (=, >=, <=, etc)
      String opStr = booleanOperator(opCode);
      String condString = "";

      // Loop thru the attributes of the CMBEntity and construct the
      // conditional part of the XPATH query string. It will be of the form
      // ‘[@attr_name opStr “value”]
      CMBEntity entity = connection.getSchemaManagement().getEntity(strEntity);
      CMBAttribute attrs[] = entity.getAttributes();
      for (int i = 0; i < attrs.length; i++) {
      // search values are passed into the servlet with the name of the
      // attribute they are for
        String dataValue =
request.getParameter(java.net.URLEncoder.encode(attrs[i].getName()));
        if (dataValue != null && dataValue.length() > 0) {
          if (condString.equals(""))
            condString = "[@" + attrs[i].getName() + opStr + "\"" + dataValue +
"\"";
          else
            condString += " AND @" + attrs[i].getName() + opStr + "\"" +
dataValue + "\"";
        }
      }

      // If an attribute value was provided to be searched on,
      // we need to close the bracket.
      if (!condString.equals(""))
        condString += "]";
      String queryString = "/" + strEntity + condString;
```

Example 7-8 shows a code snippet for conducting a search for Content Manager Version 8.

Example 7-8 RBSearch.java - Conducting a search for CM V8 code snippet

```
// Create a CMBQueryService (queryBean) on the connection, and prepare it for
sync
// searching
CMBQueryService queryBean = connection.getQueryService();
queryBean.setAsynchSearch(false);

queryBean.setQueryString(queryString, CMBBaseConstant.CMB_QS_TYPE_XPATH);
System.out.println("Performing " + queryString);
queryBean.runQueryWithCursor();
```

Note: Content Manager version 8 uses different values internally for the names of entities, attributes, and Document Routing constructs than the display names most end users would want to see. This is because the internal names need to conform to column names for DB2 (no spaces, limited length, and limited number of special characters), and the method on the CMBConnection bean *setDisplayEnabled(Boolean).*

Example 7-9 shows a code snippet for retrieving results for Content Manager Version 8.

Example 7-9 RBSearch.java - Retrieve results for CM V8 code snippet

```
// setDisplayEnabled affects the CMBItem objects that will be returned in the
result set
// by having the attribute names be the display names when retrieved.
connection.setDisplayNamesEnabled(true);

CMBSearchResults resultBean = new CMBSearchResults();
resultBean.setConnection(connection);
resultBean.newResults(queryBean.getResults());
// Sort on the values in the first defined attribute
resultBean.sort(entity.getAttributeNames()[0], true);

System.out.println("Results " + resultBean.getCount());
session.setAttribute("itemList", resultBean.getItems());
connection.setDisplayNamesEnabled(false);
RBConnectionPool.freeConnection(connection);}
```

Example 7-10 shows a code snippet for setting up a search template for OnDemand and a federated server.

Example 7-10 RBSearch.java - Set up the search template code snippet

```
else if (connection.getDsType().equals("OD") ||
connection.getDsType().equals("Fed")) {
// Create a CMBSearchTemplate (searchTemp) on the schema bean using the passed
in
// entity name
CMBSearchTemplate searchTemp =
      connection.getSchemaManagement().getSearchTemplate(strEntity);

// Set the search criteria of the template based on the parameters passed in
Vector critList = searchTemp.getSearchCriteria();
for (int i = 0; i < critList.size(); i++) {
   CMBSTCriterion critObj = (CMBSTCriterion) critList.elementAt(i);
   String dataValue =
request.getParameter(java.net.URLEncoder.encode(critObj.getName()));
```

```
   if (dataValue != null && dataValue.length() > 0) {
      critObj.setOperator(opCode);
      critObj.setValue(dataValue);
   }
}
```

Example 7-11 shows a code snippet for executing the search template-based search and retrieving the search results.

Example 7-11 RBSearch.java - Execute search template-based search; retrieve results

```
// Run the search, and put the results to the session object
searchTemp.setAsynchSearch(false);
searchTemp.runQuery();
CMBSearchResults resultBean = new CMBSearchResults();
resultBean.setConnection(connection);
resultBean.newResults(searchTemp.getResults());

session.setAttribute("itemList", resultBean.getItems());
```

For the complete source code of RBSearch.java, refer to the downloaded file from the additional material provided by this redbook, or view the source shown in Example B-14 on page 565.

7.3.15 Representing items: CMBItem

The CMBItem bean represents an individual document or individual piece of content in the back-end server. It has attributes with metadata values. It is of a particular type and may have other properties based on the type of server it comes from.

The CMBItem has a persistent ID (PID) associated with it that identifies its uniqueness within the system, or in the case of a federated system, its uniqueness across multiple systems. The information in the PID contains:

- Back-end Repository Name (such as Library Server Name, OnDemand system name, and Federated Database name).
- Repository Type (such as “ICM”, “OD”, or “Fed”).
- Object Type (such as entity name).
- PrimaryID (how the back-end repository uniquely identifies an object within itself). For Content Manager Version 8, this is the twenty-six character ItemID and Component ID.
- Version (for back-end repositories that support versioning of content).

This information can be represented in a single character string, called a pidString, that can be passed from one servlet to another and recreate a CMBItem bean representing the same content on the server:

```
String PIDString ="87 3 ICM8 ICMNLSDB6 Claims59 26" +
"A1001001A02E08B24249G9344318 A02E08B24249G934431 14 1003 ";
CMBItem item = new CMBItem(PIDString);
```

These are some of the CMBItem methods:

- int getAttrCount() — Get the number of attributes for this item.
- CMBItem[] getAttrItems(String attr) — Get the attribute items for this request attribute name.
- String[] getAttrName() — Get a list of attribute names of this item.
- String getAttrValue(String attr) — Get the String attribute value of a given attribute name.
- String[] getAttrValues() — Get a list of attribute values of this item.
- String getName() — Get the name of the item.
- CMBItem getParent() — Get the parent of the item.
- String getPidString() — Get the item ID.

7.3.16 Displaying item information

In our application, RBItemList.jsp is used to display information from an array of CMBItem beans, generated by either search results, items in a folder, or work items in a worklist. Because this is different than just formatting a string from a vector, and the PID string needs to be passed to the subsequent action, we need to develop some code within the JSP using the methods of CMBItem.

RBItemList.jsp

RBItemList.jsp display entity name, attribute name, and attribute values as follows:

- Entity type (document, folder, or item), which is kept as a small int property in CMBItem, is used as part of a switch statement to display the type to the user.
- Attribute names and attribute values are kept in two equal length arrays, so they can be referenced in a loop.

The Java scriptlets presented in Example 7-12, Example 7-13, and Example 7-14 are in the RBItemList.jsp. Working together, they show the item type and attribute values of a list of CMBItem beans.

Example 7-12 shows a scriptlet, using the switch statement to get and return item type in RBItemList.jsp.

Example 7-12 RBItemList.jsp - scriplet to return item type

```
private static String itemType(CMBItem item)
throws CMBException{
    // Return the item type of the item
    switch (item.getItemType()) {
      case CMBBaseConstant.CMB_TYPE_DOCUMENT :
        return("Document");
      case CMBBaseConstant.CMB_TYPE_FOLDER :
        return("Folder");
      case CMBBaseConstant.CMB_TYPE_ITEM :
        return("Item");
      default :
        return("Undefined");
    }
}
```

Example 7-13 shows a scriplet, displaying entity name, and going through a loop to get and display attribute names and attribute values in RBItemList.jsp.

Example 7-13 RBItemList.jsp - scriplet to display entity and attribute names and values

```
private static String itemAttrString(CMBItem item)
throws CMBException{
   String itemDesc = item.getEntityName()+" ";
   // Concat the attribute names and values onto the string
   for (int i = 0; i < item.getAttrCount(); i++){
      if (i>0)
        itemDesc += ", ";
      itemDesc += item.getAttrName(i) + "=" + item.getAttrValue(i);
   }
   return(itemDesc);
}
```

Example 7-14 shows a scriplet for processing an array of CMBItem beans in a loop, in the RBItemList.jsp. For each IBMItem, it creates an input field of hidden name for PID value. If the item type is a folder, in the drop-down action box, it puts in a Folder Open option, otherwise, it puts a View option. In addition, if this is a Content Manager Version 8 items, it hard codes “Start Process” and “Advance Process” options in the drop-down action box for user to select.

Example 7-14 RBItemList.jsp - scriplet for processing an array of CMBItems in a loop

```
// For every CMBItem in the array:
for (int i = 0; i < items.length; ++i) {%>
  <TR><td><%=itemType(items[i])%></td><TD>
  <%=itemAttrString(items[i])%></td><TD>
  <form method=POST action=RBController target=_top>
  <input type=hidden name=PID
value=<%=java.net.URLEncoder.encode(items[i].getPidString())%>>
  <%if (wpItems != null) { %>
  <input type=hidden name=wpPID
value=<%=java.net.URLEncoder.encode(wpItems[i].getPidString())%>>
  <%}%>
  <table><tr><td>
  <SELECT NAME=action SIZE=1 >
  <% if (items[i].getItemType() == CMBBaseConstant.CMB_TYPE_FOLDER) { %>
    <OPTION value="FolderOpen">Folder Open
  <%} else {%>
     <OPTION SELECTED value="View">View
  <%}%>

  <% if (session.getAttribute("dsType").equals("ICM")) { %>
    <OPTION value="SelectProcess">Start Process
    <OPTION value="SelectAdvance">Advance Process
  <%}%>
  </SELECT></td><td>
  <input type=submit value='Go'>
  </td></tr></table></form></td></tr>
<%}
```

RBItemList.jsp generated HTML

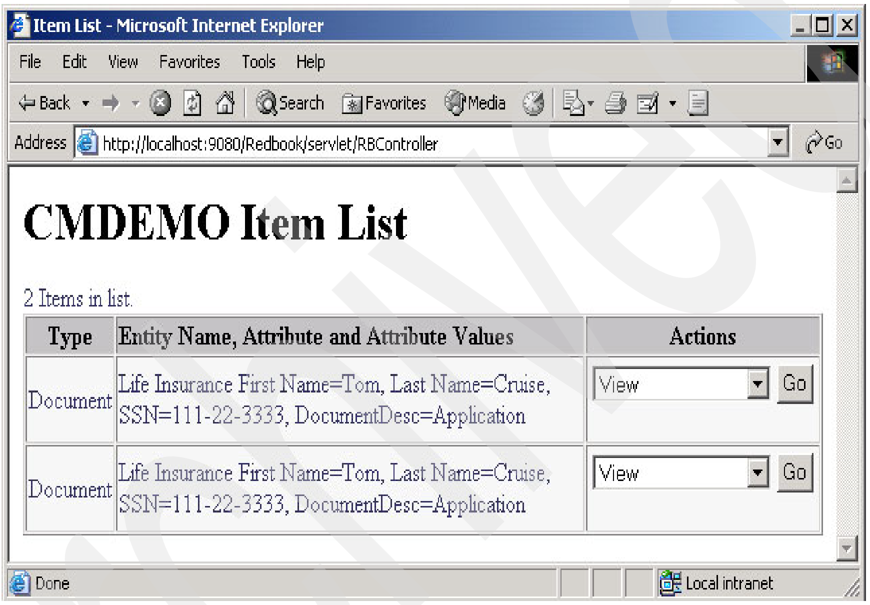
For each of the items in the table, the sample output of the RBItemList.jsp generated HTML is similar to that of Example 7-15.

Example 7-15 Sample output of RBItemList.jsp generated HTML

```
<TR><td>Document</td>
<TD>Life Insurance First Name=Tom, Last Name=Cruise, SSN=111-22-3333,
DocumentDesc=Application</td><TD>
<form method=POST action=RBController target=_top>
<input type=hidden name=PID
value=91+3+ICM6+CMDEMO13+LifeInsurance59+26+A1001001A04A07B72850H3237318+A04A07
B72850H323731+14+1219>
<table><tr><td>
  <SELECT NAME=action SIZE=1 >
  <OPTION SELECTED value="View">View
  <OPTION value="SelectProcess">Start Process
  <OPTION value="SelectAdvance">Advance Process
  </SELECT>
  </td><td><input type=submit value='Go'></td></tr>
</table></form></td></tr>
```

The FORM in this HTML will call the RBController servlet and pass it to the PID string with the name of an action to perform. With the PID string, the application can recreate the CMBItem bean for the object and perform the action on it.

Figure 7-5 shows the sample output of a search listed in a table with a follow-on actions.


Item List - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/Redbook/servlet/RBController
Go
Done
Local intranet

CMDEMO Item List

2 Items in list.

Type	Entity Name, Attribute and Attribute Values	Actions
Document	Life Insurance First Name=Tom, Last Name=Cruise, SSN=111-22-3333, DocumentDesc=Application	View Go
Document	Life Insurance First Name=Tom, Last Name=Cruise, SSN=111-22-3333, DocumentDesc=Application	View Go

Figure 7-5 Sample output of a search listed in a table with follow-on actions

From this Item List, the end user can read the attributes of the results to determine which result they are interested in. They would select an action (View, Folder Open, Start Process, or Advance Process) for it, then click **Go**.

For the complete source code of RBItemList.jsp, refer to the downloaded file from the additional material provided by this redbook or view the source shown in Example B-15 on page 568.

7.3.17 Managing content: CMBDataManagement

The CMBDataManagement bean provides data manipulation functions on items stored on content servers. This includes creating, retrieving parts, updating, and deleting documents and folders, as well as associated content, annotations, and note parts. It also includes check-in/check-out/unlock functions on data items.

To use this bean:

1. Set the current data item using setDataObject. This method requires an instance of CMBItem.
2. Invoke other methods to create, retrieve, update, delete, check-in, check-out, or unlock the data item.

These are some of the CMBDataManagement methods:

- void addContent (CMBObject obj) — Add a new content part to the current document at the end of the parts list.
- void addLink (CMBItem item, String linkType) — Add a link.
- void addNoteLog(CMBObject obj) — Add a new note log part to the current document.
- void addViewData (CMBViewData viewDataObj) — Add a new view data object to current document (for OnDemand documents only).
- void checkIn() — Check the current document back into the content server.
- void checkOut() — Check the current document out of the content server.
- void createItem (String entityName) — Create a new item and put it in the given entity name on the server.
- void createResourceItem (String className, String entityName) — Create a new resource item and put it in the given entity name on the server.
- void getContentCount() — Get the number of content parts for the current document.
- CMBItem getDataObject() — Get the data object.
- String getItemMimeType() — Get the mime content type for the item.
- short getItemType() — Get the item type for the content item.
- CMBPrivileges getPrivilege() — Get the privilege of the current document.
- void reindexItem() — Move the item from the current entity to a new entity.
- void retrieveItem() — Retrieve the latest version of the current item from the server.

For every add or create method, there is usually a corresponding delete, get, and update method which we do not show here, but you should be aware of them.

Retrieving item information

Our application uses the PID string as a parameter passed into the servlets to create a CMBItem bean. The information in this bean will consist of the information from the PID string, plus the names and values of user defined attributes from the connected server.

```
// Create a CMBItem (itemBean) using the passed in PID and set its connection.
CMBItem itemBean = new CMBItem(URLDecoder.decode(request.getParameter("PID")));
itemBean.setConnection(connection);
```

Creating new items

The Java beans are restricted to perform only the functions allowed by the back-end repository the connection bean is connected to. Creating individual items is not a function of the OnDemand or Federated repositories.

Although not a part of our sample application, Example 7-16 shows the sample code for creating a new item, with one part, in Content Manager Version 8. Note that the file name and entity name are hard coded for simplicity.

Example 7-16 Creating a new item with one part sample code

```
CMBConnection connection = RBConnectionPool.getConnection(request);
if (connection.getDsType().equals("ICM")) {
   CMBDataManagement dataManagement = connection.getDataManagement();
   String filename = "C:\\test.gif";
   String entity = "PoliceReport";


   // Construct a CMBItem object to represent the new item
   CMBItem item = new CMBItem();
   item.setConnection(dataManagement.getConnection());
   item.setItemType(CMBBaseConstant.CMB_TYPE_DOCUMENT); // or FOLDER
   item.setServerType(dataManagement.getConnection().getDsType());
   item.setServerName(dataManagement.getConnection().getServerName());
   item.setEntityName(entity);

   item.addAttr("ClaimNum", "3-4567", CMBBaseConstant.CMB_DATATYPE_VSTRING);
   dataManagement.setDataObject(item);

   // Create a new CMBObject, and set its propertise and content
   CMBObject obj = new CMBObject();
   obj.setMimeType("image/gif");
   obj.setPartType("ICMBASE");
   obj.setDataFile(filename); // CM 8 Only
   dataManagement.setDataObject(item);
   dataManagement.addContent(obj);

   // Create the item on the server
   dataManagement.createItem(entity);
}
RBConnectionPool.freeConnection(connection);
```

Opening Content Manager folders

Our application allows for the opening of Content Manager folders, and presents the member list in the RBItemList.jsp. The method on the CMBItem bean is *listFolderItems(),* but this returns a Java Vector of CMBItem instead of an array. We therefore build an array to pass it to RBItemList.jsp. The beans in the member list already have their system information.

Example 7-17 shows a code snippet for RBFolderOpen.java, where we get a list of items and build an array to store the array into itemList to be used by RBItemList.jsp later.

Example 7-17 RBFolderOpen.java - Code snippet to get list of member items

```
// If the item is of type CMB_TYPE_FOLDER, get the vector (itemList)
if (itemBean.getItemType() == CMBBaseConstant.CMB_TYPE_FOLDER) {
   Vector itemList = itemBean.listFolderItems();
   if (itemList.size() > 0) {
        Iterator ild_items = itemList.iterator();
        CMBItem[] items = new CMBItem[itemList.size()];

        // Iterate through the itemlist, and add to the CMBItem array
        // to be passed to the JSP.
        int i = 0;
        while (ild_items.hasNext()) {
          items[i] = (CMBItem) ild_items.next();
          i++;
}
        session.setAttribute("itemList", items);
```

For the complete source code of RBFolderOpen.java, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-16 on page 570.

7.3.18 Viewing content: CMBDocumentServices

Our application uses the document services bean, CMBDocumentServices, to display the image in a Web browser. It is a very powerful bean, and can be used by all supported back-ends to display their content. The document services provide the ability to:

- **Render:** Java image objects for document pages can be obtained from the document services. These images, including their graphical annotations, can be used for viewing of documents in AWT and Swing-based applications.
- **Convert:** A document or pages from the document may be converted to other forms. The form is typically GIF, JPEG, HTML, or other browser displayable forms. This conversion is useful in a middle-tier Web application where applets or plug-ins for viewing are not an acceptable solution.
- **Reconstitute:** Documents composed of multiple parts on the server can be combined to form the original document. This is useful in exporting and e-mailing.

Associated with the document services, the CMBDocument bean represents the entity created when loading a document using CMBDocumentServices. CMBDocument is a container for the pages in the document, and it also allows you to query and set properties that control a document's characteristics. The CMBPage bean provides a representation of a particular page in a document. The functionality in this class allows you to specify and control the properties of a set of renderable images that can be generated for the page.

Here are some of the CMBDocumentServices methods:

- void dropAllDocuments () — Terminate processing of all documents by the document services.
- CMBAnnotationServices getAnnotationServices() — Return the instance of CMBAnnotation Services.
- Properties getConversionProperties() — Return the conversion properties.
- CMBDocument[] getDocuments() — Return a list of documents being processed.
- String[] getPreferredFormats() — Return preferred formats for the converted documents.
- String[] getPreferredPageFormats() — Return preferred page formats for converted pages of documents.
- CMBDocument loadDocument (CMBItem item) — Initiate processing of a document by document services.

In our application, we use the CMBDocumentServices to display the first page of the document in a very simple browser window. You may note that the RBController is not redirecting control to a JSP.

To use CMBDocumentServices:

1. Construct the document services bean.
2. Set bean properties for document engines and conversion preferences.

3. Connect the document services bean to an instance of a data management bean using setDataManagement (datamanagement).
4. Obtain an item of type CMBItem for the document. Other non-visual beans that perform searching and worklist queries can create CMBItems.
5. Call loadDocument (item) to initiate document services for the document. This returns a CMBDocument object.
6. If pagination is supported, use the getPages() method on CMBDocument to obtain a CMBPage object for a particular page. Otherwise, use CMBDocument.write to convert the entire document.
7. Use CMBPage.getImage() method on CMBPage to write the page as an image or HTML file.
8. Call dropDocument (item) when document services on the document are no longer needed.

RBView.java

RBViewer.java is called by RBController to display a document for viewing. Its view_doc method and viewPage methods do the displaying work.

Example 7-18 shows a code snippet for the view_doc method.

Example 7-18 RBView.java - view_doc method code snippet

```
dataManagement.setDataObject(item);
CMBDocument doc = documentServices.loadDocument(item);
if (doc.getCanPaginate()) {
        CMBPage[] pages = doc.getPages();
        doc.setPreferredScale(scale);
        doc.setRotation(rotation);
        doc.setShowAnnotations(annotate);
        for (int j = 0; j < doc.getPageCount(); j++) {
          viewPage(response, pages[j]);
        }
      } else {
        // It is converted to a browser-friendly format if possible.
        String mimeType = doc.getWriteMimeType();
        try {
          response.setContentType(mimeType);
          doc.write(response.getOutputStream());
```

For the complete source code of RBView.java, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-17 on page 571.

7.3.19 Content Manager Version 8 Document Routing system

In this section, we explain the CM Version 8 Document Routing system, including a set of associated beans, and we develop our sample application to provide the Document Routing feature.

Content Manager Version 8 Document Routing review

In 2.3, “Document Routing” on page 37 we cover Document Routing. Here, we provide a brief review of the Document Routing system. Basically, there are four key elements involved:

- **Work Node:** A step or an activity in a process. This is the location where an item waits to be processed.
- **Process:** An ordered set of work nodes.
- **Worklist:** A view for the end user showing the items in the work nodes.
- **Work package:** The work entity that actually travels through the process visiting the work nodes.

The work node, process, and worklist are set up by the system administrator. A work package is created when a Content Manager Version 8 item is started in a process. Usually, the end user opens a worklist to view and process work package. The worklist is mapped to one or more work nodes, and the user sees all the work packages in these nodes.

Note: The work package is not the actual item going through the process, but a separate item associated with it. Use the method *getItemPidString()* to reference the actual item.

Beans associated with Document Routing

The Document Routing system has a set of beans to access all parts of the system.

CMBDocRoutingQueryServiceICM	Lists Content Manager Document Routing data objects. It provides the methods for listing processes, worklists, work packages, and work nodes.
CMBWorkNodeICM	Represents a work node in Document Routing.
CMBProcessICM	Represents a process or a workflow instance in Document Routing.

CMBWorkListICM	Represents a worklist in Document Routing.
CMBWorkPackageICM	Represents a work package definition for Document Routing.
CMBDocRoutingDataManagementICM	Manages Content Manager Document Routing objects. Provides the methods for managing a process, including starting, terminating, continuing, suspending and resuming a process. It also provides methods for managing a work package including setting the work package container data, the owner, and priority.

RBDocRouting.java

Our sample application allows the end user to start an item in a process, list worklists, open a worklist, view the item in a work package, and advance a work package to its next work node or to completion. These functions are provided by separate methods in RBDocRouting.java.

RBDocRouting.java — Selecting a process

To start an item on a process, the process needs to be selected, and then the work package needs to be created. This is done by listing the processes available and displaying it in the RBProcessEntry.jsp. The action will then start it in the process.

Example 7-19 shows the selectProcess() method of the RBDocRouting.java.

Example 7-19 RBDocRouting.java - Code for selectProcess method

```
public static void selectProcess(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // List the worklists names for display in the jsp.  The names need
    // to be passed to the next servlet for opening
    String processNames[] = drQService.getProcessNames();

    for (int i = 0; i < processNames.length; ++i) {
      vector.add(processNames[i]);
    }
    session.setAttribute("list", vector);

    CMBDataManagement dataManagement = connection.getDataManagement();
    // Create a CMBItem (itemBean) using the passed in strPID.  Set its
    // connection, and hook it up to the dataBean
    CMBItem itemBean = new
CMBItem(java.net.URLDecoder.decode(request.getParameter("PID")));
    itemBean.setConnection(connection);
    dataManagement.setDataObject(itemBean);
    dataManagement.retrieveItem();
    session.setAttribute("item", itemBean);
    RBConnectionPool.freeConnection(connection);
  }
```

RBProcessEntry.jsp

RBProcessEntry.jsp displays the attributes of the item the same way as RBItemList.jsp does, with a selection drop-down box with the process names.

Example 7-20 shows a code snippet for RBProcessEntry.jsp.

Example 7-20 RBProcessEntry.jsp - Code snippet

```
<body>
<h1><%=session.getAttribute("server")%> Select Process</h1>
<TABLE width="100%" cellpadding="1" cellspacing="1" border="1"
bgcolor="#f7f7f7">
<tr><TH>Type</th><th align="left">Entity Name,  Attribute and Attribute
Values</th></tr>
<%CMBItem item = (CMBItem) session.getAttribute("item");%>
<TR><td><%=itemType(item)%></td><TD>
<%=item.getEntityName()%>&nbsp<%=itemAttrString(item)%>
  </td></tr></table>
  <br>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="StartProcess">
<input type=hidden name=PID
value=<%=java.net.URLEncoder.encode(item.getPidString())%>>
<SELECT NAME=process SIZE=5 >
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) { %>
<OPTION value="<%=vStat.elementAt(i)%>"><%=vStat.elementAt(i) %>
<%}
```

```
%>
</SELECT><br><br><input type=submit value='Select'></form>
</body>
```

For the complete source code of RBProcessEntry.jsp, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-19 on page 577.

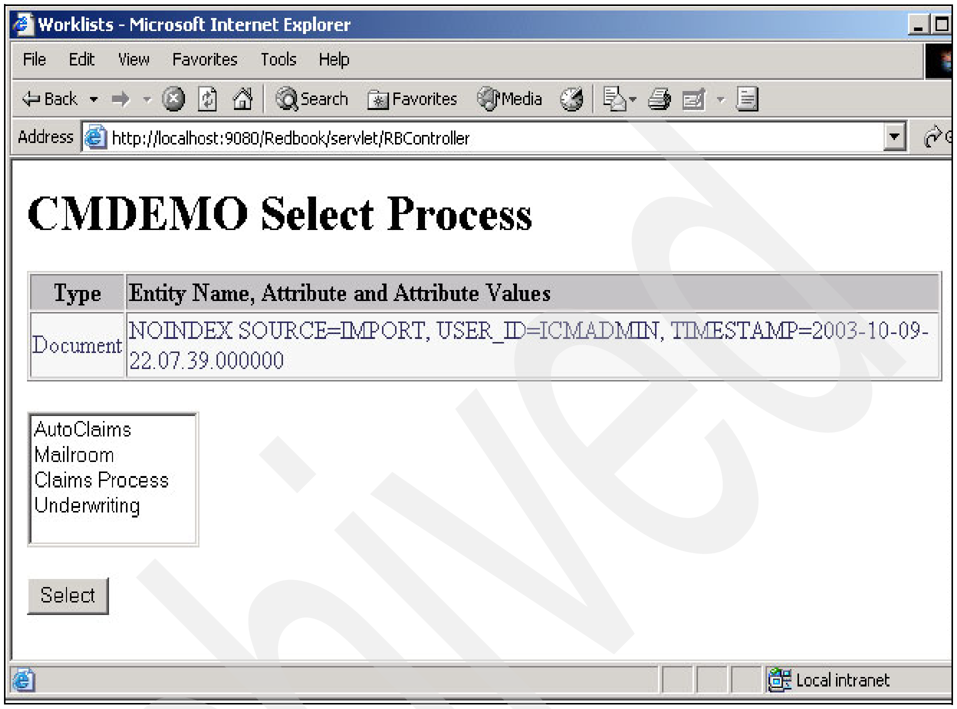
RBProcessEntry.jsp generated HTML

The sample output of RBProcessEntry.jsp generated HTML is shown in Example 7-21.

Example 7-21 Sample output of RBProcessEntry.jsp generated HTML

```
<body>
<h1>CMDEMO Select Process</h1>
<TABLE width="100%" cellpadding="1" cellspacing="1" border="1"
bgcolor="#f7f7f7">
<tr><TH>Type</th><th align="left">Entity Name,  Attribute and Attribute
Values</th></tr>

<TR><td>Document</td><TD>
NOINDEX&nbspSOURCE=IMPORT, USER_ID=ICMADMIN,
TIMESTAMP=2003-10-09-22.07.39.000000
  </td></tr></table> <br>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="StartProcess">
<input type=hidden name=PID
value=84+3+ICM6+CMDEMO7+NOINDEX59+26+A1001001A03J09C20739F5506718+A03J09C20739F
550671+14+1000>

<SELECT NAME=process SIZE=5 >
<OPTION value="AutoClaims">AutoClaims
<OPTION value="Mailroom">Mailroom
<OPTION value="Claims Process">Claims Process
<OPTION value="Underwriting">Underwriting
</SELECT><br><br><input type=submit value='Select'></form>
</body>
```

The sample screen output of RBDocRouting and RBPorcessEntry.jsp is shown in Figure 7-6.


Worklists - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/Redbook/servlet/RBController

CMDEMO Select Process

Type	Entity Name, Attribute and Attribute Values
Document	NOINDEX SOURCE=IMPORT, USER_ID=ICMADMIN, TIMESTAMP=2003-10-09-22.07.39.000000

AutoClaims
Mailroom
Claims Process
Underwriting

Figure 7-6 Sample screen output of RBDocRouting.java and RBProcessEntry.jsp

RBDocRouting.java — Starting an item in a process

Starting an item in a process when you have the item's PID and the name of the process is accomplished by calling startProcess method on the CMBDocRoutingDataManagementICM bean.

Example 7-22 shows a code snippet of startProcess() method of RBDocRouting.java.

Example 7-22 RBDocRouting.java - Code snippet for startProcess method

```
public static void startProcess(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    CMBDocRoutingDataManagementICM drManagement =
connection.getDocRoutingDataManagementICM();

    String wpItemID =
      drManagement.startProcess(
        (String) request.getParameter("process"),
        (String) java.net.URLDecoder.decode(request.getParameter("PID")),
        1, "");
    System.out.println("New Packet " + wpItemID);
    RBConnectionPool.freeConnection(connection);
  }
```

RBDocRouting.java — Listing worklists

The RBDocRouting.list() method lists the worklists.

Example 7-23 shows the list() method of RBDocRouting.

Example 7-23 RBDocRouting.java - Code snippet for list method

```
public static void list(HttpServletRequest request) throws CMBException,
Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // List the worklists names for display in the jsp.  The names need
    // to be passed to the next servlet for opening
    String worklists[] = drQService.getWorkListNames();

    for (int i = 0; i < worklists.length; ++i) {
      vector.add(worklists[i]);
    }
    session.setAttribute("list", vector);
    RBConnectionPool.freeConnection(connection);
  }
```

RBWorklist.jsp — Listing worklists

RBWorklist.jsp, called by the RBController servlet, puts the name of the worklists from the vector to the selection box.

Example 7-24 shows a code snippet of RBWorklist.jsp.

Example 7-24 RBWorklist.jsp - Code snippet

```
<body>
<h1><%=session.getAttribute("server")%> Worklists</h1>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="OpenWorklist">
<SELECT NAME=worklist SIZE=5 >
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) { %>
<OPTION value="<%=vStat.elementAt(i)%>"><%=vStat.elementAt(i) %>
<%}%>
</SELECT><br><br><input type=submit value='Select'></form>
</body>
```

For the complete source code of RBWorklist.jsp, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-20 on page 578.

RBDocRouting.java — Open a worklist

Opening a worklist involves the Document Routing Query Services *getWorkpackages(name)* to get a Collection of all work packages in the worklist with the given name. It is the work package that actually flows through the process, but the end user is probably more interested in the document or folder that is contained in the work package. For this reason we need to create an array of CMBItem beans to be displayed by our RBItemList.jsp, as well as a corresponding array of the work packages in case the end user wants to advance one in a work process.

Example 7-25 shows a code snippet for the openWorklist method of RBDocRouting.java.

Example 7-25 RBDocRouting.java - Code snippet for openWorklist() method

```
public static void openWorklist(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    // Get the name of the worklist we are opening
    String worklist = request.getParameter("worklist");
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // Get the work packages that are in the worklist for display
    Collection workpackages = drQService.getWorkPackages(worklist, "");
    Iterator wpIterator = workpackages.iterator();

    int i = 0;
    CMBWorkPackageICM[] wpItems = new CMBWorkPackageICM[workpackages.size()];
    CMBItem[] items = new CMBItem[workpackages.size()];

    while (wpIterator.hasNext()) {
      wpItems[i] = (CMBWorkPackageICM) wpIterator.next();

      // Construct an Item bean for the item that is CONTAINED in this
      // workpackect, not the item that IS the workpacket.
      items[i] = new CMBItem(wpItems[i].getItemPidString());
      items[i].setConnection(connection);
      i++;
    }
    session.setAttribute("wpItems", wpItems);
    session.setAttribute("itemList", items);
    RBConnectionPool.freeConnection(connection);
  }
```

RBDocRouting.java — Listing actions for advancing a work item

In order to find out what advancing options are allowed for a work package at a specific node, you must use the work process' methods *getProcessName()* and *getWorknode()* name to construct a CMBProcessICM bean, and go through the collection of CMBRouteListEntryICM to find the available advancing options.

Example 7-26 shows a code snippet for selectAdvance() method of RBDocRouting.java.

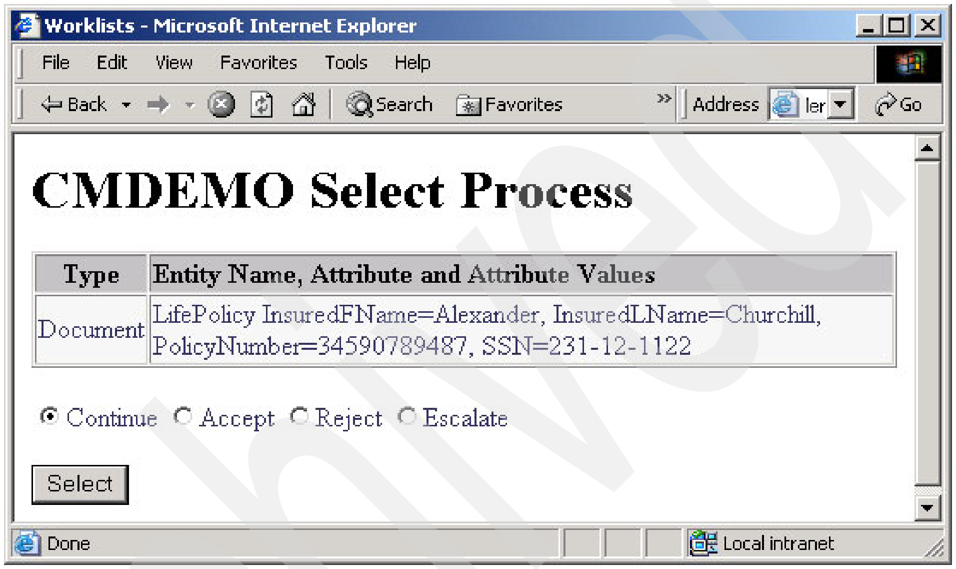
Example 7-26   RBDocRouting.java - Code snippet for selectAdvance() method

```
public static void selectAdvance(HttpServletRequest request) throws 
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    // Create a CMBItem (itemBean) using the passed in PID.  Set its 
connection, and hook it up to
    // the dataBean to retrieve the item information.
    CMBItem itemBean = new 
CMBItem(java.net.URLDecoder.decode(request.getParameter("PID")));
   CMBDataManagement dataManagement = connection.getDataManagement();

   itemBean.setConnection(connection);
   dataManagement.setDataObject(itemBean);
   dataManagement.retrieveItem();
    session.setAttribute("item", itemBean);

    CMBDocRoutingQueryServiceICM drQService = 
connection.getDocRoutingQueryServiceICM();
    // Create a workpackage using the passed in PID
    CMBWorkPackageICM wp = 

drQService.getWorkPackage(java.net.URLDecoder.decode(request.getParameter("wpPI
D")), true);

    // Create a CMBProcessICM form the name in ithe workpackage and the routes 
it has
    CMBProcessICM process = drQService.getProcess(wp.getProcessName());
    Collection route = process.getRoute();
    Iterator routeIterator = route.iterator();

    while (routeIterator.hasNext()) {
      CMBRouteListEntryICM rlEntry = (CMBRouteListEntryICM) 
routeIterator.next();
      if (rlEntry.getFrom().equals(wp.getWorkNodeName())) {
      // For the node in this process, put the names of the routes in the 
vector.
        vector.add(rlEntry.getSelection());
      }
    }
    session.setAttribute("list", vector);
    RBConnectionPool.freeConnection(connection);
  }
```

RBAdvanceEntry.jsp — Listing actions for advancing a work item

RBAdvanceEntry.jsp displays the attribute information of the item being advanced, and a list of the advancing options (such as continue, accept, reject, and escalate) from the list vector.

Figure 7-7 shows the sample screen output of the RBAdvanceEntry.jsp with advancing routing options.


Worklists - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Address Go
Done
Local intranet

CMDEMO Select Process

Type	Entity Name, Attribute and Attribute Values
Document	LifePolicy InsuredFName=Alexander, InsuredLName=Churchill, PolicyNumber=34590789487, SSN=231-12-1122

[✓ Continue] Accept Reject Escalate

Select

Figure 7-7 Sample Screen output of RBAdvanceEntry.jsp with advancing routing options

For the source code of RBAdvanceEntry.jsp, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-21 on page 579.

RBDocRouting.java — Advancing a work item

Advancing a work package involves the CMBDocRoutingManagement bean's *continueProcess()* method with the PID string of the work package, and the name of the action. Returned is the PID string of the new work package, which is a different package from the input.

Example 7-27 shows a code snippet for the advance() method of RBDocRouting.java.

Example 7-27 RBDocRouting.java - Code snippet for advance method

```
public static void advance(HttpServletRequest request) throws CMBException,
Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    CMBDocRoutingDataManagementICM drManagement =
connection.getDocRoutingDataManagementICM();

    String wpItemID =
      drManagement.continueProcess(
        (String) java.net.URLDecoder.decode(request.getParameter("wpPID")),
        (String) request.getParameter("selection"),
        connection.getUserid());
    RBConnectionPool.freeConnection(connection);
  }
```

For the complete source code of RBDocRouting.java, refer to the downloaded file from the additional material provided by this redbook or view the source in Example B-18 on page 573.

7.4 Conclusion

The sample application provided in this chapter uses Information Integrator for Content non-visual Java beans to create a generic Content Manager Web application. We demonstrated how you can log in, get a list of item types, provide search and view capabilities, and use the generic Document Routing features. In addition, the sample code also showed you how you can access OnDemand and federated servers, using search template, to query information.

We covered the major non-visual Java beans, provided some of the methods these beans have, and showed how we use them to build our application.

For a complete Java bean reference, refer to the on-line Information Center for Content Manager or Information Integrator for Content.

8

Building a case study application

In the previous chapter, we showed you how to develop a generic Content Manager Web application using the non-visual Java beans from Information Integrator for Content (II for Content).

In this chapter, based on the requirements from the case study presented in Chapter 4, “Case study” on page 61, we build a customized Web application using II for Content’s non-visual Java beans.

We cover the following topics:

- Introduction
- Application framework description
- Development environment setup
- Login use case
- Building permit application approval process use case
- External application

8.1 Introduction

We build the application based on the case study presented in Chapter 4, "Case study" on page 61. If you skipped this chapter, please go back and read the chapter before you proceed with this chapter.

In order to show the RedBrook County subsystems working together, we define a complete Web application sample framework with the objective of illustrating the implementation of the following Information Integrator for Content connectors:

- ► Content Manager connector for Content Manager Version 8
- ► Content Manager OnDemand connector for Content Manager OnDemand Version 7.
- ► Relational database connector for DB2(R) Universal Database Version 8

The business case described in Chapter 4, "Case study" on page 61 gives us the context to define our sample application. This chapter describes the framework and the WebSphere Studio Application Developer setup, and takes specific use cases to define modules to be implemented.

For each use case we implement, we walk through the development process, including the use case description, analysis, design, implementation and testing.

In this customized sample application, we only use some of the available Information Integrator for Content APIs. For a detailed look of the APIs we used, you can also refer to:

- ► Chapter 5, "Information Integrator for Content programming overview" on page 77
- ► Chapter 6, "Quick start in Web application development" on page 95
- ► Chapter 7, "Building a generic application" on page 155

For a complete description and usage of the Information Integrator for Content APIs, refer to the on-line Information Center provided with the Information Integrator for Content installation.

8.2 Application framework description

In this section, we provide a general description and diagram of the framework implemented, a technical overview of the architecture, and a short review of the Model-View-Controller (MVC) pattern.

Defining a framework helps us to plan the application development and to define how the source code and modules should be organized from technical and business conventions.

For the RedBrook County sample system, the framework objectives are as follows:

- Document an initial Web application structure based on the Model-View-Controller (MVC) pattern.

  The topics regarding best practices for Web application development are beyond the scope of this IBM Redbook. What we try to do here is to provide a framework to illustrate a starting point for any typical Web application. For further information about the MVC pattern, please refer to Chapter 1, “Web application basics” on page 3.
- Facilitate sample application implementation, developing easy-to-integrate new code for the sample application and building modules based on the implemented modules.

The customized sample application for RedBrook County consists of two Web subsystems:

- RedBrook County *Intranet System*: Used by internal users (RedBrook County personel) to process building permit related activities.
- RedBrook County *Internet System*: Used by external users (such as property owners) to query and retrieve property and building permit related information.

The framework is used in both implementations. Figure 8-1 shows the use case diagram for the entire RedBrook County sample application.

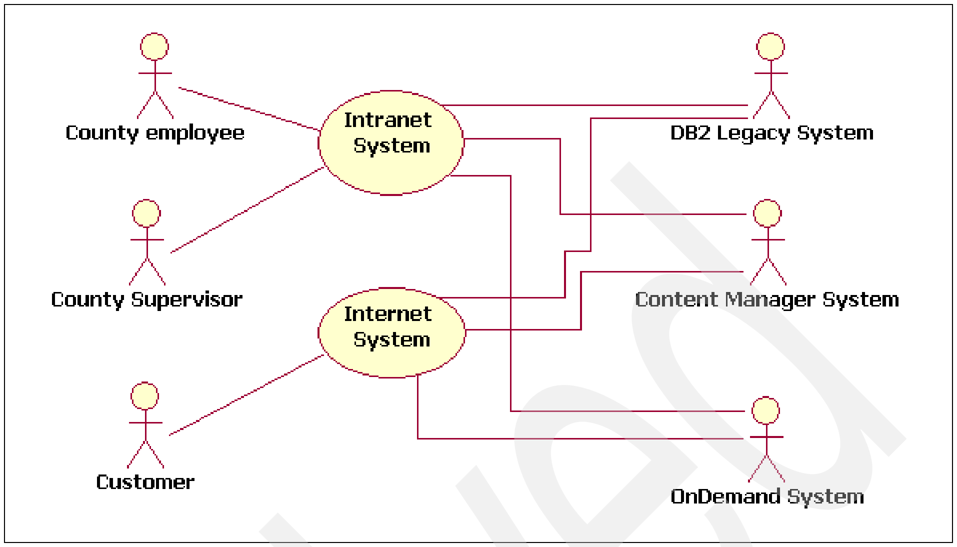


Figure 8-1 RedBrook County System use case diagram

The sample application is based on the J2EE platform. Specifically, we use Java ServerPages (JSPs), servlets, and Java beans. Figure 8-2 illustrates the technology used that maps to the MVC pattern.

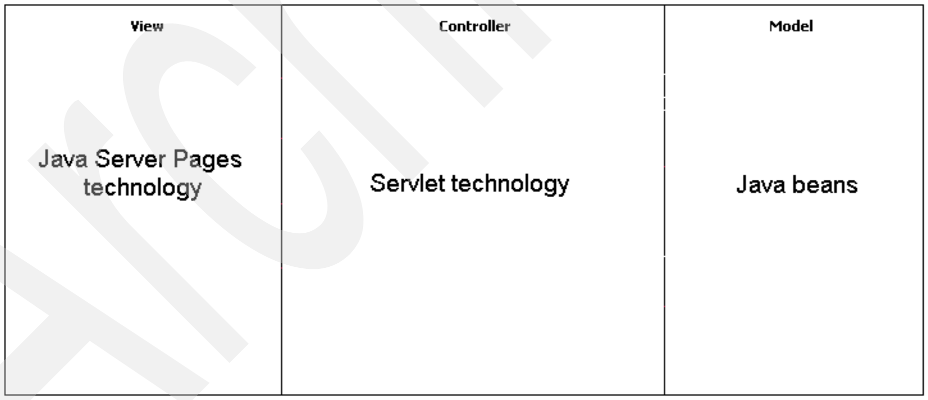


Figure 8-2 MVC and Web application technology mapping

We use the following technology to map to the MVC pattern:

JSP files	View layer. Allow user input and display result views.
Servlet files	Controller layer. In order to have a better file organization, we propose having two controller tiers. The first one is the

	general controller, the second one is for specific business control implementation classified by technology.
Java beans files	Model view. The model components wrap the APIs that manage content and implement specific business logics.

Translating the technologies into the actual implementation, Figure 8-3 illustrates the sample framework overview to be implemented within this chapter.

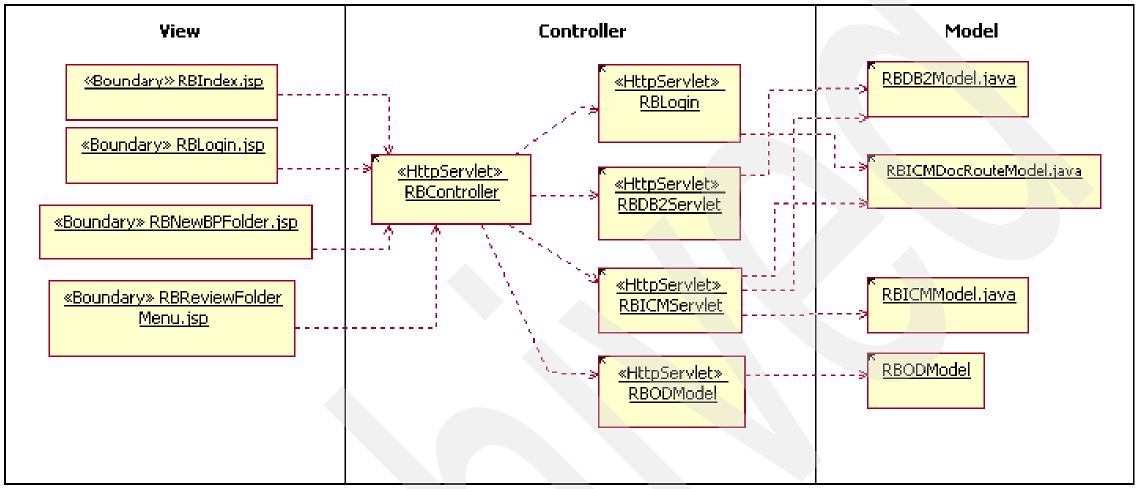


Figure 8-3 RedBrook County framework system overview

The RedBrook County application framework has three layers:

- **View layer:** There are four main JSP files. Each JSP file is explained during its specific use case. Note that not all of the JSP files are shown in Figure 8-3.
- **Controller layer:** There are five servlets used to implement the business logics and main application process flow. The servlets are implemented based on the main repository they are involved for specific tasks. Table 8-1 provides brief descriptions for these servlets.
- **Model layer:** There are Java beans making up the model layer. They implement business logics and are classified by the selected APIs to access the data mode. Table 8-2 provides brief descriptions for these Java beans.

Table 8-1 Servlets and their descriptions

Servlet	Description
RBController	Main controller. Initialize required session parameters. Redirect main action commands to specialized servlets.
RBDB2Servlet	DB2 repository tasks controller.

Servlet	Description
RBICMServlet	Content Manager repository tasks controller.
RBODServlet	Content Manager OnDemand repository tasks controller.
RBLogin	Login Servlet.

Table 8-2 Java beans and their descriptions

Java bean	Description
RBDB2Model	DB2 data access bean implemented using Information Integrator APIs.
RBICMModel	Content Manager data access and manipulation using Information Integrator OO Java APIs.
RBICMDocRouteModel	Content Manager Document Routing tasks using CM Java beans APIs.
RBODModel	Content Manager OnDemand data access using Information Integrator for Content APIs.

In addition to the servlets and Java beans mentioned, there are helper classes implemented when implementing each use case. These helper classes are important for specific tasks regarding their corresponding use cases and are explained later; the helper classes are excluded from the framework to keep the presentation simple.

For the remainder of the chapter, we develop the sample application by walking you through each step and implementing the required Java code over this proposed framework. You can either build this sample application piece by piece as you read through the chapter, or you can set up the required environment and deploy the EAR project within WebSphere Application Server by following the procedure documented in Appendix A, “Setting up case study infrastructure” on page 487. If you decide to take the second approach, you can take a quick look at how the final application works before jumping into the details of implementation.

8.3 Development environment setup

In order to have a better project organization, we recommend that you set up a specific workplace to deploy the RedBrook County System within your development tool.

If you follow the redbook chapter by chapter, you should create this workplace as described in 6.2.1, “Setting up the development directory” on page 98. See that section for the detailed steps of setting up the directory. For your convenience, we summarize the steps here as follows:

1. Create a separate directory:

   Example: `c:\wsad_sg246338`.

2. Create a new WebSphere Studio Application Developer shortcut:

   Example: `"<WSAD_HOME>\wsappdev.exe" -data c:\wsad_sg246338`

   In which <WSAD_HOME> is where you installed WebSphere Studio Application Developer.

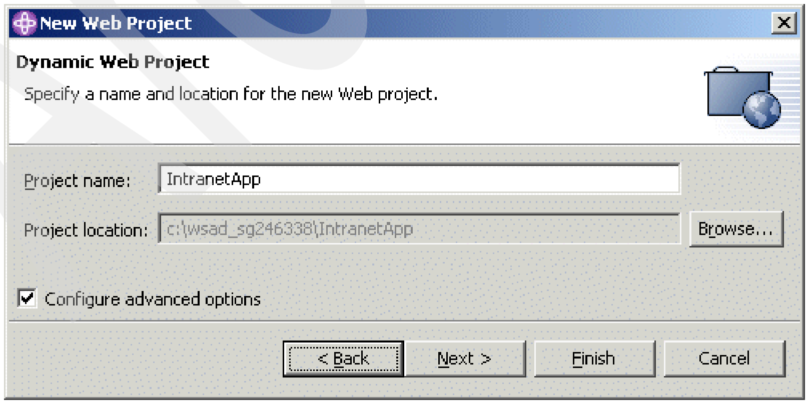
8.3.1 Create a new EAR and WAR project

There are two Web applications involve in this case study:

- RedBrook County *Intranet System*
- RedBrook County *Internet System*

To create the first Web application, go through the following steps:

1. Launch WebSphere Studio Application Server from the shortcut icon with the specific workplace (c:\wsad_sg246338).
2. Select **File** → **New** → **Project**.
3. From the New Project window, select **Web** from the left panel, and select **Dynamic Web Project** from the right panel. Click **Next**.
4. In the New Web Project window (Figure 8-4), enter `IntranetApp` for the Project name field. Select **Configure advanced options**. Click **Next**.



Specify a name and location for the new Web project.

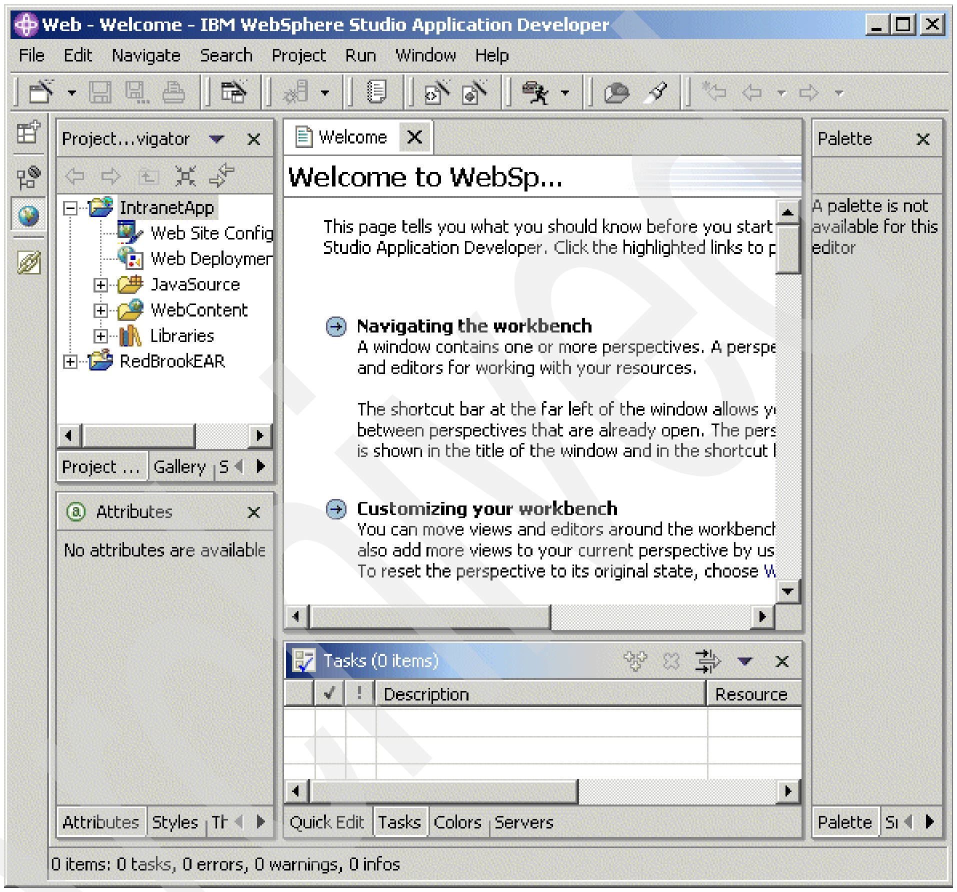
Project name: IntranetApp

Project location: c:\wsad_sg246338\IntranetApp

☑ Configure advanced options

Figure 8-4 Create a dynamic Web project

5. From the J2EE Setting Page, enter `RedBrookEAR` for the EAR project field. Use the default values for context root and J2EE level. Click **Finish**.
6. If the system prompts you to open the Web Perspective for your project, click **Yes**. You should get a screen similar to Figure 8-5.



This page tells you what you should know before you start
Studio Application Developer. Click the highlighted links to p

A window contains one or more perspectives. A perspe
and editors for working with your resources.

The shortcut bar at the far left of the window allows y
between perspectives that are already open. The pers
is shown in the title of the window and in the shortcut l

You can move views and editors around the workbencl
also add more views to your current perspective by us
To reset the perspective to its original state, choose W

0 items: 0 tasks, 0 errors, 0 warnings, 0 infos

Figure 8-5 IntranetApp Web project created

Now you are ready to start implementing your application.

8.3.2 Set up the Java build path and required libraries

Before starting to implement any code with the IBM Content Manager Information Integrator for Content APIs, configure the required libraries within the Java build path:

1. Right-click **IntranetApp**, and select **Properties** from the context menu.

2. Select **Java Build Path** from the left panel, and then select the **Libraries** tab on the right panel.
3. Click **Add External JARs**, and add the following files:
   – From the <CMBROOT> \lib directory:

     `cmb81.jar, cmbcm81.jar, cmbfed81.jar, cmbicm81.jar, cmblog4j81.jar, cmbsdk81.jar, cmbutil81.jar, cmbview81.jar, xalan, xercers, cmbod81.jar, cmbjdbc81.jar`

     <CMBROO> is where you installed Information Integrator for Content.

     By default, it is in `C:\CMBROOT`.

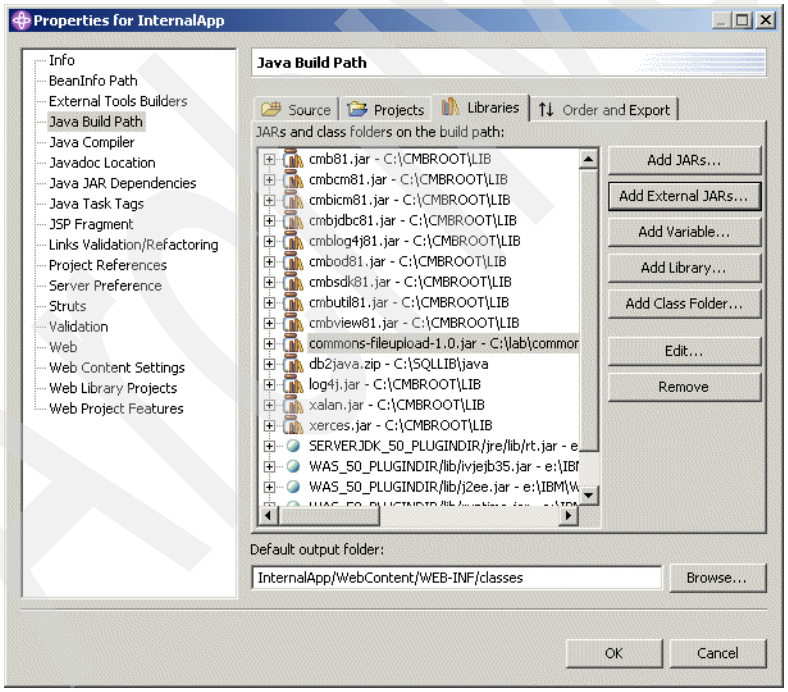
   – From the <DB2_HOME>\SQLLIB\java directory:

     `db2java.zip`

     <DB2_HOME> is where you installed DB2.

     By default, it is in `C:\Program Files\IBM`.

   Figure 8-6 shows the resulting screen.



JARs and class folders on the build path:

```
cmb81.jar - C:\CMBROOT\LIB
cmbcm81.jar - C:\CMBROOT\LIB
cmbicm81.jar - C:\CMBROOT\LIB
cmbjdbc81.jar - C:\CMBROOT\LIB
cmblog4j81.jar - C:\CMBROOT\LIB
cmbod81.jar - C:\CMBROOT\LIB
cmbsdk81.jar - C:\CMBROOT\LIB
cmbutil81.jar - C:\CMBROOT\LIB
cmbview81.jar - C:\CMBROOT\LIB
commons-fileupload-1.0.jar - C:\lab\commor
db2java.zip - C:\SQLLIB\java
log4j.jar - C:\CMBROOT\LIB
xalan.jar - C:\CMBROOT\LIB
xerces.jar - C:\CMBROOT\LIB
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - e
WAS_50_PLUGINDIR/lib/ivjejb35.jar - e:\IBI
WAS_50_PLUGINDIR/lib/j2ee.jar - e:\IBM\W
```

Default output folder: InternalApp/WebContent/WEB-INF/classes

Figure 8-6 Java Build Path setup

8.3.3 Create a server project to run the sample code

To run and test the sample code in WebSphere Studio Application Developer environment, we need to create a Server project. If you have already created a server project from the previous chapter exercise, skip this section.

To create the server project:

1. From the WebSphere Studio Application Developer, select **File** → **New** → **Other**.
2. Select **Server** from the left panel and select **Server and Server Configuration** from the right panel. Click **Next**.
3. Enter `RedBrookServer` for Server name field. For Server type field, expand **WebSphere version 5.0** and select **Test Environment**. Click **Next**.
4. Accept defaults: Use default port numbers with 9080 as HTTP port number. Click **Finish**.
5. Change to the Server perspective by selecting **Window** → **Open Perspective** → **Other**. Select **Server** and click **OK**.
6. From the lower left panel, expand Servers folder, right-click **RedBrookServer** and select **Open** from the context menu.
7. At the bottom of the upper right panel, select the **Environment** tab.
8. In the Class Path section, click **Add External JARs** and add the same files (JARs and ZIP) used within the Web projects mentioned earlier.
9. Click **Add External Folder** and add the folder where the cmbicmenv.properties file is located. By default, it is:

   ```
   C:\Program Files\IBM\CMgmt
   ```
10. Save the Server project and close the Server configuration window.
11. Right-click **RedBrookServer** and select **Add and remove projects** from the context menu. Add RedBrookD Enterprise project. Click **Finish**.

8.3.4 Create the basic framework setup

Now let us start organizing the Web project framework. This application is based on the MCV pattern. In order to start implementing the first use case, we define some conventions to achieve a good project organization.

File organization

For the Intranet application instance, create three Java packages in the Java Source folder:

itso.rb.intranet.bean	Data beans, helper beans regarding the intranet application.
itso.rb.intranet.model	Java beans wrapping repositories and data model access.
itso.rb.intranet.servlet	Controller classes. All servlet files are to be brief.

To create these packages, do the following steps:

1. Open the Web perspective. In upper left panel, Project Navigator tab, under IntranetApp, right-click **JavaSource**, and select **New** → **Package** from the context menu.
2. At the Java Package creation wizard, enter `itso.rb.intranet.bean` in the Name field. Click **Finish**.
3. Repeat the process for the remaining two packages.

Configuration property file

To keep the sample application configuration simple, we create a configuration property file in the framework within the Web Content Folder. This file is used through the project development to set up needed configuration parameters.

To create the configuration property file, do the following steps:

1. Select from menu **File** → **New** → **Other**.
2. Select **Simple** from the left panel and select **File** from the right panel. Click **Next**.
3. In the New File window, expand **IntranetApp**, and select **WebContent**. Enter `Config.properties` for the File name field. Click **Finish**.
4. The Workbench should open the file, if not, double-click the new file and write the parameters shown in Example 8-1. Customize the values for your environment.

Example 8-1 Config.properties file

```
# RedBrook County Intranet System
# config properties file

# ICM Default connection parameters
ICMServer = lsdb
ICMUser   = lsadmin
ICMPasswd = lspasswd
```

After the first framework setup, your navigator tab should look similar to Figure 8-7.

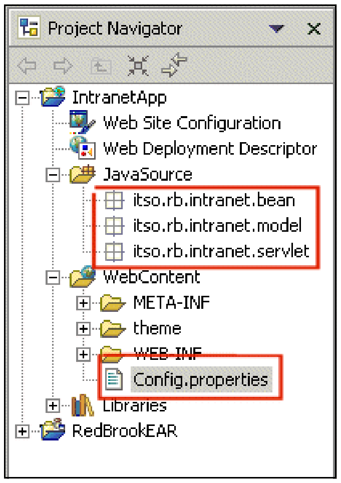


Figure 8-7 Initial framework setup

8.4 Login use case

Since this is the first module to be implemented, some helper classes shared within other modules are created here beside the corresponding model, control, and view components as described in 8.2, “Application framework description” on page 209.

8.4.1 Use case description

The following table describes the Login RedBrook Intranet system use case. In this use case, we authenticate user login. Table 8-3 describes the login use case in detail.

Table 8-3 Login use case

Element	**Description**
Name	Login - Intranet system
Description	A user logs in using ICM authentication
Actors	User, Intranet system, Content Manager system
Inputs	User name, password
Pre-conditions	Valid user name in Content Manager system

Element	Description
Steps	1. The user inputs user name and password at the login form 2. The systems displays a menu
Alternatives	1.1.1 The user or password is incorrect 1.1.2 The systems displays an error message
Post-conditions	A valid Web session is created

8.4.2 Design

Based on the framework described at the beginning of the chapter, the Login user case design is shown the class diagram in Figure 8-8.

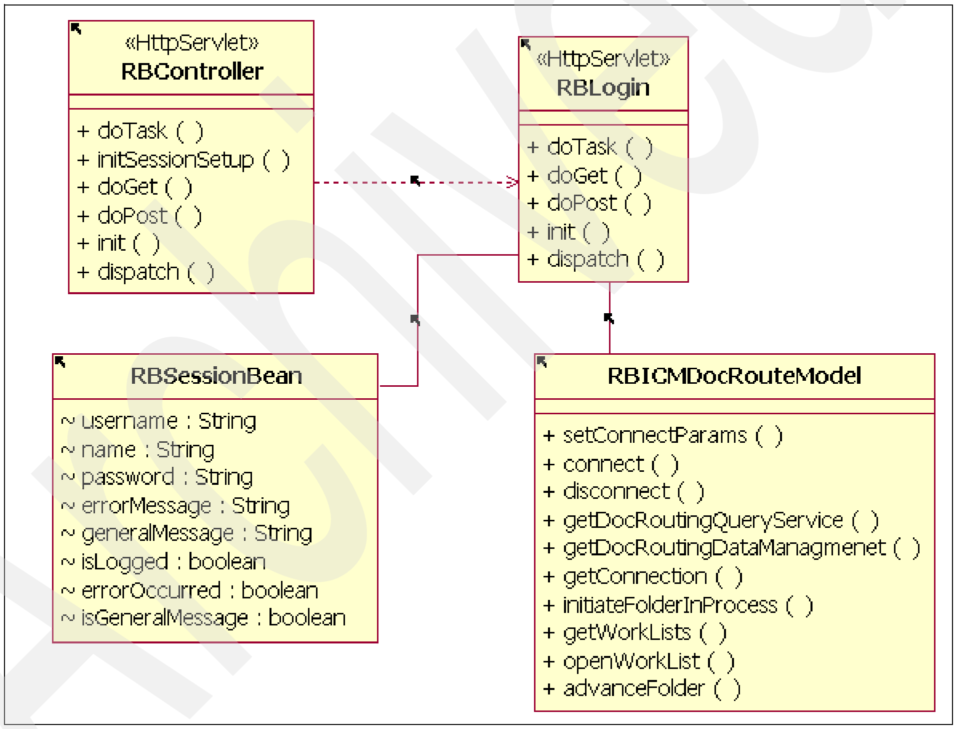


Figure 8-8 Login use case - Class diagram

Note that as we implement these classes, we do not develop all the functionality the classes should have as in the final application; rather, we only implement the functions the classes must have to fulfill the particular use case in question.

The following classes are involved in implementing the use case:

RBController	Main controller servlet. Initialize init or session parameters if it is not initialized yet. Dispatch request to a specialized servlet.
RBLogin	Specialized servlet. Handles login. Use the Content Manager model bean in order to authenticate users logged in the system. We define two Content Manager models: one is based on the Java OO APIs and one is based on the Java beans. In this case, we use Java beans for this first use case.
RBICMDocRouteModel	Content Manager model bean. Represents the interface to the content repository using the Content Manager Java beans to implement the RedBrook County system business logics.

The RBController and the RBICMDocRouteModel are shared within many use cases. For now, we develop a partial implementation of these classes.

Before developing Java code, review the sequence diagram presented in Figure 8-9 that illustrates the collaboration between every object. It is not the intention of this diagram to illustrate exactly every call or request dispatches, but to document how the objects interact with each other.

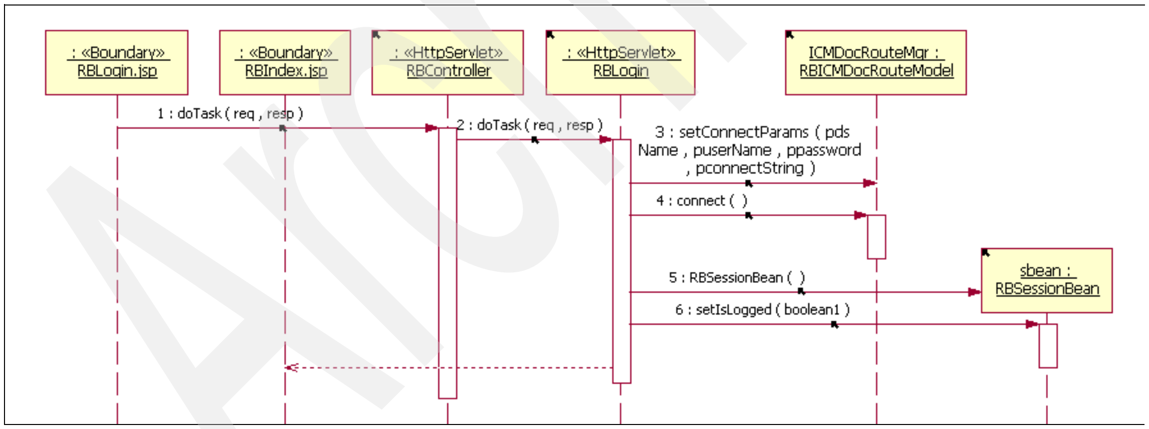

: «Boundary» RBLogin.jsp
: «Boundary» RBIndex.jsp
: «HttpServlet» RBController
: «HttpServlet» RBLogin
ICMDocRouteMgr : RBICMDocRouteModel
sbean : RBSessionBean
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : setConnectParams (pdsName , puserName , ppassword , pconnectString)
4 : connect ( )
5 : RBSessionBean ( )
6 : setIsLogged (boolean1)

Figure 8-9 Login use case - Sequence diagram

In the following sections, we implement every Java class starting with the model layer, followed by the controller layer, and last but not least, the view layer.

8.4.3 Model layer implementation

As described in the use case, we need to open a connection to the Content Manager server in order to authenticate the user.

RBICMDocRouteModel

Create a new Java Class, RBICMDocRouteModel as follows:

1. Open the Web Perspective, if it is not already open.
2. Open the IntranetApp's **Java Source** folder.
3. Right-click the Java package **itso.rb.intranet.model** and select **New** → **Class** from the context menu.
4. Verify the package value is set to `itso.rb.intranet.model`.
5. Enter `RBICMDocRouteModel` for the Name field.
6. Click **Finish**.

We plan to use this bean later to implement the Document Routing business logics using the Information Integrator for Content Java beans. For now, we only implement the code that is necessary to open a connection to the repository server.

Note: Later, we explain how to use the Information Integrator for Content Java OO APIs to open a connection using the connection pool feature.

Open the Java class code if it is not already open and paste the Java code shown in Example 8-2 in the class body.

Example 8-2 RBICMDocRouteModel.java - Partial code to connect to server

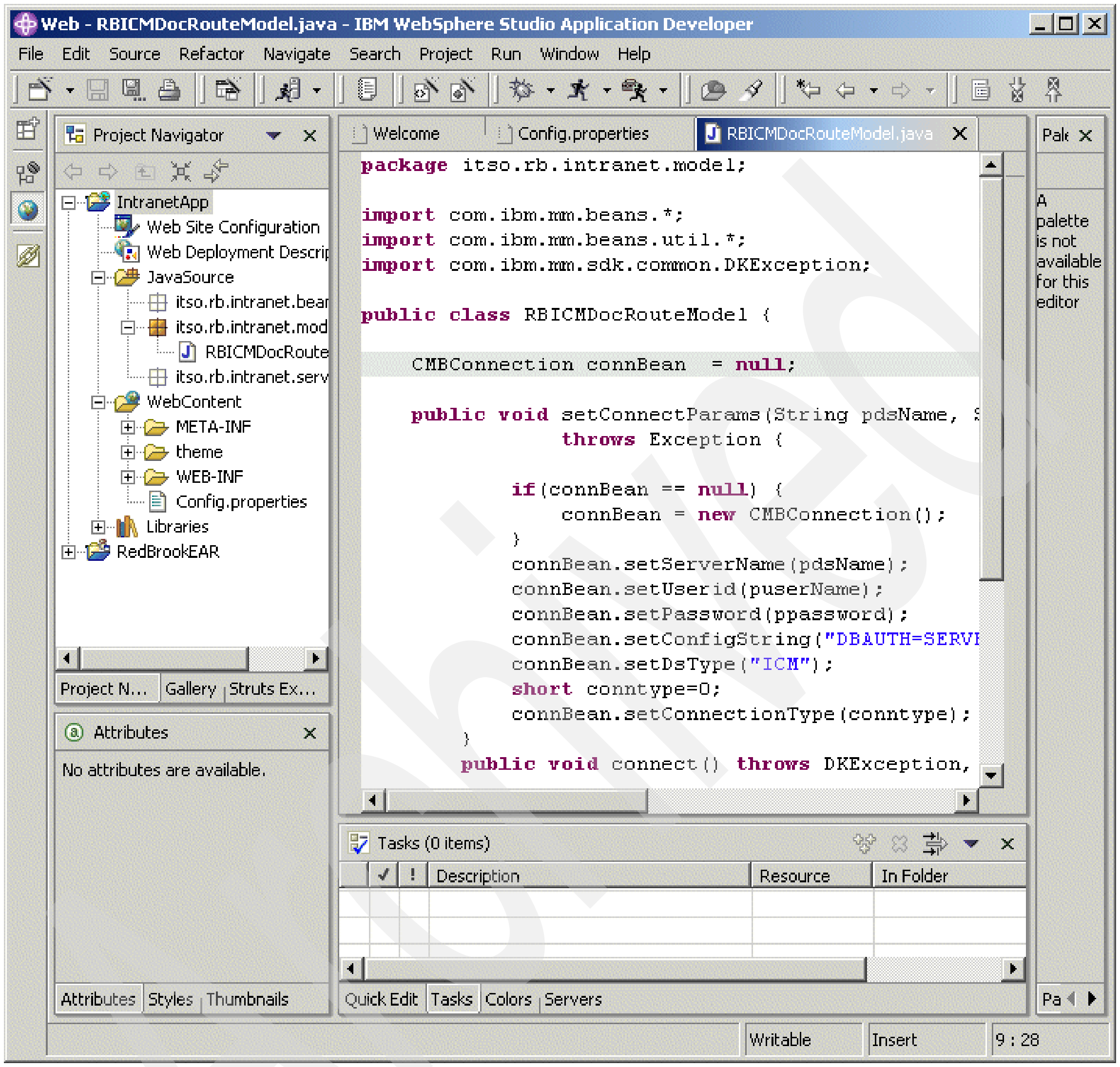
```
CMBConnection  connBean = null;

public void setConnectParams(String pdsName, String puserName,
   String ppassword, String pconnectString)
   throws Exception {

   if(connBean == null) {
      connBean = new CMBConnection();
   }
   connBean.setServerName(pdsName);
   connBean.setUserid(puserName);
   connBean.setPassword(ppassword);
   connBean.setConfigString("DBAUTH=SERVER");
   connBean.setDsType("ICM");
   short conntype=0;
   connBean.setConnectionType(conntype);
}
public void connect() throws DKException, Exception {

   if(connBean!= null && !connBean.isConnected()){
      connBean.connect();
      System.out.println("Connected to: " + connBean.getServerName() );
   }
}
public void disconnect() throws DKException, Exception {
      connBean.disconnect();
}
public CMBConnection getConnection() {
   return connBean;
}
```

Also include the following Java packages in the class file:

```
com.ibm.mm.beans.*
com.ibm.mm.beans.util.*
com.ibm.mm.sdk.common.DKException.*
```

If you have any errors listed at the tasks section, please review your Java Build Path configuration; refer to 8.3, "Development environment setup" on page 212 for details.

Figure 8-10 shows the WebSphere Studio Application Developer desktop after the creation of the model bean, RBICMDocRouteModel.


Web - RBICMDocRouteModel.java - IBM WebSphere Studio Application Developer
File Edit Source Refactor Navigate Search Project Run Window Help
Project Navigator
IntranetApp
Web Site Configuration
Web Deployment Descrip
JavaSource
itso.rb.intranet.bear
itso.rb.intranet.mod
RBICMDocRoute
itso.rb.intranet.serv
WebContent
META-INF
theme
WEB-INF
Config.properties
Libraries
RedBrookEAR
Project N... Gallery Struts Ex...
Attributes
No attributes are available.
Attributes Styles Thumbnails
Welcome Config.properties RBICMDocRouteModel.java
Pale
A palette is not available for this editor
Tasks (0 items)
Description Resource In Folder
Quick Edit Tasks Colors Servers
Writable Insert 9 : 28

```
package itso.rb.intranet.model;

import com.ibm.mm.beans.*;
import com.ibm.mm.beans.util.*;
import com.ibm.mm.sdk.common.DKException;

public class RBICMDocRouteModel {

    CMBConnection connBean  = null;

    public void setConnectParams(String pdsName, S
            throws Exception {

            if(connBean == null) {
                connBean = new CMBConnection();
            }
            connBean.setServerName(pdsName);
            connBean.setUserid(puserName);
            connBean.setPassword(ppassword);
            connBean.setConfigString("DBAUTH=SERVI
            connBean.setDsType("ICM");
            short conntype=0;
            connBean.setConnectionType(conntype);
        }
        public void connect() throws DKException,
```

Figure 8-10 WebSphere Studio Application Developer desktop after model bean definition

This bean opens a connection to the Content Manager server using the *CMBConnection* Java bean. For more information about the Java bean API, refer to the on-line Information Center of the Information Integrator for Content API References.

8.4.4 Session bean implementation

Before starting the controller layer implementation, we need to create a helper bean in order to have quick access through the servlet session to such useful information as these examples:

- User information. Example: user ID, password, user name.
- System and error messages. Example: Data bean used to implement a simple error display handling mechanism.

RBSessionBean.java

We need to create a new Java class, RBSessionBean. This data bean is used throughout the sample application. We explain how to use it in a later section.

Create the new Java class as described for the RBICMDocRouteModel Java class (see “RBICMDocRouteModel” on page 221).

Overwrite the default content by pasting or writing down the Java code as shown in Example 8-3.

Example 8-3 RBSessionBean.java - Data bean code

```
package itso.rb.intranet.bean;

public class RBSessionBean {

   String username, name, password, errorMessage, generalMessage;
   boolean isLogged;
   boolean errorOccurred;
   boolean isGeneralMessage;

   public RBSessionBean () {
      isLogged = false;
      errorOccurred = false;
   }
   public String getName() {
      return name;
   }
   public String getUsername() {
      return username;
   }
   public void setName(String string) {
      name = string;
   }
   public void setUsername(String string) {
      username = string;
   }
   public boolean getIsLogged() {
        return isLogged;
    }
    public void setIsLogged(boolean boolean1) {
        isLogged = boolean1;
    }
    public String getErrorMessage() {
        return errorMessage;
    }
    public boolean getErrorOccurred() {
        return errorOccurred;
    }
    public void setErrorMessage(String string) {
        errorMessage = string;
    }
    public void setErrorOccurred(boolean b) {
        errorOccurred = b;
    }
    public String getGeneralMessage() {
        return generalMessage;
    }
    public boolean isGeneralMessage() {
        return isGeneralMessage;
    }
    public void setGeneralMessage(String string) {
        generalMessage = string;
    }
    public void setGeneralMessage(boolean b) {
        isGeneralMessage = b;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String string) {
        password = string;
    }
}
```

Save and close the edit window.

8.4.5 Controller layer implementation

The controller layer captures user events and determines which action each event involves depending on the application's current state.

Within this framework implementation (see Figure 8-3 on page 211), we implement two servlet layers:

General servlet	RBController servlet. This main servlet initializes applications parameters and redirects the request to a specialized servlet.
Specialized servlet	Second layer of servlets. For this use case, it is the *RBLogin* servlet. This special servlet evaluates the application's state and executes the requested command. If it is necessary, it creates and calls the model data beans. After the task is done, it dispatches the request to the view layer in order to display the results.

RBController servlet

This main controller is used throughout the entire application. It always evaluates the *cmd* (command) parameter, in order to dispatch the request to the specialized servlet. We can also evaluate pre-conditions and modify the application's state if necessary before it dispatches to the specialized servlet.

To create the new servlet, use the New Servlet wizard, which updates the Web descriptor file with the servlet configuration:

1. Right-click the **itso.rb.intranet.servlet** package and select **New** → **Servlet** from the context menu.
2. If the package is not defined, set the value to `itso.rb.intranet.servlet`.
3. Enter `RBController` for Class name. Click **Finish**.

Verify and update the Web deployment descriptor file:

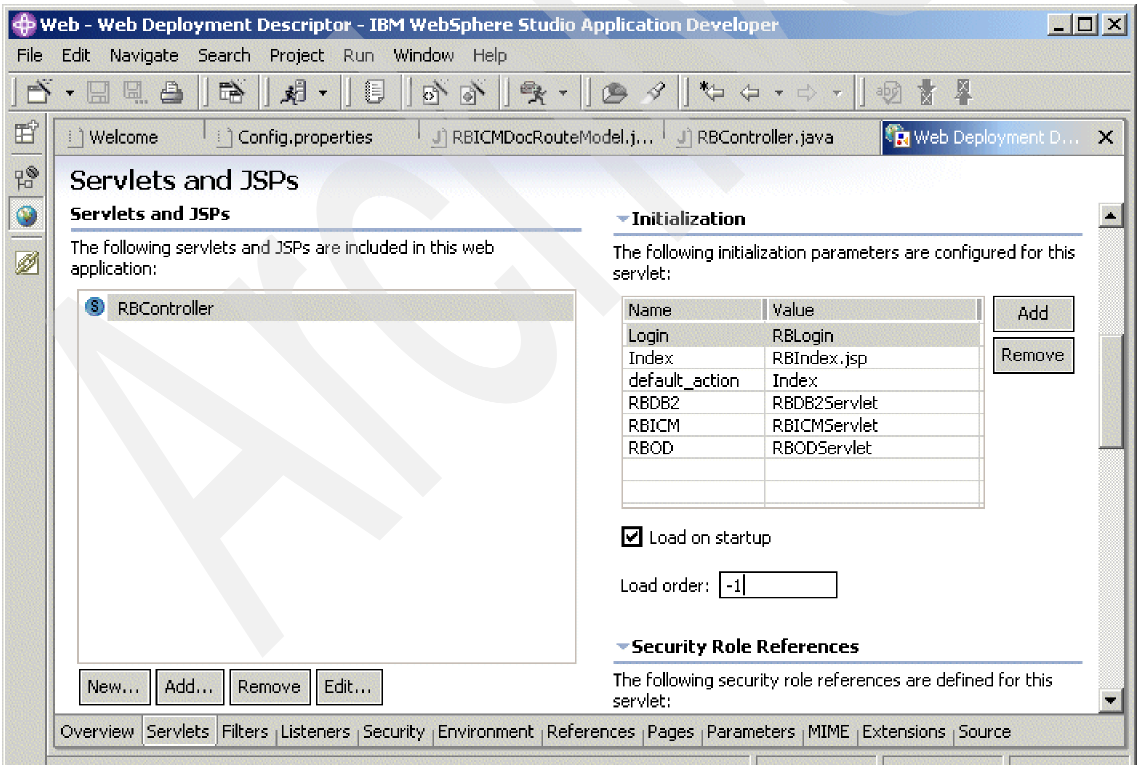
1. Within the IntranetApp Web project, select **WebContent** → **WEB-INF**.
2. Open the web.xml file by double-clicking it.
3. Select the **Servlets** tab within the Web Deployment Descriptor editor.
4. Verify that RBController is listed and click **RBController** to display the associate properties:
   - The Details section on the right side should list the following values:
     - Servlet class: `itso.rb.intranet.servlet.RBController`
     - Display name: `RBController`
   - The URL Mappings section should have the following value:
     - /RBController
5. Scroll down the screen to the Initialization section. Click **Add** and add all the initialization parameters as defined in Table 8-4.

Table 8-4 Initialization parameters for RBController servlet

Name	Value
Login	RBLogin
Index	RBIndex.jsp
default_action	Index
RBDB2	RBDB2Servlet
RBICM	RBICMServlet
RBOD	RBODServlet

These parameters are used by the servlet to dispatch the request to a specialized servlet depending on the *cmd* request parameter value.

6. Select the Load on startup option.
7. Save the Web descriptor file. Figure 8-11 shows what the Web descriptor file should look like.
8. Close the edit window.


Web - Web Deployment Descriptor - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Run Window Help
Welcome
Config.properties
RBICMDocRouteModel.j...
RBController.java
Web Deployment D...
Servlets and JSPs
Servlets and JSPs
The following servlets and JSPs are included in this web application:
RBController
New...
Add...
Remove
Edit...
Initialization
The following initialization parameters are configured for this servlet:

Name	Value
Login	RBLogin
Index	RBIndex.jsp
default_action	Index
RBDB2	RBDB2Servlet
RBICM	RBICMServlet
RBOD	RBODServlet

Add
Remove
☑ Load on startup
Load order: -1
Security Role References
The following security role references are defined for this servlet:
Overview Servlets Filters Listeners Security Environment References Pages Parameters MIME Extensions Source

Figure 8-11 Web descriptor file update

We need to implement the following RBController methods:

```
dispatch(...)
doTask(...)
initSessionSetup(...)
```

The *dispatch* method is a helper service. In this sample application development, you need to add this method to every servlet created from now on.

To implement the dispatch method, proceed as follows:

1. Open the RBController Java code.
2. Add the dispatch method as shown in after the *doPost* method. The *dispatch* method is a helper service.

Example 8-4 Generic dispatch method

```
public void dispatch(HttpServletRequest request,
                 HttpServletResponse response, String nextPage)
                          throws ServletException, IOException {
   RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
   dispatch.forward(request, response);
}
```

3. Add the following classes to the import definition:

```
javax.servlet.RequestDispatcher
java.util.ResourceBundle
```

In this sample application development, another convention is the *doTask* method. We want to centralize servlet requests, regardless if the call is a doPost or doGet method. For every servlet created from now on, we need to add or modify the doTask method, and modify both the doPost and doGet methods to call doTask.

Example 8-5 shows what these methods should look like.

Example 8-5 Generic doTask, doGet, and doPost methods

```
public void doTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   // Main task definition

}
public void doGet(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {
      // Consolidate main task
      doTask(req, resp);
}
```

```
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
         // Consolidate main task
         doTask(req, resp);
   }
```

Example 8-6 shows the implementation for the doTask method in RBController.java servlet.

Example 8-6 RBController.java - Code for doTask method

```
public void doTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   String actionServlet = null;

   // Verify Session values initialization
   if( req.getSession().getAttribute("SessionInit") == null) {
      // Initialize session parameter values
      this.initSessionSetup(req, resp);
   }

   // Define the action
   String action = req.getParameter("cmd");
   System.out.println("Command: " + action);

   if( action == null ) {
      action = getInitParameter("default_action");
   }

   // Define the action Servlet
   actionServlet = getInitParameter(action);
   System.out.println("AServlet: " + actionServlet);
   dispatch(req, resp, actionServlet);

}
```

Let us now focus on the RBController servlet's main task, the doTask method. This task consists of the following operations:

1. Verify if the *SessionInit* attribute is defined. If not, initialize some session attributes. The *initSessionSetup* method is described in Example 8-7.
2. Evaluate the *cmd* request parameter. If the value is null, the *default _action* value is used, else the request is dispatched to the defined action servlet.

At this point, our sample application does not require any further pre-condition evaluations. If needed, this is a good place to write down your code within the framework.

The *initSessionSetup* method reads the *Config.properties* file and set the values as session attributes.

Create the method in RBController.java as shown in Example 8-7.

Example 8-7 RBController.java - Code for initSessionSetup method

```
public void initSessionSetup(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   // Open Config.properties files
   ResourceBundle Config = ResourceBundle.getBundle("Config");

   // ICM default parameters
   req.getSession().
   setAttribute("CMSERVER", (String) Config.getObject("ICMServer"));
   req.getSession().
   setAttribute("CMUSER", (String) Config.getObject("ICMUser"));
   req.getSession().
   setAttribute("CMPASSWORD", (String) Config.getObject("ICMPasswd"));

   req.getSession().setAttribute("SessionInit","ok");
}
```

RBLogin servlet

Once we have almost every part of the framework, complete the controller layer by creating the specialized servlet for the Login command.

Create a new servlet as described earlier for the RBController servlet. Use the Java package *itso.rb.intranet.servlet* and set the name to `RBLogin`.

Verify that the servlet is added to the Web deployment descriptor and the URL Mapping is defined as: `/RBLogin`.

Update the RBLogin.java as follows:

1. Add the following import definitions:

   ```
   com.ibm.mm.beans.CMBException
   itso.rb.intranet.bean.RBSessionBean
   itso.rb.intranet.model.RBICMDocRouteModel
   ```

2. Add dispatch() method, modify doTask(), doGet(), and doPost() methods.

Example 8-8 shows the RBLogin.java source code. You can cut and paste the code to your RBLogin.java servlet or type it in.

Example 8-8 RBLogin.java - Source code

```
package itso.rb.intranet.servlet;

import java.io.IOException;
import javax.servlet.Servlet;
import javax.servlet.ServletException;
import javax.servlet.RequestDispatcher;

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.mm.beans.CMBException;
import itso.rb.intranet.bean.RBSessionBean;
import itso.rb.intranet.model.RBICMDocRouteModel;

/**
 * @version 1.0
 * @author
 *
 * RedBrook County System Sample
 * Login Servlet.  Authenticate user againts
 * IBM Content Manager users definition.
 *
 */
public class RBLogin extends HttpServlet implements Servlet {

   public void doTask(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

      // Get parameters
      String username = req.getParameter("username");
      String password = req.getParameter("password");
      String cmserver = (String) req.getSession().getAttribute("CMSERVER");
      String nextPage = req.getParameter("nextPage");
      RBSessionBean sbean = new RBSessionBean();

      // Log on to CM Server
      // Create model controller bean handler
      RBICMDocRouteModel ICMDocRouteMgr = null;

      try {

         // Search bean on session or create from scratch
         ICMDocRouteMgr =
         (RBICMDocRouteModel) req.getSession().getAttribute("ICMDocRouteMgr");
         if( ICMDocRouteMgr == null ) {
            ICMDocRouteMgr = new RBICMDocRouteModel();
         }

         // Open Connection
         ICMDocRouteMgr.setConnectParams(cmserver,username,password,"");
         ICMDocRouteMgr.connect();

         //  Fulfill session helper bean if no exception was caught
         sbean.setName(ICMDocRouteMgr.getConnection().getUserid());
         sbean.setPassword(password);
         sbean.setUsername( username );
         sbean.setIsLogged(true);
         req.getSession().setAttribute("ICMDocRouteMgr", ICMDocRouteMgr);

      } catch (Exception e) {
         System.out.println( e );
         sbean.setErrorOccurred(true);
         sbean.setErrorMessage("Failed to connect to the server");
      }

      req.getSession().setAttribute("SessionBean", sbean);
      dispatch(req, resp, nextPage);

   }

   public void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
         doTask(req, resp);
   }
   public void doPost(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {
         doTask(req, resp);
   }
   public void init() throws ServletException {

      super.init();

   }
   public void dispatch(HttpServletRequest request,
         HttpServletResponse response, String nextPage)
          throws ServletException, IOException {
      RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
      dispatch.forward(request, response);
   }
}
```

For the RBLogin.java servlet, the main task description is as follows:

1. Retrieves request's input parameters: user name, password, and nextPage.
2. Create a new RBSessionBean instance object.
3. Verify if the model bean RBICMDocRouteModel is created and is set as a session attribute. If not, create the bean and set the connection parameters.
4. Connect to the Content Manager server.
5. If no exception is thrown, fulfill the session data bean and place it as a session attribute.
6. Dispatch the request to the display JSP defined by the parameter *nextPage*.

Now, the model and controller layers are complete. Let us proceed with the view layer in order to complete the Login use case implementation.

8.4.6 View layer implementation

In this section, we implement the necessary mechanisms to render the model according to the way the user expects to see it. The view retrieves the data directly from the model or receives it from the controller.

Within the J2EE platform, the view layer is usually implemented using JSPs and Java beans technology.

For the Login use case instance, we defined four JSP files used through the whole sample application, as listed in Table 8-5.

Table 8-5 Login use case JSP files

JSP file	Description
RBIndex.jsp	Main JSP file. Define the application menu.
RBLogin.jsp	Login input form.
RBLogout.jsp	Logout task.
RBShowMessages.jsp	Display error or system messages defined at the RBSessionBean.

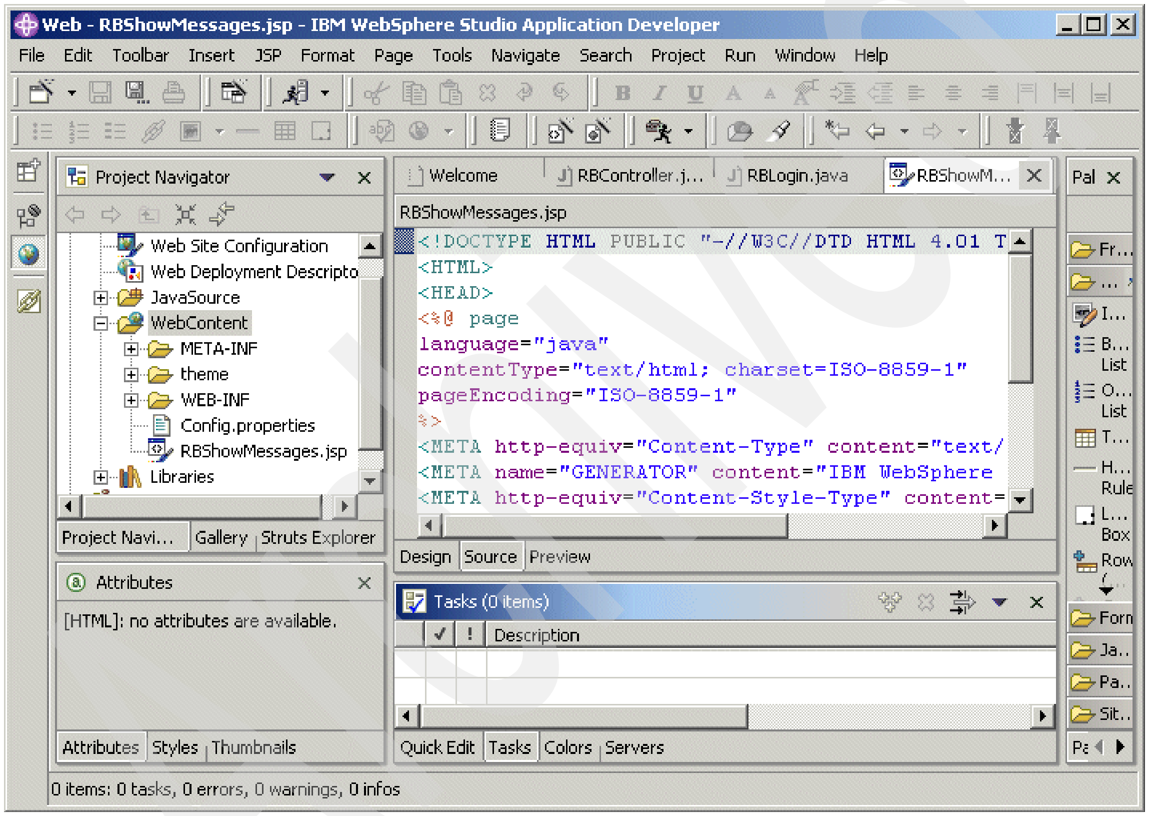
RBShowMessages.jsp

RBShowMessages.jsp is a helper JSP file used throughout the sample application. It is used as an included JSP fragment page to evaluate and display any error or system messages.

Use the following steps to create a JSP file using the wizard:

1. Under InternalApp, right-click **WebContent** and select **New** → **JSP File** from the context menu.
2. Enter `/IntranetApp/WebContent` for Folder field.
3. Enter `RBShowMessages.jsp` for File name.
4. Unselect Configure advance options. Click **Finish**.

   Review the created file (see Figure 8-12).


Web - RBShowMessages.jsp - IBM WebSphere Studio Application Developer
File Edit Toolbar Insert JSP Format Page Tools Navigate Search Project Run Window Help
Project Navigator
Web Site Configuration
Web Deployment Descripto
JavaSource
WebContent
META-INF
theme
WEB-INF
Config.properties
RBShowMessages.jsp
Libraries
Project Navi... Gallery Struts Explorer
Attributes
Attributes Styles Thumbnails
Welcome RBController.j... RBLogin.java RBShowM...
RBShowMessages.jsp
Design Source Preview
Tasks (0 items)
Description
Quick Edit Tasks Colors Servers
Pal
Fr...
I...
B... List
O... List
T...
H... Rule
L... Box
Row
Forr
Ja..
Pa..
Sit..
Pa

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 T
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/
<META name="GENERATOR" content="IBM WebSphere
<META http-equiv="Content-Style-Type" content=
```

[HTML]: no attributes are available.

0 items: 0 tasks, 0 errors, 0 warnings, 0 infos

Figure 8-12 RBShowMessage JSP File

5. Delete the predefined code and replace it with the code in Example 8-9.

Example 8-9 RBShowMessage.jsp - Source code

```
<jsp:useBean id="SessionBean"
   class="itso.rb.intranet.bean.RBSessionBean"
        scope="session"></jsp:useBean>

<% if (SessionBean.isGeneralMessage() ) { %>
<TABLE border="0" bgcolor="BLACK">
   <TBODY>
      <TR bgcolor="#c0c0c0">
         <TH>System Message</TH>
      </TR>
      <TR>
         <TD bgcolor="#FFFFFF">
         <%=SessionBean.getGeneralMessage()%>
         </TD>
      </TR>
   </TBODY>
</TABLE>
<%   SessionBean.setGeneralMessage(false);
   } %>

<% if (SessionBean.getErrorOccurred() ) { %>
<TABLE border="0" bgcolor="BLACK">
   <TBODY>
      <TR bgcolor="#ff8040">
         <TH>Error Message</TH>
      </TR>
      <TR>
         <TD bgcolor="#FFFFFF">
         <%=SessionBean.getErrorMessage()%>
         </TD>
      </TR>
   </TBODY>
</TABLE>
<%   SessionBean.setErrorOccurred(false);
   } %>
<P><BR>
</P>
```

6. Save the file and close the edit window.

RBLogin.jsp

The RBLogin.jsp file renders the Login form page.

Create the RBLogin.jsp file using the same procedures as described for creating RBShowMessages.jsp file.

Delete the predefined HTML code and replace it with the content in Example 8-10.

Example 8-10 RBLogin.jsp - Source code

```
<Form name="LoginForm" action="RBController" method="POST">
<p>Username<BR>
<INPUT type="text" name="username" size="20">
<P>Password<BR>
<INPUT type="password" name="password" size="20"></P>
<input type="hidden" name="nextPage"
value="<%=request.getParameter("nextPage") %>">
<input
   type="hidden" name="errorPage"
   value='<%=request.getParameter("nextPage") %>'>
   <input type="hidden" name="cmd" value="Login">
<P><BR>
<INPUT type="submit" name="Login" value="Login">
</P>
</FORM>
```

The JSP fragment page is included within the RBIndex.jsp file, using some evaluations to find out if the user is logged in or not. We cover RBIndex.jsp file later in this section.

Review the RBLogin servlet file, and find the required parameters defined within this JSP File: user name, password and nextPage.

The *cmd* hidden parameter is used by the *RBController* servlet to determine which servlet is used to dispatch the request.

For the sequence diagram, review Figure 8-9 on page 220.

RBLogout.jsp

In order to avoid making this sample application unnecessarily complex, the logout task is reduce to a session invalidation. This JSP file explicitly calls the disconnect from the possible existing model beans and invalidates the session object.

As an extra exercise or for your formal project, you should consider implementing a Logout servlet to explicitly free resources or process pre- and post-conditions.

For our simplified sample application, create a new JSP file, RBLogout.jsp. Replace the predefined code with the content in Example 8-11.

Example 8-11 RBLogin.jsp - Source code

```
<jsp:useBean id="SessionBean" class="itso.rb.intranet.bean.RBSessionBean"
   scope="session"></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>RBLogout.jsp</TITLE>
</HEAD>
<BODY>
<% if (SessionBean.getIsLogged()) {

    // Explicity end CM Connection
    itso.rb.intranet.model.RBICMDocRouteModel Mgr =
       (itso.rb.intranet.model.RBICMDocRouteModel)
session.getAttribute("ICMDocRouteMgr");
    if(Mgr!=null) Mgr.disconnect();

    session.invalidate();

   }
 %>
<P>[ <A href="RBIndex.jsp">Home</A> ]<BR>
<BR>
Logged out ..</P>
</BODY>
</HTML>
```

A warning task related to a broken link appear, but do not worry about this right now. It will be cleared once the RBIndex.jsp file is created.

7. Save the file and close the edit window.

RBIndex.jsp

The RBIndex.jsp is the application’s main page. Based on the SessionBean data, it displays a menu if user is logged in. If the user is not logged in, the JSP displays the login form.

Create a new JSP File named: RBIndex.jsp. Replace its content with the code in Example 8-12.

Example 8-12   RBIndex.jsp - Source code

```
<jsp:useBean id="SessionBean" class="itso.rb.intranet.bean.RBSessionBean"
   scope="session"></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML><HEAD>
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>index.jsp</TITLE>
</HEAD>
<BODY>
<H1>RedBrook County Intranet System</H1><HR>
<jsp:include page="RBShowMessages.jsp" flush="false"></jsp:include>
<% if ( !SessionBean.getIsLogged() ) {%>
      <jsp:include page="RBLogin.jsp" flush="true">
         <jsp:param name="nextPage" value="RBIndex.jsp" />
      </jsp:include>
<% } else { %>
      <DIV align="right">
       <p>Welcome again <FONT color="#808080">
      <%=SessionBean.getName()%></FONT>
      [ <A href="RBLogout.jsp">Logout</A> ] </p><BR>
      </DIV>
      <BR>
      <P>Welcome to RedBrook County, please select one of the following
      options<BR>
      </P>
      <UL>
         <LI>Option one</LI>
          <LI>Option two</LI>
      </UL>
<% } %>
<HR><P><BR>
</BODY>
</HTML>
```

8.4.7 Test

In 8.3, “Development environment setup” on page 212, the test environment server configuration is set. You are now ready to test the first piece of the sample application.

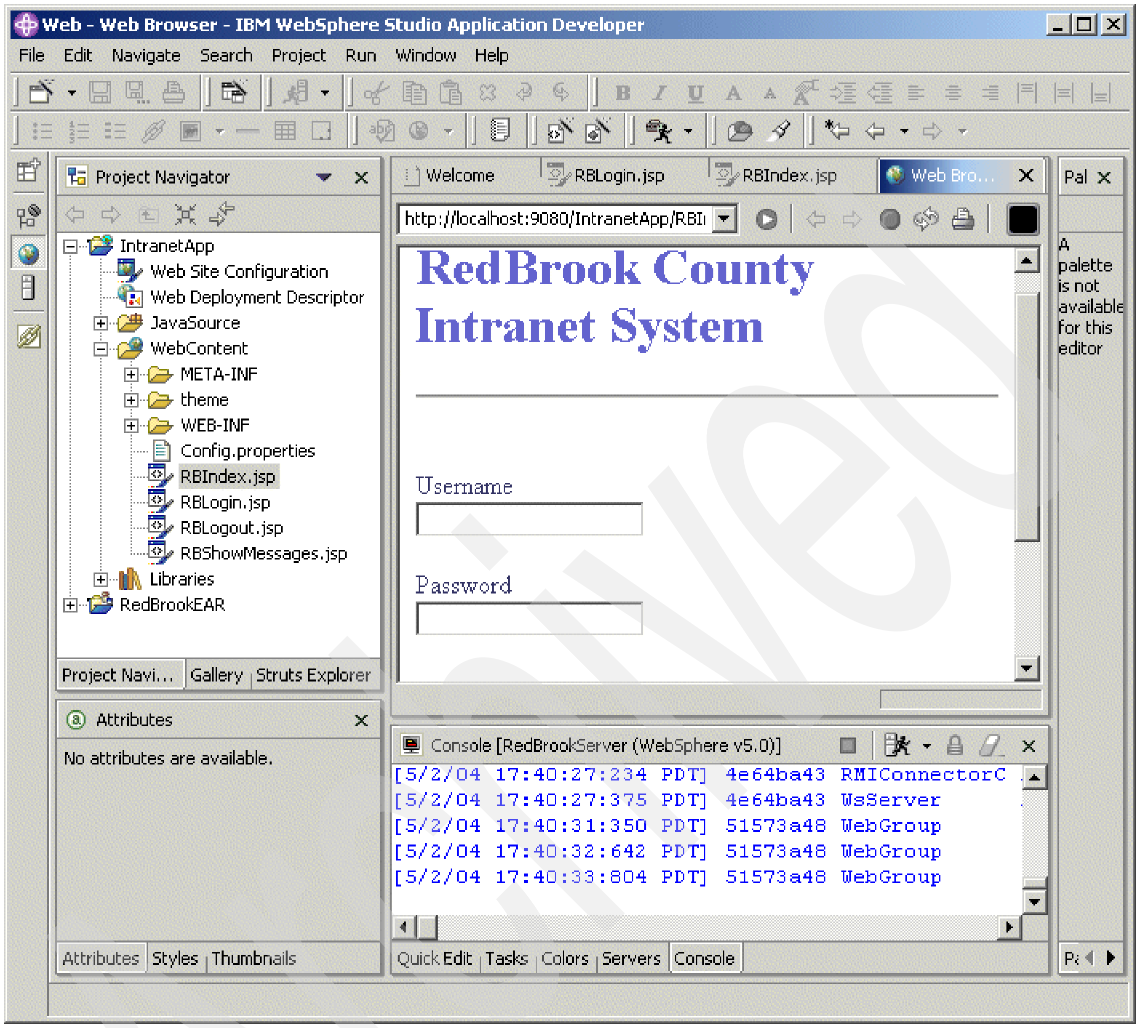
Make sure that your Content Manager Server is up an running, and do the following steps:

1. In Web perspective, under InternalApp, right-click **RBIndex.jsp** and select **Run on Server** from the context menu.
2. The Server Selection window is launched. Select the **RedBrookServer** configuration and click **Finish**.

   The application is launched with an internal browser. You can modify your WebSphere Studio Application Developer preference to launch an external browser. Alternatively, you can launch the application by copy and paste the URL to your Web browser.

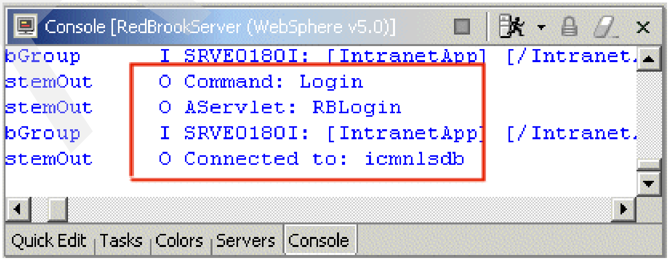
   Figure 8-13 shows the home page of the RedBrook County Intranet System.
3. Login using a valid Content Manager user.
4. If the login is successful, the RBIndex.jsp displays a user menu. Verify your console for log messages (see Figure 8-14).
5. Click **Logout**.
6. Click **Home** after you have logged out.
7. Because you logged out, the system prompts you to log in again. Enter an invalid user name and password. Click **Login**.
8. Verify that the system-error messages display is working.

Congratulations! You have just finished your first use case implementation using the RedBrook County sample system framework.


Web - Web Browser - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Run Window Help
Project Navigator
IntranetApp
Web Site Configuration
Web Deployment Descriptor
JavaSource
WebContent
META-INF
theme
WEB-INF
Config.properties
RBIndex.jsp
RBLogin.jsp
RBLogout.jsp
RBShowMessages.jsp
Libraries
RedBrookEAR
Project Navi... Gallery Struts Explorer
Attributes
No attributes are available.
Attributes Styles Thumbnails
Welcome RBLogin.jsp RBIndex.jsp Web Bro...
http://localhost:9080/IntranetApp/RBI
RedBrook County Intranet System
Username
Password
Pal
A palette is not available for this editor
Console [RedBrookServer (WebSphere v5.0)]
Quick Edit Tasks Colors Servers Console

```
[5/2/04 17:40:27:234 PDT] 4e64ba43 RMIConnectorC
[5/2/04 17:40:27:375 PDT] 4e64ba43 WsServer
[5/2/04 17:40:31:350 PDT] 51573a48 WebGroup
[5/2/04 17:40:32:642 PDT] 51573a48 WebGroup
[5/2/04 17:40:33:804 PDT] 51573a48 WebGroup
```

Figure 8-13 Login use case test - RedBrook County Intranet System


Console [RedBrookServer (WebSphere v5.0)]
Quick Edit Tasks Colors Servers Console

```
bGroup     I SRVE0180I: [IntranetApp] [/Intranet.
stemOut    O Command: Login
stemOut    O AServlet: RBLogin
bGroup     I SRVE0180I: [IntranetApp] [/Intranet.
stemOut    O Connected to: icmnlsdb
```

Figure 8-14 WebSphere Studio Application Developer console log

8.5 Building permit application approval process use case

After implementing the Login use case, you should be familiar with the RedBrook County sample system framework. In this section, we focus on the first business use case, building permit application approval. If you are not familiar with the business case, refer to Chapter 4, "Case study" on page 61.

We start by reviewing and creating a synopsis for the building permit application package approval process. The activity diagram in Figure 8-15 illustrates the simplified approval workflow we want to implement.

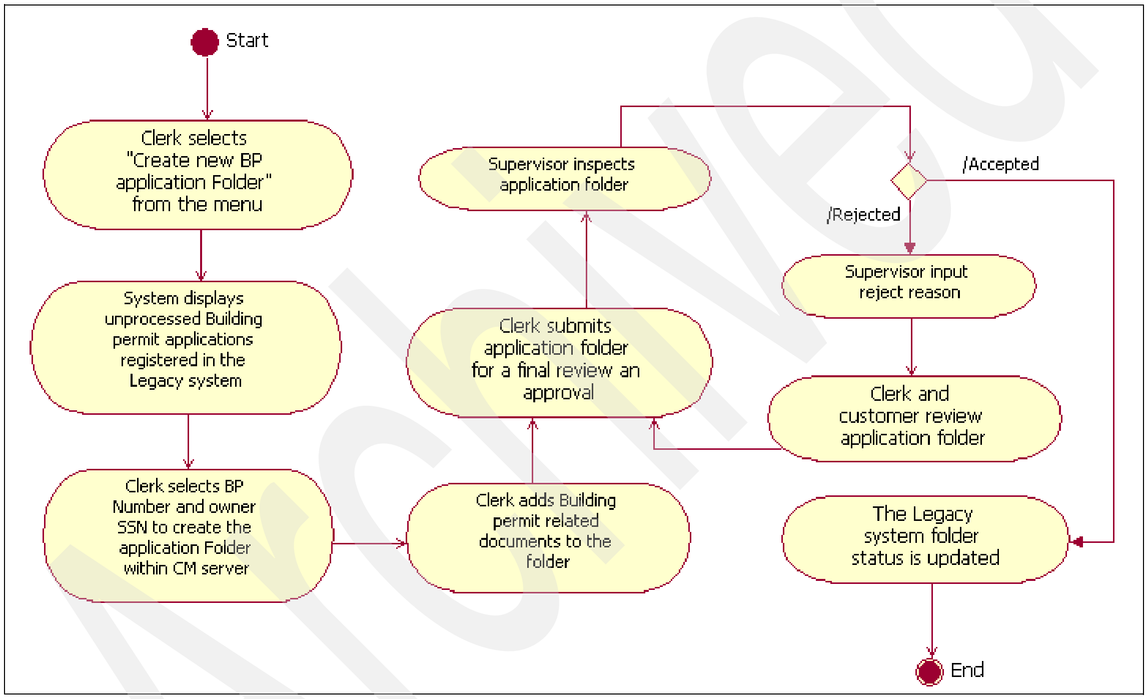

Start
Clerk selects "Create new BP application Folder" from the menu
System displays unprocessed Building permit applications registered in the Legacy system
Clerk selects BP Number and owner SSN to create the application Folder within CM server
Clerk adds Building permit related documents to the folder
Clerk submits application folder for a final review an approval
Supervisor inspects application folder
/Accepted
/Rejected
Supervisor input reject reason
Clerk and customer review application folder
The Legacy system folder status is updated
End

Figure 8-15 Building permit application approval process overview

8.5.1 General use case description

Table 8-1 describes the general use case.

Table 8-6 General use case description

Element	Description
Name	Building permit application package approval process
Description	RedBrook county approval process overview, for the building permit application package.
Actors	Clerk, supervisor, DB2 legacy system, Content Manager server
Inputs	Initial building permit (BP) number, owner's SSN, application package documents
Pre-conditions	► The Initial BP number has already been generated by the DB2 legacy system. ► The property owner record exists within the DB2 legacy system and can be retrieve using their SSN.
Steps	1. The clerk logs in to the RedBrook intranet system. 2. The clerk selects Create a new building permit application folder from the menu. 3. The clerk selects a valid initial building permit from the legacy system and the owner SSN, and creates the application folder. 4. The clerk selects the process building permit folder from the menu 5. The clerk lists the folders within the first setup in the workflow. 6. The clerk opens a specific folder and adds the necessary documents provided by the customer to the folder. 7. The folder is submitted for revision. 8. The supervisor accepts/rejects the application package. 9. The accepted applications update the DB2 legacy system and Content Manager status.
Alternatives	1. The accepted folder continues to the last step in the workflow. 2. The rejected folder is attached with a text annotation and may be re-submitted for review. 3. The clerk modifies and updates the building permit application folder, in coordination with the customer, and submits it for a second review.
Post-conditions	The DB2 legacy system record for the building permit is updated after the workflow is completed.

This use case is the most complete or complex one defined for this sample application. To simplify the implementation and explanation, we break the implementation into four different use cases:

- Create building permit application folder.
- Process building permit application folder.
- Edit building permit application folder.
- Import document to the application folder.

Prior to the implementation, we need to set up the DB2 legacy system, Content Manager data model, Document Routing, access control list and users definition.

The next few sections focus on the system setup, followed by the use case descriptions and implementation.

8.5.2 RedBrook DB2 legacy system setup

Refer to A.1, "Set up RedBrook County legacy database system" on page 488 to set up the sample database.

8.5.3 Content Manager definition setup

Refer to A.2, "Set up Content Manager system" on page 488 to set up your Content Manager server.

8.5.4 Create building permit application folder use case

In this section, we show how to create a building permit application folder use case. This use case is a piece of the building permit application approval process.

Description

Table 8-7 describes the building permit application folder use case in detail.

Table 8-7 Building permit application folder use case

Element	Description
Name	Create the building permit application folder.
Description	Create an electronic folder within Content Manager folder that contains the documents regarding the application package.
Actors	Clerk, RedBrook DB2 legacy system, Content Manager system.
Inputs	Initial building permit number, owner SSN.

Element	Description
Pre-conditions	The initial building permit record is defined within the RedBrook DB2 legacy system
Steps	1. The clerk logs in to the RedBrook intranet system. 2. The clerk selects Create a new building permit application folder from the menu. 3. The clerk selects a valid initial building permit from the legacy system and the owner SSN. 4. The clerk selects Create the application folder using the input parameters.
Alternatives	
Post-conditions	The created item folder is initiated within the approval Document Routing process.

Design

Based on the RedBrook County framework, we design the class diagram as shown in Figure 8-17 to implement the use case.

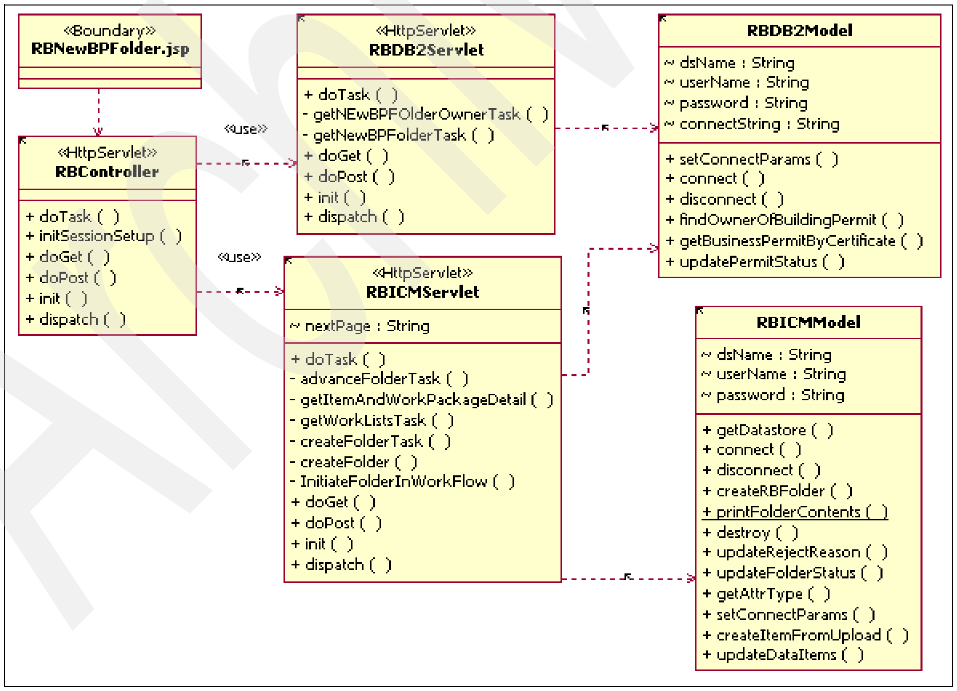
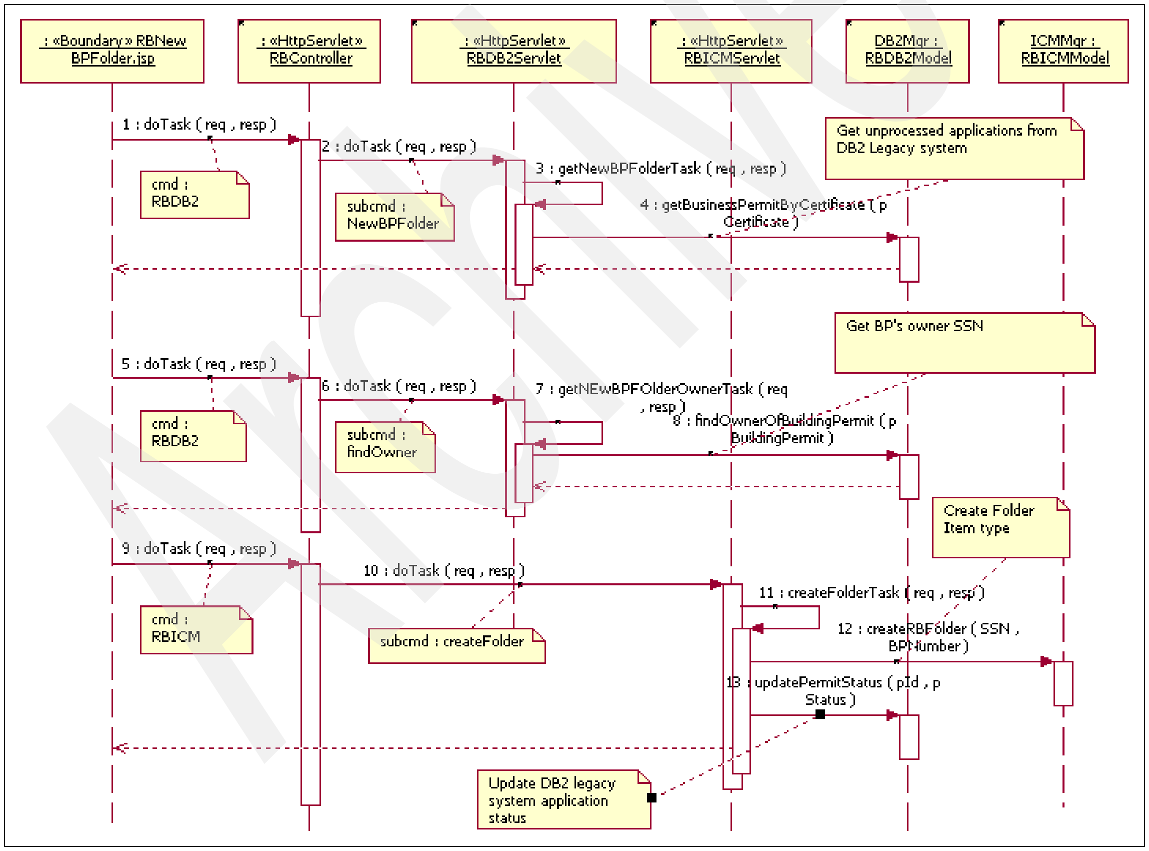


Figure 8-16 Create building permit application folder - Class diagram

Here is a description of the classes involved in implementing the use case:

RBDB2Servlet	Specialized servlet. Handles requests involving interaction with the DB2 legacy system database.
RBICMServlet	Specialized servlet. Handles requests involving interaction with the Content Manager repository.
RBDB2Model	Model bean. Implements DB2 interaction business logics.
RBICMModel	Model bean. Implements Content Manager interaction business logics using OO Java APIs, except for Document Routing tasks. See RBICMDocRouteModel bean for details.

Before developing Java code, review the sequence diagram presented in Figure 8-17 that illustrates the collaboration between every object. It is not the intention of this diagram to illustrate exactly every call or every request dispatch, but to document how the objects interact with each other.



: «Boundary» RBNew BPFolder.jsp
: «HttpServlet» RBController
: «HttpServlet» RBDB2Servlet
: «HttpServlet» RBICMServlet
DB2Mgr : RBDB2Model
ICMMgr : RBICMModel
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : getNewBPFolderTask (req , resp)
4 : getBusinessPermitByCertificate (p Certificate)
Get unprocessed applications from DB2 Legacy system
cmd : RBDB2
subcmd : NewBPFolder
Get BP's owner SSN
5 : doTask (req , resp)
6 : doTask (req , resp)
7 : getNEwBPFOlderOwnerTask (req , resp)
8 : findOwnerOfBuildingPermit (p BuildingPermit)
cmd : RBDB2
subcmd : findOwner
Create Folder Item type
9 : doTask (req , resp)
10 : doTask (req , resp)
11 : createFolderTask (req , resp)
12 : createRBFolder (SSN , BPNumber)
13 : updatePermitStatus (pId , p Status)
cmd : RBICM
subcmd : createFolder
Update DB2 legacy system application status

Figure 8-17 Create building permit application folder - Sequence diagram

In the remaining part of this section, we implement every Java class that is involved by starting with the model layer, followed by the controller layer, and then the view layer.

Model layer

The model layer for this use case is composed of two major classes:

- “RBDB2Model” on page 246
- “RBICMModel” on page 252

In addition, we implement two other classes:

- “RBBuildingPermitBean” on page 246
- “ICMConnectionPool” on page 250

RBBuildingPermitBean

RBBuildingPermitBean is a Java helper data bean. Use the following steps to create it:

1. Create a Java Class within the package *itso.rb.intranet.bean* called *RBBuildingPermitBean*.
2. Add the following attribute definitions to the Java code:

```
String Id;
String Property_nbr;
String Electrical;
Date   EInspectionDate;
String Mechanical;
Date   MInspectionDate;
String Plumbing;
Date   PInspectionDate;
String Certificate;
Date   CInspectionDate;
Double Fees;
String IssuedTo;
```

3. Add the include definition for `java.sql.Date`.
4. Right-click in the edit window of the source code and select **Source** → **Generate getter and setter** from the context menu.
5. Click **Select All** and then click **OK**.
6. Save the file and close the editor.

RBDB2Model

In order to create the building permit folder, the business requirement specifies that the record should exists in the DB2 legacy system. Our model bean allows us to interact with DB2 database using Information Integrator Java APIs to:

1. Get the list of unprocessed building permit records.
2. Get property owner's social security number (SSN) associated with the building permit number (bpnumber).
3. Update building permit record once the approval process is started and when the process is complete.

For details about the DB2 legacy system data model for our case study, refer to Appendix A.7, "Redbrook County legacy system database schema" on page 525

To implement the RBDB2Model.java:

1. Create a new Java class within the package *itso.rb.intranet.model* called *RBDB2Model*.
2. Copy and paste the source code in Example B-22 on page 581 to the class file.

To understand this class, let us examine the three main methods in this class:

- public Vector getBusinessPermitByCertificate(String pCertificate)
- public Vector findOwnerOfBuildingPermit (String pBuildingPermit)
- public void updatePermitStatus (String pId, String pStatus)

Method getBusinessPermitByCertificate(String pCertificate) does the following operations:

1. Performs a query against the DB2 table *itso.building_permit* for unprocessed records. Unprocessed records are indicated by certificate status of the records. There are three possible certificate status for a record:
   - 'N' : The building permit record has not been processed.
   - 'P' : The building permit record is on the process.
   - 'Y' : The building permit has been approved.

   The certificate status is stored in the *certificate* column in the table. This method uses the input parameter value (pCertificate) to filter the records in the table to get unprocessed certificates. The method is a generic method that also can be used to query for processed records and approved records in the table.

   Example 8-13 shows a code snippet for querying against the table.

Example 8-13 getBusinessPermitByCertificate method: Code snippet for query

```
// Check connection
if(dsDB2 == null || !dsDB2.isConnected() ) {
   this.connect();
}
// Define query
```

```
        String cmd = "SELECT * FROM itso.building_permit WHERE certificate = 
'" + pCertificate + "'";
        pQry = dsDB2.createQuery(cmd, DKConstant.DK_CM_SQL_QL_TYPE, null);
        pQry.execute(null);
```

2. Retrieved building permit results in a vector and return the vector.

   Example 8-14 shows a code snippet for retrieving the results.

Example 8-14 getBusinessPermitByCertificate method: Code snippet for get results

```
        pResults = (DKResults)pQry.result();
        dkIterator iter = pResults.createIterator();
...
        while(iter.more()) {
           item = (DKDDO) iter.next();
           if (item != null) {
              RBBuildingPermitBean lbean = new RBBuildingPermitBean();

              lbean.setId( (String) item.getDataByName("ID"));
              lbean.setProperty_nbr(
                   (String) item.getDataByName("PROPERTY_NBR"));
              lbean.setElectrical(
                   (String) item.getDataByName("ELECTRICAL"));
...
              resultTable.add(lbean);
...
      return resultTable;
```

Method findOwnerOfBuildingPermit(String pBuildingPermit) does the following operations:

1. Performs a query against the DB2 table *itso.owner*, *itso.owner_history*, and *itso.building_permit* to find the owner of a given building permit record. Owner's social security number (SSN) and building's property number (PROPERTY_NBR) are used to join the table together to get the information.

   Example 8-15 shows a code snippet for querying against the table.

Example 8-15 findOwnerOfBuildingPermit method: Code snippet for query

```
        // Define query
        String cmd = "select a.SSN, a.firstName, a.lastname " +
                " from itso.owner a, itso.owner_history b, 
itso.building_permit c " +
               " where b.SSN = a.SSN and b.PROPERTY_NBR = c.PROPERTY_NBR and 
c.id = '" + pBuildingPermit + "'";
        pQry = dsDB2.createQuery(cmd,
                  DKConstant.DK_CM_SQL_QL_TYPE,
                  null);
        pQry.execute(null);
```

2. Retrieves the results in a vector and returns the vector.

   Example 8-16 shows a code snippet for retrieving the results.

Example 8-16 findOwnerOfBuildingPermit method: Code snippet for get results

```
        DKResults pResults = (DKResults)pQry.result();
        dkIterator iter = pResults.createIterator();
...
        while(iter.more()) {
           item = (DKDDO) iter.next();
           if (item != null) {
              String SSN = (String) item.getDataByName("SSN");
              String DisplaySSN = SSN + " - " +
item.getDataByName("FIRSTNAME") + " " + item.getDataByName("LASTNAME");

              result.add(SSN);
              result.add(DisplaySSN);
...
     return result;
```

Method updatePermitStatus(String pId, String pStatus) does the following operations:

1. Performs a query against the DB2 table *itso.building_permit* to find the record that has the given building permit ID, pId.

   Example 8-17 shows a code snippet for querying against the table.

Example 8-17 updatePermitStatus method: Code snippet for query

```
        String cmd = "SELECT * FROM itso.building_permit where id = '" +
pId.trim() +"'";
        pQry = dsDB2.createQuery(cmd,
                  DKConstant.DK_CM_SQL_QL_TYPE,
                  null);
        pQry.execute(null);
```

2. Updates the status of the building permit record with the provided input parameter value, pStatus.

   Example 8-18 shows a code snippet for updating the status.

Example 8-18 updatePermitStatus method: Code snippet for updating the status

```
         DKResults pResults = (DKResults) pQry.result();
         dkIterator iter = pResults.createIterator();

         DKDDO    item = null;
         while(iter.more()) {
             item = (DKDDO) iter.next();
             // Update item
            item.setData(
               item.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"CERTIFICATE"),
               pStatus);
            // Saves changes to persistence store
            item.update();
         }
         dsDB2.commit();
...
```

For the complete source code of RBDB2Model.java, refer to Example B-22 on page 581.

ICMConnectionPool

We use the DKDatastorePool class in order to obtain the connection from a Connection Pool mechanism. This class is used for the model bean class, RBICMModel.

To implement the ICMConnectionPool.java:

1. Create a new Java class within the *default package* called *ICMConnectionPool*.
2. Copy and paste the source code in Example B-23 on page 586 to the class file.

To understand this class, let us examine the four methods in this class:

- getConnection(String ICMServer, String ICMUser, String ICMPasswd)

  Instantiates a connection pool if it does not exist. Obtains a connection from the connection pool, given a Content Manager server name, user ID, and password. See Example 8-19 for a code snippet of the method.

- returnConnection(dkDatastore connection)

  Returns a connection to the connection pool:

  ```
  connectionPool.returnConnection(connection);
  ```

- clearConnections()

  Clears all connections in the connection pool.

  ```
  connectionPool.clearConnections();
  ```

- `destroyConnections`

- Destroy the connection pool.

  ```
  if (connectionPool != null) {
      connectionPool.destroy();
      connectionPool = null;
  }
  ```

Example 8-19 getConnection method: Code snippet

```
public static synchronized DKDatastoreICM
      getConnection(String ICMServer, String ICMUser, String ICMPasswd)
    throws IllegalAccessException, InstantiationException, DKException,
Exception {

    // Create a String (fullClassName) and set it to the
   // ICM datastore class name
    String fullClassName = "com.ibm.mm.sdk.server.DKDatastoreICM";
    String server = ICMServer;
    String userid = ICMUser;
    String password = ICMPasswd;

    if (connectionPool == null) {
      System.out.println("Setting new Connection Pool");

      // Set connectionPool to a new DKDatastorePool using fullClassName
      connectionPool = new DKDatastorePool(fullClassName);
      connectionPool.setDatastoreName(server);
      connectionPool.setConnectString("");
      connectionPool.setMaxPoolSize(10);
      connectionPool.setMinPoolSize(3);

     System.out.println("userid: " + userid + " password: " + password);
      connectionPool.initConnections(userid, password, 5);
      connectionPool.setTimeOut(10);
    }
    DKDatastoreICM connection =
      (DKDatastoreICM) connectionPool.getConnection(userid, password);
    System.out.println("Connection to Datastore " + server + " returned");

    return connection;
  }
```

For the complete source code of ICMConnection.java, refer to Example B-23 on page 586.

RBICMModel

The RBICMModel bean encapsulates the main applications tasks related to the Content Manager repository. It is implemented using the Java OO APIs from Information Integrator for Content.

To implement the RBICMModel.java:

1. Create a new Java class within the *itso.rb.intranet.model* called *RBICMModel*.
2. Add the following import definitions:

   ```
   import com.ibm.mm.sdk.server.*;
   import com.ibm.mm.sdk.common.*;
   import ICMConnectionPool;
   import java.util.*;
   ```
3. Set the connection related methods and class attributes as shown in Example 8-20.

Example 8-20 RBICMModel connection method

```
DKDatastoreICM dsICM;
String dsName, userName, password;

public DKDatastoreICM getDatastore() {
   return dsICM;
}

public void setConnectParams(String pdsName, String puserName, String
ppassword){
   dsName = pdsName;
   userName = puserName;
   password = ppassword;
}
public void connect() throws DKException, Exception {
   if(dsICM == null) {
      dsICM = ICMConnectionPool.getConnection(dsName, userName, password);
   }
}

public void disconnect() {
   try{
      if(dsICM != null) {
         ICMConnectionPool.returnConnection(dsICM);
         dsICM = null;
      }
```

```
   } catch (Exception e) {
      System.out.println(e);
   }
}
```

4. Implement the business logics within this use case:

   Create RB Folder - Create a folder Item type in the Content Manager.

   Use the code shown in Example 8-21.

Example 8-21 createRBFolder

```
public DKDDO createRBFolder(String SSN, String BPNumber)
   throws DKException, Exception {

   DKDDO ddoFolder;

   if(dsICM == null) {
      connect();
   }

   // Create DKDDO Object
   ddoFolder = dsICM.createDDO("BP_FOLDER", DKConstant.DK_CM_FOLDER);
   // Set Metadata atributes
   ddoFolder.setData(
      ddoFolder.dataId(
         DKConstant.DK_CM_NAMESPACE_ATTR,"BP_NUMBER"),
         BPNumber.trim());
   ddoFolder.setData(
      ddoFolder.dataId(
         DKConstant.DK_CM_NAMESPACE_ATTR,"BP_TAX_ID"),
         SSN.trim());
   ddoFolder.setData(
      ddoFolder.dataId(
         DKConstant.DK_CM_NAMESPACE_ATTR,"BP_STATUS"),
         "In Process");

   // Save to persistent storage
   ddoFolder.add();
   this.disconnect();
   return ddoFolder;
}
```

Later on, we define more business logics as we walk through the use cases.

Controller layer

The controller layer is composed of *RBController*, *RBICMServlet*, and *RBDB2Servlet*. The RBController servlet does not need any update at this point. We need to implement:

- "RBDB2Servlet" on page 254
- "RBICMServlet" on page 257

RBDB2Servlet

The RBDB2Servlet encapsulates requests related only to the DB2 legacy system repository. The subcommands supported by this servlet, as shown in the sequence diagram, are as follows:

- NewBPFolder: Retrieves unprocessed BP records.
- findOwner: Retrieves owner SSN.

To implement RBDB2Servlet.java:

1. Create a new servlet class by following the steps described for creating "RBController servlet" on page 226. Name the new servlet as RBDB2Servlet.
2. Add the following import statements:

   ```
   import itso.rb.intranet.bean.RBSessionBean;
   import itso.rb.intranet.model.RBDB2Model;
   ```

3. Add to the servlet body the source code in Example 8-22.

Example 8-22 RBDB2Servlet code snippet

```
public void doTask(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

   String nextPage = req.getParameter("nextPage");
   String subcmd   = req.getParameter("subcmd");
   if(subcmd != null && subcmd.equals("NewBPFolder") ) {
      this.getNewBPFolderTask(req, resp);
   }
   if(subcmd != null && subcmd.equals("findOwner") ) {
      // Refresh new BP Folder list
      this.getNewBPFolderTask(req, resp);
      // Get BP Folder owner information
      this.getNEwBPFOlderOwnerTask(req,resp);
   }
   this.dispatch(req, resp, nextPage);
}

/**
 * Gets BP Folder owner's information
 * @param req
 * @param resp
```

```
 */
private void getNEwBPFOlderOwnerTask(HttpServletRequest req,
HttpServletResponse resp) {
   // Create model controller bean handler
   RBDB2Model DB2Mgr;

   // Search bean on session or create from scratch
   DB2Mgr = (RBDB2Model) req.getSession().getAttribute("DB2Mgr");
   if( DB2Mgr == null ) {
      DB2Mgr = new RBDB2Model();
      DB2Mgr.setConnectParams((String)req.getSession().getAttribute("DB2DB"),
               (String)req.getSession().getAttribute("DB2USER"),
               (String)req.getSession().getAttribute("DB2PASSWORD"),
               "");

      req.getSession().setAttribute("DB2Mgr", DB2Mgr);
   }

   // Find SSN of the related owner
   // Set Attribute:  SSN / SSN_Display
   String bpnumber = req.getParameter("bpnumber");

   // Generate error Message
   if(bpnumber == null) {
      RBSessionBean SB = (RBSessionBean)
req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("Building Permit Number MUST be provided for
'findOwner' subcommand");
   } else {
      // Dispose result to JSP
      req.getSession().setAttribute("SSNVector",
              DB2Mgr.findOwnerOfBuildingPermit(bpnumber));
   }
}

/**
 * Gets every Building permit application defined within the legacy system with
a CERTIFICATE
 * value as  "N" waiting to be processed.
 *
 * @param req
 * @param resp
 */
private void getNewBPFolderTask(HttpServletRequest req, HttpServletResponse
resp) {
   // Create model controller bean handler
   RBDB2Model DB2Mgr;
```

```
    // Search bean on session or create from scratch
    DB2Mgr = (RBDB2Model) req.getSession().getAttribute("DB2Mgr");
    if( DB2Mgr == null ) {
       DB2Mgr = new RBDB2Model();
       DB2Mgr.setConnectParams((String)req.getSession().getAttribute("DB2DB"),
                (String)req.getSession().getAttribute("DB2USER"),
                (String)req.getSession().getAttribute("DB2PASSWORD"),
                "");
       req.getSession().setAttribute("DB2Mgr", DB2Mgr);
    }

    // Search for not procesed Building Permits definition on the legacy system
    req.getSession().setAttribute("BPVector",
       DB2Mgr.getBusinessPermitByCertificate("N") );
}
```

DB2 default connection attributes handling

As seen in the Java code presented in Example 8-22, we open a connection to DB2 database using Information Integrator for Content with the *DB2DB*, *DB2User* and *DB2Passwd* session attributes. To obtain this information, we need to update the Config.properties and RBController.java files to support these new default attributes, as follows:

1. Open the Config.properties file located at **IntranetApp** → **WebContent**.
2. Append the following lines to the file:

   ```
   # DB2 Default connection settings
   DB2DB = REDBDB
   DB2User = db2admin
   DB2Passwd = thepasswd
   ```

3. Save and close the file.
4. Open the RBController.java file at **IntranetApp** → **JavaSource** → **itso.rb.intranet.servlet**.
5. Update the method *initSessionSetup* as shown in Example 8-23. See the bold text for new added code to handle DB2 parameters.

Example 8-23 Update RBController servlet

```
public void initSessionSetup(HttpServletRequest req, HttpServletResponse resp)
    throws ServletException, IOException {

    // Open Config.properties files
    ResourceBundle Config = ResourceBundle.getBundle("Config");

    // ICM default parameters
    req.getSession().setAttribute(
          "CMSERVER", (String) Config.getObject("ICMServer"));
```

```
   req.getSession().setAttribute(
         "CMUSER", (String) Config.getObject("ICMUser"));
   req.getSession().setAttribute(
         "CMPASSWORD", (String) Config.getObject("ICMPasswd"));

   // DB2 default parameter
   req.getSession().setAttribute(
         "DB2DB", (String) Config.getObject("DB2DB"));
   req.getSession().setAttribute(
         "DB2USER", (String) Config.getObject("DB2User"));
   req.getSession().setAttribute(
         "DB2PASSWORD", (String) Config.getObject("DB2Passwd"));

   req.getSession().setAttribute("SessionInit","ok");
}
```

6. Save and close the file.

RBICMServlet

The RBICMServlet encapsulates the main applications tasks related to Content Manager. It uses the many of the implemented Model beans, but mostly the RBICMModel.

The subcommand implemented by this servlet within this use case is:

```
createFolder
```

Use the RBICMModel bean to create a new folder in Content Manager.

To implement RBICMServlet.java:

1. Create a new servlet, RBICMServlet. Refer to “RBController servlet” on page 226 for detailed steps in creating servlet.
2. Add the import statements:

```
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.workflow.*;
import com.ibm.mm.sdk.common.*;
import javax.servlet.RequestDispatcher;
import itso.rb.intranet.bean.RBSessionBean;
import itso.rb.intranet.model.*;
```

3. Add the code shown Example 8-24 to implement the main task and subcommands.
4. Save and close your file.

Example 8-24 RBICMServlet business logics

```
String nextPage;

public void doTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   nextPage = req.getParameter("nextPage");
   String subcmd   = req.getParameter("subcmd");

   if(subcmd != null && subcmd.equals("createFolder") ) {
      this.createFolderTask(req, resp);
   }
   this.dispatch(req, resp, nextPage);

}

private void createFolderTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   DKDDO newFolder = null;
   String nextPage = req.getParameter("nextPage");
   // Create Folder in CM
   newFolder = createFolder(req, resp, nextPage );

   // Create display message
   RBSessionBean SB = (RBSessionBean)
req.getSession().getAttribute("SessionBean");
   SB.setGeneralMessage("Folder created successfully");
   SB.setGeneralMessage(true);
}


private DKDDO createFolder(HttpServletRequest req, HttpServletResponse resp,
String nextPage)
   throws ServletException, IOException {

   DKDDO newFolder = null;

   // Create model controller bean handler
   RBICMModel ICMMgr;

   // Search bean on session or create it from scratch
   ICMMgr = (RBICMModel) req.getSession().getAttribute("ICMMgr");
   if( ICMMgr == null ) {
      ICMMgr = new RBICMModel();
      ICMMgr.setConnectParams(
         (String)req.getSession().getAttribute("CMSERVER"),
         (String)req.getSession().getAttribute("CMUSER"),
          (String)req.getSession().getAttribute("CMPASSWORD"));

      req.getSession().setAttribute("ICMMgr", ICMMgr);
   }

   String SSN = req.getParameter("SSN");
   String BPNumber = req.getParameter("bpnumber");

   // Generate error Message
   if(SSN == null || BPNumber == null) {
      RBSessionBean SB = (RBSessionBean)
            req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("Building Permit Number and SSN MUST be provides to
create the BPFolder");

      this.dispatch(req, resp, nextPage);
   }

   try {
      // Create BP Folder using the ICM Model Bean
      newFolder = ICMMgr.createRBFolder(SSN, BPNumber);
   } catch (DKException e) {
      RBSessionBean SB = (RBSessionBean)
req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("DKException: Please verify you console for
details");
      System.out.println("ErrorCode: " + e.errorCode());
      System.out.println(e.getMessage());

   } catch (Exception e1) {
      System.out.println("Exception: " + e1 );
      RBSessionBean SB = (RBSessionBean)
req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("DKException: Please verify you console for
details");
   }
   return newFolder;
}
```

View layer

Implementing the view layer for this use case involves:

- Creating an input JSP, RBNewBPFolder.jsp, for the folder creation
- Modifying RBIndex.jsp and adding a new option to the main menu

RBNewBPFolder.jsp

This new JSP file is used to get user input for folder creation. It is called from the RBDB2Servlet after executing the *NewBPFolder* subcommand. It renders an HTML form with two elements:

- Building permit number selection (Combo Box)
- Owner's SSN Label and hidden input

As documented at the sequence diagram, after selecting the Building permit number, a request is generated with the cmd set to RBDB2 and subcommand set to *findOwner*.

The last request in the sequence is to create the folder, the submitted form, set the cmd to *RBICM*, and set the subcommand to *createFolder*.

To implement the file:

1. Create the JSP file under the Web Content folder.
2. Cut and paste the code shown in Example 8-25 for the JSP Code.

Example 8-25 RBNewBPFolder.jsp

```
<%@ page import="itso.rb.intranet.bean.RBBuildingPermitBean"%>
<%@ page import="java.util.Vector"%>

<jsp:useBean id="BPVector" class="java.util.Vector" scope="session"></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">

<SCRIPT language="JavaScript">
function createFolder_findOwner() {
   document.createFolder.subcmd.value = "findOwner";
   document.createFolder.submit();
}
function createFolder_submit() {
   document.createFolder.cmd.value = "RBICM";
   document.createFolder.nextPage.value = "RBIndex.jsp";
   document.createFolder.subcmd.value = "createFolder";
   document.createFolder.submit();
}
</SCRIPT>
<TITLE>RBNewBPFolder.jsp</TITLE>
</HEAD>
<BODY>

<H1>Create a new Building Permit Application Package Folder</H1>
<HR>

<P>[ <A href="RBIndex.jsp">Home</A> ]<BR>
</P>
<BLOCKQUOTE>
<P>Select one of the Building Permit numbers submitted in the legacy
system in order to create an electronic folder for the Application
Package<BR>
</P>
</BLOCKQUOTE>
<Form name="createFolder" method="POST" action="RBController">
<input type="hidden" name="cmd" value="RBDB2">
<input type="hidden" name="subcmd" value="NewBPFolder">
<input type="hidden" name="nextPage" value="RBNewBPFolder.jsp">

<BLOCKQUOTE>
<TABLE border="0" width="65%" bgcolor="black">
<TBODY>
   <TR bgcolor="WHITE">
      <TD><B>BP Number / Issued to:</B></TD>
      <TD><SELECT name="bpnumber" onChange="createFolder_findOwner()">
          <OPTION value="">
         <%
           RBBuildingPermitBean lbean;
           String bpnumber = (String) request.getParameter("bpnumber");
           for(int i = 0; i < BPVector.size(); i++ ) {
              lbean = (RBBuildingPermitBean) BPVector.elementAt(i);
           %>
             <% if( ( bpnumber != null) && (lbean.getId().equals(bpnumber)) ) { %>
              <OPTION value="<%=lbean.getId() %>" SELECTED>
                              <%=lbean.getId() %> - <%=lbean.getIssuedTo() %></OPTION>
             <% } else { %>
              <OPTION value="<%=lbean.getId() %>">
                              <%=lbean.getId() %> - <%=lbean.getIssuedTo() %></OPTION>
             <% } %>
          <%}%>
      </SELECT></TD>
   </TR>
   <TR bgcolor="WHITE">
      <TD><B>SSN Owner:</B></TD>
      <TD>
       <B>
        <% Vector BPOwner = (Vector) request.getSession().getAttribute("SSNVector");
           if( BPOwner != null) {
        %>
           <%=BPOwner.elementAt(1)%>
           <input type="hidden" name="SSN" value="<%=BPOwner.elementAt(0)%>">
              <% } %>
       </B>
      </TD>
   </TR>
   <TR bgcolor="WHITE">
         <TD colspan="2" align="right">
           <input type="button"  onclick="createFolder_submit()" value="Create Folder"></TD>
   </TR>
</TBODY>
</TABLE>
</BLOCKQUOTE>
</FORM>
<HR>
</BODY>
</HTML>
```

RBIndex.jsp

Update RBIndex.jsp as follows:

1. Open the file located at **IntranetApp** → **WebContent**.
2. Update the index file by adding the link as shown in Example 8-26 to the main menu. Substitute the tag `<LI>Option one</LI>` with the tag in Example 8-26.
3. Save and close the JSP file.

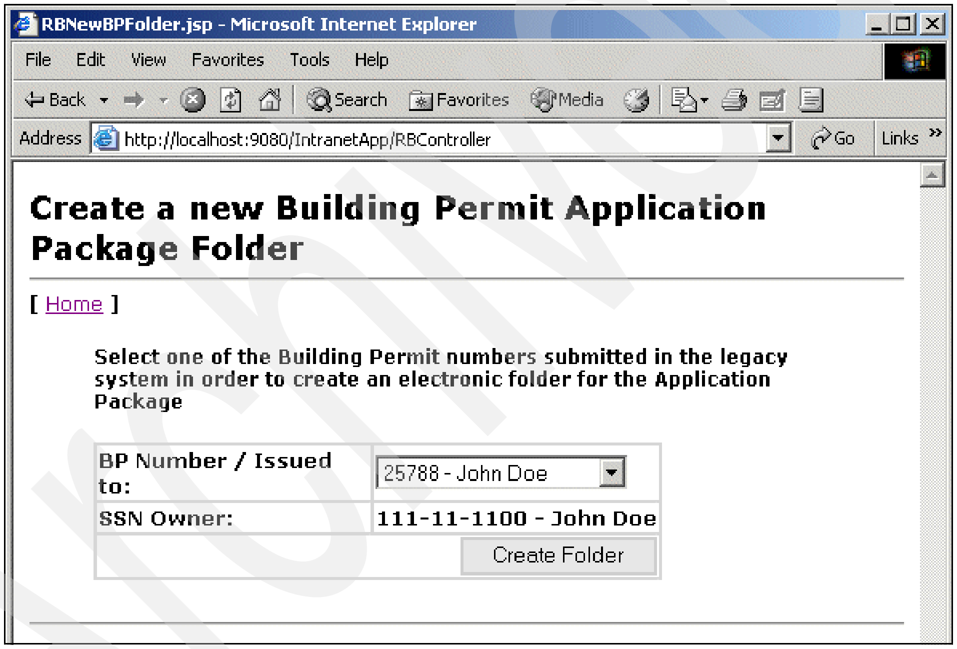
Example 8-26 Create new BP folder link

```
<LI>
 <A
href="RBController?cmd=RBDB2&subcmd=NewBPFolder&nextPage=RBNewBPFolder.jsp">
    Create a new Building permit application folder
 </A>
</LI>
```

Test

After implementing the use case, test if everything is running properly:

1. Under **InternalApp** → **Web Content**, right-click **RBIndex.jsp** and select **Run on Server** from the context menu.
2. Select **RedBrookServer** and click **Finish**.
3. Login using the `rbclerk` user.
4. Select **Create a new Building permit application folder** from the menu.
5. Select one of the unprocessed building permit numbers.
6. The Create a new Building Permit Application Package Folder screen appears (Figure 8-18). Click **Create Folder**.


RBNewBPFolder.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/IntranetApp/RBController
Go Links

Create a new Building Permit Application Package Folder

[Home]

Select one of the Building Permit numbers submitted in the legacy system in order to create an electronic folder for the Application Package

BP Number / Issued to:	25788 - John Doe
SSN Owner:	111-11-1100 - John Doe
	Create Folder

Figure 8-18 RBNewBPFolder.jsp test

7. RBIndex.jsp is called and it displays a message stating that the creation was successful.

At this time we cannot display the created folder with the sample application. You can, however, open the Content Manager Windows client and search against the BP_Folder item type. You should see newly created folder.

After your test is done, *reset* the data by deleting the created folders using the Content Manager Windows client.

8.5.5 Process building permit application folder use case

This use case is the second piece of the whole Building permit application approval process.

The following use case description highlights the goals for this implementation section.

Description

Table 8-8 describes processing the building permit application folder use case in detail.

Table 8-8 Process building permit application folder use case

Element	Description
Name	Process the building permit application folder.
Description	Process the application folder through the defined Content Manager Document Routing process.
Actors	Clerk, Supervisor, RedBrook DB2 legacy system, IBM Content Manager system.
Inputs	
Pre-conditions	The initial building permit record is defined within the RedBrook DB2 legacy system. The BP application folder has been created in the Content Manager repository and has been initiated within the approval Document Routing process.
Steps	1. The clerk logs into the RedBrook intranet system. 2. The clerk selects process application folders from the menu. 3. The clerk selects one of the available worklists and displays the existing BP Folders to be processed. 4. The clerk opens the BP Folder detail and selects a process action.
Alternatives	
Post-conditions	

Refer to "Process definition" on page 494 for further details about the defined route document process for this sample application.

The goal of this use case is to enable the Web application to process a BP application folder through the entire approval process. Since the sample application does not have any user interface to edit the folder and the related documents yet, we use the Content Manager Windows client to test the use case.

Design

To implement the new business logics, most of the framework involves updating the existing components.

Figure 8-19 illustrates the objects involved in this use case.

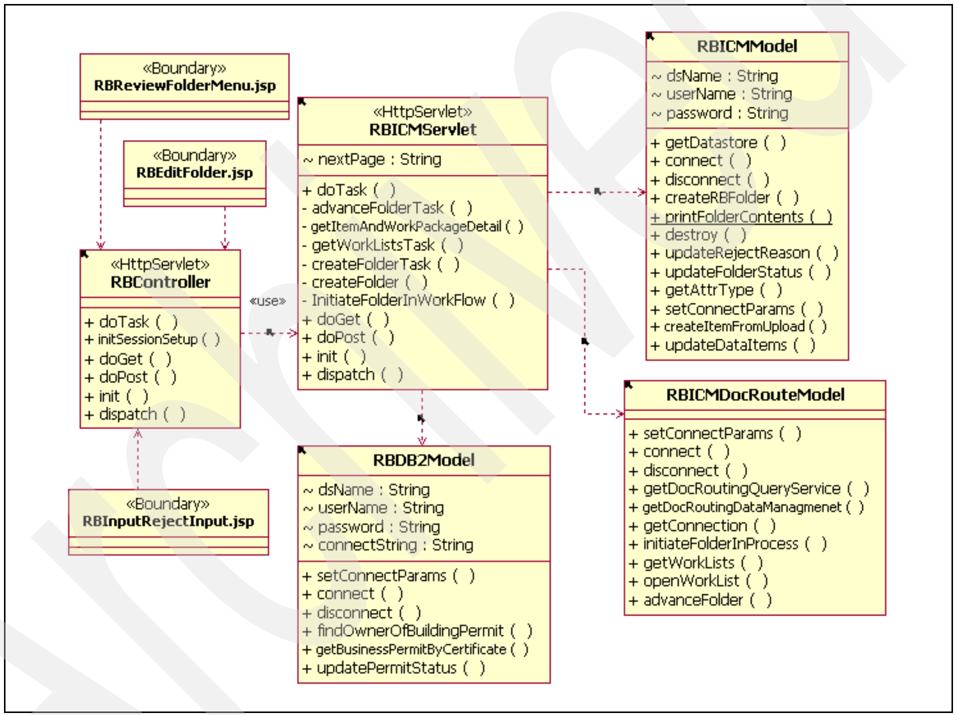


Figure 8-19 Process BP Application Folder - Class diagram

Figure 8-20 shows how the classes interact with each other. As we go through the implementation, we explain each class.

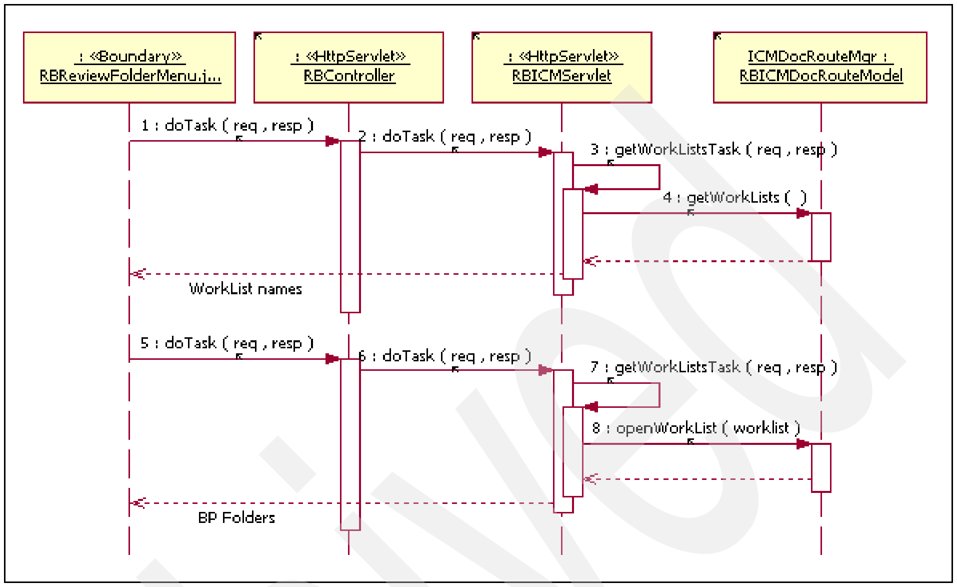

: <<Boundary>> RBReviewFolderMenu.j...
: <<HttpServlet>> RBController
: <<HttpServlet>> RBICMServlet
ICMDocRouteMgr : RBICMDocRouteModel
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : getWorkListsTask (req , resp)
4 : getWorkLists ( )
WorkList names
5 : doTask (req , resp)
6 : doTask (req , resp)
7 : getWorkListsTask (req , resp)
8 : openWorkList (worklist)
BP Folders

Figure 8-20 Query worklist's Building permit folders

Model layer

The model layer is consist of RBReviewFolderMenu.jsp. This JSP allows the user to select an available workList that contains a list of related Building permit folders to be processed.

The following tasks need to be done:

- Modify the create building permit folder within Content Manager method to initiate it in the approval process.
- Update *ICMDocRouteMgr* model bean to implement the getWorkLists() and openWorkList() methods.
- Update *RBICMServlet* servlet to handles the getWorkListsTask() method and the getWorklists subcommand.

Initiate building permit folder in Document Routing process

The BP Folder should be initiated at the Document Routing process after it is created. Do the following steps:

1. Add two attribute definitions in the RBICMDocRouteModel model bean:

```
CMBDocRoutingQueryServiceICM drqs = null;
CMBDocRoutingDataManagementICM  drdm  = null;
```

2. Add the import statements:

```
import com.ibm.mm.sdk.common.*;
import com.ibm.mm.beans.workflow.*;
```

3. Modify the setConnectParams() method to initialize the defined attributes. See Example 8-27 for the updated code.

Example 8-27 setConnectParams method: Updated code

```
public void setConnectParams(String pdsName, String puserName, String
ppassword, String pconnectString)
      throws Exception {

   if(connBean == null) {
      connBean = new CMBConnection();
   }
   connBean.setServerName(pdsName);
   connBean.setUserid(puserName);
   connBean.setPassword(ppassword);
   connBean.setConfigString("DBAUTH=SERVER");
   connBean.setDsType("ICM");
   short conntype=0;
   connBean.setConnectionType(conntype);

   // get the document routing query service bean
   drqs = connBean.getDocRoutingQueryServiceICM();
   // set the trace on for the document routing query service bean
   drqs.setTraceEnabled(true);
   // get the document routing data management bean
   drdm = connBean.getDocRoutingDataManagementICM();
   // set the trace on for the document routing data management bean
   drdm.setTraceEnabled(true);
   // set the trace on for the connection bean
   connBean.setTraceEnabled(true);
}
```

4. Add initiateFolderInProcess method as shown in Example 8-28.

Example 8-28 initiateFolderInProcess method

```
public String initiateFolderInProcess(String pprocessName, DKDDO pfolder)
      throws Exception {

   this.connect();

   String wpItemID =
           drdm.startProcess(
            pprocessName,
            pfolder.getPidObject().pidString(),
            1,
            connBean.getUserid());

   return wpItemID;
}
```

The `CMBDocRoutingDataManagementICM.startPRocess()` method starts a document in a Document Routing process. The input parameters are:

- String pprocessName: The name of the process.
- String pid: The PID of the item in the work package.
- int priority: The priority of the work package.
- String owner: The owner. This parameter is optional.

5. Update the controller servlet definition in the *RBICMServlet* code as shown in Example 8-29.

Example 8-29 createFolderTask method update

```
private void createFolderTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

      DKDDO newFolder = null;
      String nextPage = req.getParameter("nextPage");
      // Create Folder in CM
      newFolder = createFolder(req, resp, nextPage );
      // Initiate Folder in Route Document Proccess
      String wppid = null;
      if (newFolder != null ) {
         wppid = initiateFolderInWorkFlow(req, resp, nextPage, newFolder);
      }
      // Update legacy system building permit Status
      if(wppid != null) {
         // Retrieve legacy system model
         RBDB2Model BPMgr = (RBDB2Model)
            req.getSession().getAttribute("DB2Mgr");
         if( BPMgr == null ) {
            BPMgr = new RBDB2Model();
            BPMgr.setConnectParams("REDBDB","Administrator","demo4you","");
            req.getSession().setAttribute("DB2Mgr", BPMgr);
         }
         // update building permit record to "P" (Pending approve)
         BPMgr.updatePermitStatus(req.getParameter("bpnumber"),"P");

         // Create display message
         RBSessionBean SB = (RBSessionBean)
            req.getSession().getAttribute("SessionBean");
         SB.setGeneralMessage("Folder created successfully");
         SB.setGeneralMessage(true);
      }
}
```

6. Update the DB2 legacy system. Once the folder is initiated at the workflow, the Building permit status in the legacy system should updated to “P” (Pending approve). A new method, InitateFolderInWorkFlow(), needs to be created to update the status. See Example 8-30 for the source code.

Example 8-30 initiateFolderInWorkFlow method

```
private String initiateFolderInWorkFlow(HttpServletRequest req,
HttpServletResponse resp, String nextPage, DKDDO newFolder)
   throws ServletException, IOException {
   String WorkPackagePID = null;
   // Create model controller bean handler
   RBICMDocRouteModel ICMDocRouteMgr;

   try {

   // Search bean on session or create from scratch
      ICMDocRouteMgr = (RBICMDocRouteModel)
req.getSession().getAttribute("ICMDocRouteMgr");
      if( ICMDocRouteMgr == null ) {
         ICMDocRouteMgr = new RBICMDocRouteModel();

         ICMDocRouteMgr.setConnectParams(
            (String)req.getSession().getAttribute("CMSERVER"),
            (String)req.getSession().getAttribute("CMUSER"),
            (String)req.getSession().getAttribute("CMPASSWORD"),
            "");

         ICMDocRouteMgr.connect();
         req.getSession().setAttribute("ICMDocRouteMgr", ICMDocRouteMgr);
      }

      // Initiate Folder in Route doc Process
      WorkPackagePID =
ICMDocRouteMgr.initiateFolderInProcess("RBBPApprove",newFolder);

   } catch (Exception e1) {
      System.out.println( e1 );
      RBSessionBean SB = (RBSessionBean)
         req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("Exception: And error occured during the workflow
definition. Check console for details");
   }
   return WorkPackagePID;
}
```

7. To test the new modifications:
   a. Run the *RBIndex.jsp* on the server and create a new folder.
   b. Use the IBM Content Manager Client for Windows to list the worklists and verify that the folder is on the *BPCollection* worklist.
   c. Connect to the REDBDB database and verify that the row is updated:

      ```
      select id, property_nbr, certificate from itso.building_permit
      ```

   If you had any errors during this test, review the WebSphere Studio Application Developer console and verify your code.

   If you want to reset the data, make sure that you:

   a. Delete the created folders from the Content Manager system.
   b. Update the itso.building_permit table by set the certificate value to “N”:

      ```
      update itso.building_permit set certificate = ‘N’
      ```

Querying the Worklist packages

Now that our building permit folder has been initiated at the Document Routing process, let us prepare the model to query this structure as shown in the sequence diagram Figure 8-20 on page 266.

Open the *RBICMDocRoutemodel.java* file again, to implement these methods:

- public Vector getWorkLists(): Get a list of available worklists for a user. See Example 8-31 on page 271 for the code snippet.
- public Vector openWorkList(String worklistname): Get a list of documents or folders (work packages) related to a selected worklist. See Example 8-32 on page 271 for the code snippet.
- public void advanceFolder(String wpPID, String selection): Advance a work package by ID to the selected step. See Example 8-33 on page 272 for the code snippet.

Make sure you add the following import statements:

```
import java.util.Vector;
import java.util.Collection;
import java.util.Iterator;
```

Note that the getWorkList method uses the *CMBDocRoutingQueryServiceICM* bean to retrieve the available worklists for a given user. Review the openWorkList method to find out how to retrieve the documents related to a worklist using the same bean API. In order to make the code a bit more simple, we return the *CMBWorkPackageICM* and *CMBItem* beans related to these work packages contained within the worklist. These beans are used directly by the View layer for rendering the results display.

Example 8-31 getWorkLists method

```
public Vector getWorkLists() {
   Vector WLVector = new Vector();
   try {

      // List the worklists names for display in the jsp.  The names need
      // to be passed to the next servlet for opening
      String worklists[] = drqs.getWorkListNames();

      for (int i = 0; i < worklists.length; i++) {
        WLVector.add(worklists[i]);
      }
   } catch (CMBException e ) {
      System.out.println(e);
   }
   return WLVector;
}
```

Example 8-32 openWorkList method

```
/**
 * Retrieves items and workpackages related to a worklist
 * Return: java.util.Vector
 *   elementAt(0) CMItem[x]
 *   elementAt(1) CMWorkPackageICM[x]
 *
 * @param worklist
 * @param req
 * @return
 * @throws Exception
 */
public Vector openWorkList(String worklist)
   throws Exception {

   Vector data = new Vector(2);
   this.connect();
   // Get the work packages that are in the worklist for display
   Collection workpackages = drqs.getWorkPackages(worklist, "");
   Iterator wpIterator = workpackages.iterator();

   int i = 0;
   CMBDataManagement dataManagement = getConnection().getDataManagement();
   CMBWorkPackageICM[] wpItems = new CMBWorkPackageICM[workpackages.size()];
   CMBItem[] items = new CMBItem[workpackages.size()];

   while (wpIterator.hasNext()) {
     wpItems[i] = (CMBWorkPackageICM) wpIterator.next();
     // Construct an Item bean for the item that is CONTAINED in this
```

```
      // workpackect, not the item that IS the workpacket.
      items[i] = new CMBItem(wpItems[i].getItemPidString());
      items[i].setConnection(getConnection());
      // Retrieve the item's information so a descriptive attribute string
      // can be created in the JSP
      dataManagement.setDataObject(items[i]);
      dataManagement.retrieveItem();
      i++;
   }

   data.add(0, items);
   data.add(1, wpItems);

   return data;
}
```

Example 8-33 advanceFolder method

```
public void advanceFolder(String wpPID, String selection)
      throws CMBException {

   drdm.continueProcess(
      wpPID,
      selection,
      getConnection().getUserid() );
}
```

Controller layer

Next, let us work on the controller layer.

Update RBICMServlet servlet

We need to update RBICMServlet.java file to add the getWorkListsTask() method and the getWorklists subcommand:

1. Use the code in Example 8-34 to implement the method in the RBICMServlet.java file.

Example 8-34 getWorkListsTask method

```
private void getWorkListsTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   RBSessionBean lbean = (RBSessionBean)
req.getSession().getAttribute("SessionBean");

   // Get the available Worklists
   // Create model controller bean handler
   RBICMDocRouteModel ICMDocRouteMgr;
   try {

      // Search bean on session or create from scratch
      ICMDocRouteMgr = (RBICMDocRouteModel)
req.getSession().getAttribute("ICMDocRouteMgr");
      if( ICMDocRouteMgr == null ) {
         ICMDocRouteMgr = new RBICMDocRouteModel();

         ICMDocRouteMgr.setConnectParams(
            (String)req.getSession().getAttribute("CMSERVER"),
            (String)req.getSession().getAttribute(lbean.getUsername()),
            (String)req.getSession().getAttribute(lbean.getPassword()),
            "");
         req.getSession().setAttribute("ICMDocRouteMgr", ICMDocRouteMgr);
      }

      // Initiate Folder in Route doc Process
      Vector WLVector = ICMDocRouteMgr.getWorkLists();

      req.getSession().setAttribute("WLVector", WLVector);

      // Load list detail if requested
      if(req.getParameter("WorkList") != null ) {
         // Get ICMWorkPackageICM and Items
         Vector resultTable =
         ICMDocRouteMgr.openWorkList((String) req.getParameter("WorkList"));

         // Set vector available for JSP's display
         req.getSession().setAttribute("WorkListItems", resultTable);
      }

   } catch (Exception e1) {
      System.out.println( e1 );

      RBSessionBean SB = (RBSessionBean)
         req.getSession().getAttribute("SessionBean");
      SB.setErrorOccurred(true);
      SB.setErrorMessage("Exception: And error occured retriving CM
Worklists");
   }
}
```

2. Update the *doTask*() method to call the new getWorkListsTasks. See Example 8-35 for the updated code.

Example 8-35 doTask method: Updated code

```
public void doTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   nextPage = req.getParameter("nextPage");
   String subcmd   = req.getParameter("subcmd");

   if(subcmd != null && subcmd.equals("createFolder") ) {
      this.createFolderTask(req, resp);
   }
   if(subcmd != null && subcmd.equals("getWorkLists") ) {
      this.getWorkListsTask(req, resp);
   }
   this.dispatch(req, resp, nextPage);
}
```

View layer

As shown in the sequence diagram in Figure 8-20 on page 266, these are some of the designed boundary interfaces:

- RBReviewFolderMenu.jsp: Lists the available worklists for a user and retrieves the documents within a selected worklist.
- RBEditFolder.jsp: Displays the detailed building permit folder page. Displays the allowed actions within the Document Routing process, and during the next use case definition. Implements the file upload capability.

Create the JSP file under WebContent and copy and paste the source code in Example 8-36 into the code.

Example 8-36 RBReviewFolderMenu

```
<jsp:useBean id="WLVector" class="java.util.Vector" scope="session"></jsp:useBean>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<%@ page import="com.ibm.mm.beans.workflow.*"%>

<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
```

```
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>RBReviewFolderMenu.jsp</TITLE>
<SCRIPT language="JavaScript">

function verifyWLSelection() {
  if(document.StatusFilterForm.WorkList.value == "") {
    alert("Please select a valid WorkList value.");
  }else {
    document.StatusFilterForm.submit()
  }
}

</SCRIPT>
</HEAD>
<BODY>
<H1>Process Permit Application Folders</H1>
<HR>
<P>[ <A href="RBIndex.jsp">Home</A> ]<BR>
</P>

<jsp:include page="RBShowMessages.jsp" flush="false"></jsp:include>

<!--- Worklist combo box -->
<Form name="StatusFilterForm" method="POST" action="RBController">
<input type="hidden" name="cmd" value="RBICM">
<input type="hidden" name="subcmd" value="getWorkLists">
<input type="hidden" name="nextPage" value="RBReviewFolderMenu.jsp">
<TABLE border="0" width="80%">
   <TBODY>
      <TR>
         <TD><B>Review and process Permit Applications Folders by Status:</B></TD>
         <TD>
         <% if( (WLVector == null) || (WLVector.size() == 0) ) { %>
            <B> No worklist were found </B>
         <% } else { %>
         <SELECT name="WorkList"
            onChange="verifyWLSelection()">
            <OPTION value="">
            <%
               String WorkList = request.getParameter("WorkList");
               if(WorkList == null ) WorkList = "";
               for(int i=0; i < WLVector.size() ; i++) { %>
               <% if( WorkList.equals((String)WLVector.elementAt(i))) { %>
                  <option value="<%=(String)WLVector.elementAt(i)%>" SELECTED>
                   <%=(String)WLVector.elementAt(i)%>
               <% } else { %>
                  <option
value="<%=(String)WLVector.elementAt(i)%>"><%=(String)WLVector.elementAt(i)%>
               <% } %>
            <% } %>
         </SELECT>
         <% } %>

         </TD>
      </TR>
   </TBODY>
</TABLE>
</FORM>
<!--- / Worklist combo box -->
<hr>
<% if( request.getParameter("WorkList") != null ) { %>

<h1> Folder List</h1>
<TABLE width="80%" bgcolor="black" >
<TR bgcolor="#c0c0c0"><TH>Building Permit</TH><TH>SNN</th><th>Status</th><TH>Action</TH></TR>
<%
  java.util.Vector WLResult = (java.util.Vector) session.getAttribute("WorkListItems");
  CMBItem items[] = (CMBItem[]) WLResult.elementAt(0);
  CMBWorkPackageICM wpItems[] = (CMBWorkPackageICM[]) WLResult.elementAt(1);

   // For every CMBItem in the array:
   for (int i = 0; i < items.length; ++i) {
     // List Only Application Packages of type CMB_TYPE_FOLDER
     if(items[i].getItemType() == CMBBaseConstant.CMB_TYPE_FOLDER ) { %>
     <TR bgcolor="white">
       <TD>
         <%=items[i].getAttrValue("BP_NUMBER") %>
       </TD>
       <TD>
         <%=items[i].getAttrValue("BP_TAX_ID") %>
       </TD>
       <TD>
         <%=items[i].getAttrValue("BP_STATUS") %>
       </TD>
       <TD>
          <A
HREF="RBController?cmd=RBICM&subcmd=FolderDetail&nextPage=RBEditFolder.jsp&WorkList=<%=request.
getParameter("WorkList")%>&PID=<%=java.net.URLEncoder.encode(items[i].getPidString())%>&wpPID=<
%=java.net.URLEncoder.encode(wpItems[i].getPidString())%>&uix=<%=i%>">
             Open Folder
          </A>
       </TD>
      </TR>
      <% // if the folder was rejected, display the reason.
        if( request.getParameter("WorkList").equals("BPRejected") ) { %>
        <TR bgcolor="#c0c0c0"><TH>Rejected Reason:</TH>
            <TD colspan="3"><%=items[i].getAttrValue("BP_REJ_DESC")%></TD>
        </TR>
        <% } %>

      <%}// If%>
    <%} // For%>
    </table>
<% } // Main if %>
<HR>
</BODY>
</HTML>
```

Note that the RBController is called before rendering RBReviewFolderMenu.jsp. One of the generated beans is retrieved using the JSP Tag:

```
<jsp:useBean id="WLVector" class="java.util.Vector"
scope="session"></jsp:useBean>
```

The others are retrieved directly from the session object:

```
java.util.Vector WLResult = (java.util.Vector)
session.getAttribute("WorkListItems");
CMBItem items[] = (CMBItem[]) WLResult.elementAt(0);CMBWorkPackageICM wpItems[]
= (CMBWorkPackageICM[]) WLResult.elementAt(1);
```

You choose the best way to retrieve a bean from the session object for your application.

The first part of the JSP renders a combo box with the available worklists names (see Figure 8-21).

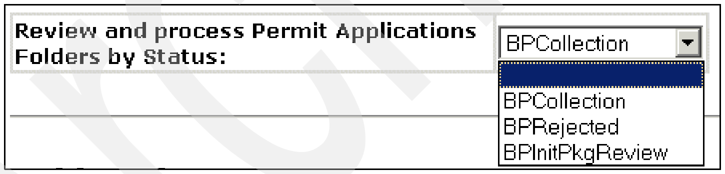
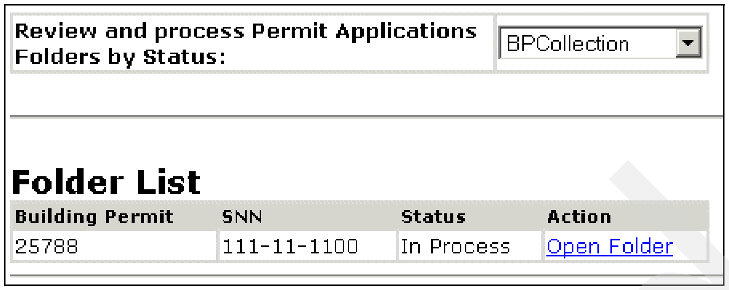


Figure 8-21 List available worklist names

If a worklist is selected, the second part of the JSP file lists the documents related to that workList (see Figure 8-22).


Review and process Permit Applications Folders by Status:

BPCollection

Folder List

Building Permit	SNN	Status	Action
25788	111-11-1100	In Process	Open Folder

Figure 8-22 Worklist detail

Before testing your code, update the RBIndex.jsp file to call this new module by add code shown in Example 8-37.

Example 8-37 RBIndex.jsp: Update

```
<LI>
    <A
href="RBController?cmd=RBICM&subcmd=getWorkLists&nextPage=RBReviewFolderMenu.js
p">
      Process Permit Applicationsfolders
    </A>
</LI>
```

Right-click the *RBIndex.jsp* file and select *Run on Server* from the context menu. If no building permit folder has been created, create one and list it from the first step at the Document Routing process *RBCollection*.

Right now we are not able to advance the folder from the Web interface or even open the folder. The link is already defined, but it is not working yet. You can work with the Content Manager Windows Client to advance the folder from the RBCollection work node to the next node by importing three files, one for each of the created item types: BP_APPLICATION, BP_DRAWING and BP_PLOT. Move the folder through the folder definition and verify that your Web interface is listing the folder in the right worklist.

Implement the process action functions

Comparing the “Building permit application approval process overview” on page 241 with the actual Document Routing approval process, we need to implement additional business logics on top of the Document Routing definition:

1. If the application package is rejected, the Supervisor must input a “reject reason” and this information must be written down to an item type attribute.
2. Once the application package has been *accepted*, the DB2 legacy system must be updated as well.

This business logics are implemented in a fixed mode for this sample application; but the main idea is to be open to implement specific business logics in order to extent the Document Routing deployment capabilities.

The sequence diagram shown in Figure 8-23 illustrates the designed interaction between the objects in order to implement these features.

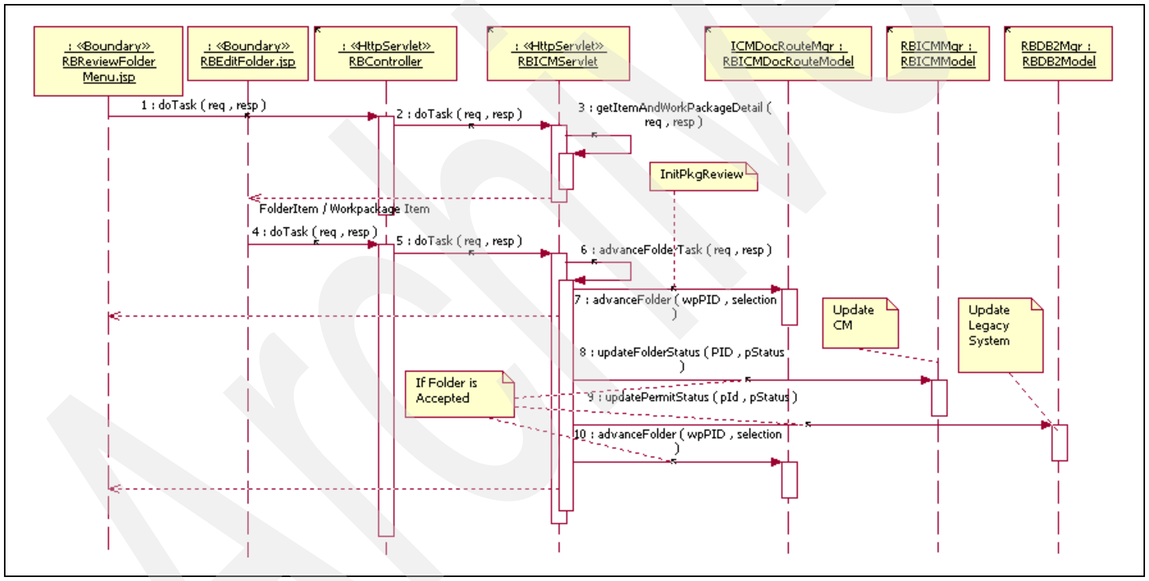

: «Boundary» RBReviewFolderMenu.jsp
: «Boundary» RBEditFolder.jsp
: «HttpServlet» RBController
: «HttpServlet» RBICMServlet
ICMDocRouteMgr : RBICMDocRouteModel
RBICMMgr : RBICMModel
RBDB2Mgr : RBDB2Model
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : getItemAndWorkPackageDetail (req , resp)
InitPkgReview
FolderItem / Workpackage Item
4 : doTask (req , resp)
5 : doTask (req , resp)
6 : advanceFolderTask (req , resp)
7 : advanceFolder (wpPID , selection)
Update CM
Update Legacy System
8 : updateFolderStatus (PID , pStatus)
If Folder is Accepted
9 : updatePermitStatus (pId , pStatus)
10 : advanceFolder (wpPID , selection)

Figure 8-23 Process application package: InitPkgReview and Accepted actions.

The cases for InitPkgReview and Accepted action are shown in Figure 8-23, the rejected action is detailed at Figure 8-24.

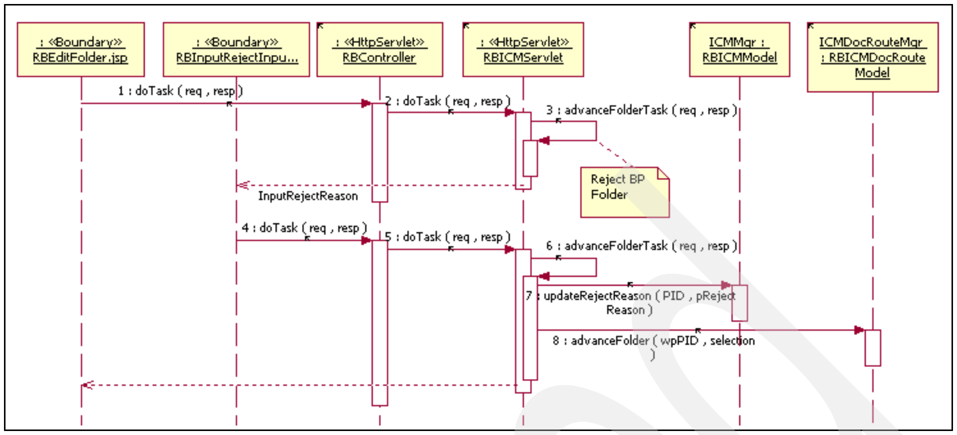

: «Boundary» RBEditFolder.jsp
: «Boundary» RBInputRejectInpu...
: «HttpServlet» RBController
: «HttpServlet» RBICMServlet
ICMMgr : RBICMModel
ICMDocRouteMgr : RBICMDocRouteModel
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : advanceFolderTask (req , resp)
Reject BP Folder
InputRejectReason
4 : doTask (req , resp)
5 : doTask (req , resp)
6 : advanceFolderTask (req , resp)
7 : updateRejectReason (PID , pRejectReason)
8 : advanceFolder (wpPID , selection)

Figure 8-24 Process application package: reject action.

Model layer

At his time we do not create any new Class definitions; we just need to implement new methods or update the existing ones.

RBICMDocRouteModel.java

Methods required:

- public void advanceFolder(String wPID, String selection): This is already implemented, so we do nothing here.

RBICMModel.java

Methods required:

- public void updateRejectReason(String PID, String pRejectReason): Write down the reject reason description within the item type attribute RB_REJ_DESC.
- public void updateFolderStatus(String PID, String pStatus): Write down the folder status definition within the item type attribute BP_STATUS.

Update RBICMModel.java with the two new methods mentioned above. Cut and paste the code in Example 8-38 for the updateRejectReason method, and the code in Example 8-39 for the updateFolderStatus method. Pay attention to the bold text to make sure you understand how the code works.

Example 8-38 RBICMModel.java - updateRejectReason method

```
public void updateRejectReason(String PID, String pRejectReason)
   throws DKException, Exception {

   DKDDO ddoFolder;
   if(dsICM == null) {
      connect();
   }
   //--------------------------------------------------
   // Create a DKDDO (ddoFolder) using the passed in PID string
   ddoFolder = dsICM.createDDO(PID);
   // retrieve the current attribute information and
   // check the item out.
   ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ATTRONLY |
      DKConstant.DK_CM_CHECKOUT);
   // Get Attribute id
   short dataId = ddoFolder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
      "BP_REJ_DESC");
   // Update data
   ddoFolder.setData(dataId, pRejectReason);
   // Update object in persistence storage
   dsICM.updateObject(ddoFolder, DKConstant.DK_CM_CHECKIN);

   this.disconnect();
}
```

Example 8-39 RBICMModel.java - updateFolderStatus method

```
public void updateFolderStatus(String PID, String pStatus)
   throws DKException, Exception {

   DKDDO ddoFolder;
   if(dsICM == null) {
      connect();
   }
   //--------------------------------------------------
   // Create a DKDDO (ddoFolder) using the passed in PID string
   ddoFolder = dsICM.createDDO(PID);
   // retrieve the current attribute information and
   // check the item out.
   ddoFolder.retrieve(DKConstant.DK_CM_CONTENT_ATTRONLY |
DKConstant.DK_CM_CHECKOUT);
   // Get Attribute ID
   short dataId = ddoFolder.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,
"BP_STATUS");
   // Update data
   ddoFolder.setData(dataId, pStatus);
   // Update object in persistence storage
   dsICM.updateObject(ddoFolder, DKConstant.DK_CM_CHECKIN);

   this.disconnect();
}
```

Controller layer

In the controller layer, we need to update the RBICMServlet.java file to handle the new business logics.

RBICMServlet.java

We need to make the following updates to the file:

- Implement the method getItemAndWorkPackageDetail() and define the task "FolderDetail."
- Implement the method advanceFolderTask() and define the task "advanceFolder."

These methods use the defined methods at the model layer. To implement the methods in RBICMServlet.java, update the doTask method by adding the evaluations as shown in Example 8-40. Create a new method, getItemAndWorkPackageDetail, as shown in Example 8-41.

Example 8-40 RBICMServlet.java - Updated code for doTask method

```
if(subcmd != null && subcmd.equals("FolderDetail") ) {
   this.getItemAndWorkPackageDetail(req,resp);
}

if(subcmd != null && subcmd.equals("advanceFolder") ) {
   this.advanceFolderTask(req,resp);
}
```

Example 8-41 RBICMServlet.java - getItemAndWorkPackageDetail method

```
private void getItemAndWorkPackageDetail(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   Vector WLItems = (Vector)
      req.getSession().getAttribute("WorkListItems");
   String suix    =
      req.getParameter("uix");
   RBICMModel ICMMgr = (RBICMModel)
      req.getSession().getAttribute("ICMMgr");
   if( ICMMgr == null ) {
      ICMMgr = new RBICMModel();
      ICMMgr.setConnectParams(
            (String)req.getSession().getAttribute("CMSERVER"),
            (String)req.getSession().getAttribute("CMUSER"),
            (String)req.getSession().getAttribute("CMPASSWORD"));
      req.getSession().setAttribute("ICMMgr", ICMMgr);
   }

   try {
```

```
Vector resultTable = new Vector();
// Verify if the data needed is already on the Session Object
if( (WLItems != null) && (WLItems.size() > 0) && (suix != null) ) {
   int uix = new Integer(suix).intValue();
   CMBItem Items[] = (CMBItem[]) WLItems.elementAt(0);
   CMBWorkPackageICM WPI[] = (CMBWorkPackageICM[]) WLItems.elementAt(1);

   // Prepare display data
   resultTable.add(0, Items[uix] );
   resultTable.add(1, WPI[uix]);

   // Get Items child's part for items display
   Vector ChildItems = Items[uix].listFolderItems();
   DKLobICM DKLobs[] = new DKLobICM[ChildItems.size()];

   for(int i=0 ; i < ChildItems.size(); i++ ) {

      // Get Folder child item's part ID
      DKDDO ddo = ((CMBItem)ChildItems.elementAt(i)).getDDO();

      ddo.retrieve(DKConstant.DK_CM_CONTENT_ONELEVEL);

      // Create a short (dataid) and set it to the ID (1 to n)
      //of the data item whose name
      // was passed in the variable (dataName)
      short dataid = ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"DKParts");

      // if a DKParts collection exists, print contents
      if (dataid > 0) {
         // Create a DKParts (dkParts) and set it to the
         // data that is the parts collection
         DKParts dkParts = (DKParts) ddo.getData(dataid);

         // Create an dkIterator (iter) to go through Collection
         dkIterator iter = dkParts.createIterator();
         // get first part
         if (iter.more()) {
            // Create a DKLobICM (part) and set it
            // to the next one ine the collection.
            DKLobICM part = (DKLobICM) iter.next();

            // retrieve the part's information, but not the content
            part.retrieve(DKConstant.DK_CM_CONTENT_NO);
            DKLobs[i] = part;
         }
      }
   }

   // prepare display data, see RBEditFolder.jsp
```

```
         resultTable.add(2,DKLobs);
         // Make it available for JSP's Display
         req.getSession().setAttribute("BPFolder",resultTable);

      } else {

         // Retrieve CMBItem and CMBWorkPackgeICM using POST Parameters
         // To be implemented if application flow requieres it.
         // leaved as an optional exercise for the reader.

      }
   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

The getItemAndWorkPackageDetail method retrieves the building permit folder detail and all the related documents within the folder. Because of the Web application flow, we can use the Folder *CMBItem* in memory (Session object) to retrieve the additional item types and parts. If the application flow requires it, it should be necessary to implement the query within the repository. This part of the code is left as an optional exercise to the reader.

Implement the advanceFolderTask() method using the code in Example 8-42.

Example 8-42 RBICMServlet.java - advanceFolderTask method

```
private void advanceFolderTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   RBICMDocRouteModel Mgr = (RBICMDocRouteModel)
         req.getSession().getAttribute("ICMDocRouteMgr");
   RBSessionBean sb = (RBSessionBean)
         req.getSession().getAttribute("SessionBean");

   try {
      // Identify Reject selection for extra routing tasks to be done
      if(req.getParameter("selection").equals("Reject")) {
         // has the Reject Reason been entered?
         if( req.getParameter("RejectReason") == null ) {
            nextPage = req.getParameter("InputReasonPage");
            return;
         }
         //Update FolderItem reject reason using RBICMModel
         RBICMModel ICMMgr = (RBICMModel)
               req.getSession().getAttribute("ICMMgr");
         if( ICMMgr == null ) {
            ICMMgr = new RBICMModel();
```

```
        ICMMgr.setConnectParams(
            (String)req.getSession().getAttribute("CMSERVER"),
            (String)req.getSession().getAttribute("CMUSER"),
            (String)req.getSession().getAttribute("CMPASSWORD"));

        req.getSession().setAttribute("ICMMgr", ICMMgr);
    }

    ICMMgr.updateRejectReason(
        req.getParameter("PID"),
        req.getParameter("RejectReason") );

} // IF Reject

// Identify Accept selection for extra routing tasks to be done
if(req.getParameter("selection").equals("Accept")) {
    //Update FolderItem Status using RBICMModel
    RBICMModel ICMMgr = (RBICMModel) req.getSession().getAttribute("ICMMgr");
    if( ICMMgr == null ) {
        ICMMgr = new RBICMModel();
        ICMMgr.setConnectParams(
            (String)req.getSession().getAttribute("CMSERVER"),
            (String)req.getSession().getAttribute("CMUSER"),
            (String)req.getSession().getAttribute("CMPASSWORD"));
        req.getSession().setAttribute("ICMMgr", ICMMgr);
    }

    ICMMgr.updateFolderStatus(
            req.getParameter("PID"),
            "Application Accepted" );
    // Update legacy System using DB2Model
    RBDB2Model DB2Mgr = (RBDB2Model) req.getSession().getAttribute("DB2Mgr");
    if( DB2Mgr == null ) {
        DB2Mgr = new RBDB2Model();
        DB2Mgr.setConnectParams(
            (String)req.getSession().getAttribute("DB2DB"),
            (String)req.getSession().getAttribute("DB2USER"),
            (String)req.getSession().getAttribute("DB2PASSWORD"),
            "");
        req.getSession().setAttribute("DB2Mgr", DB2Mgr);
    }
    // update building permit record to "Y" (Approved)
    DB2Mgr.updatePermitStatus(req.getParameter("BPNumber"),"Y");
} // IF Accept

  // Use RBICMDocRouteModel to advance BP folder
  Mgr.advanceFolder(
    req.getParameter("wpPID"),
    req.getParameter("selection"));
```

```
      // Set user respond message
      sb.setGeneralMessage("The folder has been advanced successfully");
      sb.setGeneralMessage(true);
    } catch (Exception e) {
      e.printStackTrace();
      sb.setErrorOccurred(true);
      sb.setErrorMessage("An error occured during folder workflow advance.");
  }
    req.getSession().setAttribute("SessionBean",sb);
}
```

Review the sequence diagrams presented in Figure 8-23 on page 279 and Figure 8-24 on page 280 for a graphical explanation of the advanceFolderTask() method.

View layer

The boundary interfaces related to this use case are:

- RBReviewFolderMenu.jsp (Already implemented)
- RBEditFolder.jsp. See Appendix , “View layer” on page 274
- RBInputRejectReason.jsp. Input form for the reject description at the “reject” process action.

The call link to the RBEditFolder.jsp is already defined at the RBReviewFolderMenu.jsp, so use the following sample code to implement the RBEditFolder.jsp within the WebContent folder, the same way as explained for the last JSP file.

(The lengthy JSP file shown in Example 8-43 is explained after the example.)

Example 8-43 RBEditFolder.jsp

```
<jsp:useBean id="BPFolder" class="java.util.Vector" scope="session">
</jsp:useBean>

<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<%@ page import="com.ibm.mm.beans.workflow.*"%>
<%@ page import="itso.rb.intranet.model.*"%>
<%@ page import="com.ibm.mm.sdk.common.*"%>

<%
   CMBItem FolderItem = (CMBItem) BPFolder.elementAt(0);
   CMBWorkPackageICM WPItem = (CMBWorkPackageICM) BPFolder.elementAt(1);
   DKLobICM DKLobs[]  = (DKLobICM[]) BPFolder.elementAt(2);
   Vector ChildItems = FolderItem.listFolderItems();
%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<!-- <script language="JavaScript" src="cmv8client.js"></script> -->
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>RBEditFolder.jsp</TITLE>
<SCRIPT language="JavaScript">

function setSelection(pSelection) {
   document.advanceForm.selection.value = pSelection;
   document.advanceForm.submit();
}

function setItemType(pItemType) {
   document.docUpload.ItemType.value = pItemType;
   //document.docUpload.submit();
}

</SCRIPT>
</HEAD>
<BODY>
<H1>Building Permit Application Folder Details</H1>
<HR>
<P>[ <A href="RBIndex.jsp">Home</A> ]
   [ <A
     href="RBController?cmd=RBICM&subcmd=getWorkLists&nextPage=RBReviewFolderMenu.jsp">
   Back to the list</A> ]
</P>
<P><BR></P>
<TABLE border="1">
   <TBODY>
      <TR>
        <TH colspan="2">Folder Information</TH>
      </TR>
      <TR>
         <TH>SSN</TH>
         <TD><%=FolderItem.getAttrValue("BP_TAX_ID")%></TD>
      </TR>
      <TR>
         <TH>BP Number</TH>
         <TD><%=FolderItem.getAttrValue("BP_NUMBER")%></TD>
      </TR>
      <TR>
         <TH>STATUS</TH>
         <TD><%=FolderItem.getAttrValue("BP_STATUS")%></TD>
      </TR>
      <TR>
         <TH>Step</TH>
         <TD><%=WPItem.getWorkNodeName()%></TD>
      </TR>
   </TBODY>
</TABLE>
<HR>

<Form name="docUpload" action="RBController" method="post" >
<INPUT TYPE="hidden" NAME="ItemType" VALUE="">
<INPUT TYPE="hidden" NAME="BP_NUMBER" VALUE="<%=FolderItem.getAttrValue("BP_NUMBER")%>">
<INPUT TYPE="hidden" NAME="cmd" VALUE="RBICM">
<INPUT TYPE="hidden" NAME="subcmd" VALUE="prepareUpload">
<INPUT TYPE="hidden" NAME="nextPage" VALUE="RBImportItem.jsp">
<!-- parameters needed to come back to this page -->
<INPUT TYPE="hidden" NAME="WorkList" VALUE="<%=request.getParameter("WorkList")%>">
<INPUT TYPE="hidden" NAME="PID" VALUE="<%=request.getParameter("PID")%>">
<INPUT TYPE="hidden" NAME="wpPID" VALUE="<%=request.getParameter("wpPID")%>">
<INPUT TYPE="hidden" NAME="uix" VALUE="<%=request.getParameter("uix")%>">


<DIV align="center">
<TABLE border="1" width="80%">
   <TBODY>
   <TR>
      <TH width="30%">BP Application(s)</TH>
      <TH align="right">
        <INPUT type="button" name="AddApp" value="Add document"
         onClick="setItemType('BP_APPLICATION')" >
        <INPUT type="button" name="DelApp" value="Delete document">
      </TH>
   </TR>
   </TBODY>
</TABLE>
</DIV>
<DIV align="center">
<TABLE border="1" width="80%">
   <TBODY>
   <TR>
      <TD></TD>
      <!-- You could also retrive the metadata dynamically -->
      <TH>BP Number</TH>
      <TH>Document Description</TH>
      <TH>Submission Date</TH>
   </TR>
   <% for( int i=0 ; i < ChildItems.size() ; i++ ) {
        CMBItem CItem = (CMBItem)ChildItems.elementAt(i);
        String pid = CItem.getPidString();
        if( CItem.getEntityName().equals("BP_APPLICATION") ) { %>
          <TR>
          <TD><INPUT type="checkbox" name="App" value="1">
          <% String MT = DKLobs[i].getMimeType();%>
            <a 
href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(DKLobs[i].getPidObject().pidString())%>
"
         target="_blank">View</A>
      </TD>
      <TD><%=CItem.getAttrValue("BP_NUMBER")%></TD>
      <TD><%=CItem.getAttrValue("BP_DOC_DESC")%></TD>
      <TD align="center"><%=CItem.getAttrValue("BP_SUB_DATE")%></TD>
      </TR>
       <% } // if
   } // for %>
   </TBODY>
</TABLE>
</DIV>
<HR>
<DIV align="center">
<TABLE border="1" width="80%">
   <TBODY>
   <TR>
   <TH width="30%">BP Drawings</TH>
   <TH align="right">
     <INPUT type="button" name="AddApp" 
      value="Add document" onClick="setItemType('BP_DRAWING')">
     <INPUT type="button" name="DelApp" 
      value="Delete document">
   </TH>
   </TR>
   </TBODY>
</TABLE>
</DIV>
<DIV align="center">
<TABLE border="1" width="80%">
   <TBODY>
   <TR>
     <TD></TD>
     <TH>BP Number</TH>
     <TH>Document Description</TH>
```

```
      <TH>Submission Date</TH>
      <TH>Version</TH>
    </TR>
    <% for( int i=0 ; i < ChildItems.size() ; i++ ) {
         CMBItem CItem = (CMBItem)ChildItems.elementAt(i);
         String pid = CItem.getPidString();
         if( CItem.getEntityName().equals("BP_DRAWING") ) { %>
        <TR>
        <TD><INPUT type="checkbox" name="App" value="1">
        <% String MT = DKLobs[i].getMimeType(); %>
          <a

href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(DKLobs[i].getPidObject().pidString())%>
"
            target="_blank">View</A>
        </TD>
        <TD><%=CItem.getAttrValue("BP_NUMBER")%></TD>
        <TD><%=CItem.getAttrValue("BP_DOC_DESC")%></TD>
        <TD align="center"><%=CItem.getAttrValue("BP_SUB_DATE")%></TD>
        <TD align="center"><%=CItem.getAttrValue("BP_DOC_VERSION")%></TD>
        </TR>
      <% }
    } %>
    </TBODY>
</TABLE>
</DIV>
<HR>
<DIV align="center">
<TABLE border="1" width="80%">
    <TBODY>
    <TR>
        <TH width="30%">BP Plots</TH>
        <TH align="right">
         <INPUT type="button" name="AddApp"
           value="Add document"
           onClick="setItemType('BP_PLOT')">
         <INPUT type="button"
           name="DelApp" value="Delete document">
        </TH>
    </TR>
    </TBODY>
</TABLE>
</DIV>
<DIV align="center">
<TABLE border="1" width="80%">
    <TBODY>
    <TR>
        <TD></TD>
        <TH>BP Number</TH>
        <TH>Document Description</TH>
        <TH>Submission Date</TH>
        <TH>Version</TH>
    </TR>
    <% for( int i=0 ; i < ChildItems.size() ; i++ ) {
         CMBItem CItem = (CMBItem)ChildItems.elementAt(i);
         String pid = CItem.getPidString();
         if( CItem.getEntityName().equals("BP_PLOT") ) {
     %>
      <TR>
        <TD><INPUT type="checkbox" name="App" value="1">
        <% String MT = DKLobs[i].getMimeType(); %>
        <a

href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(DKLobs[i].getPidObject().pidString())%>
"
        target="_blank">View</A>
        </TD>
        <TD><%=CItem.getAttrValue("BP_NUMBER")%></TD>
        <TD><%=CItem.getAttrValue("BP_DOC_DESC")%></TD>
        <TD align="center"><%=CItem.getAttrValue("BP_SUB_DATE")%></TD>
        <TD align="center"><%=CItem.getAttrValue("BP_DOC_VERSION")%></TD>
    </TR>
      <% }
     } %>
    </TBODY>
</TABLE>
</DIV>
</FORM>
<!-- DOCUMENT ROUTING ACTION FORMS -->
<HR>
<DIV align="center">
<Form name="advanceForm" action="RBController" method="POST">
<input type="hidden" name="wpPID"
    value="<%=(String) java.net.URLDecoder.decode(request.getParameter("wpPID")) %>">
<input type="hidden" name="PID"   value="<%= FolderItem.getPidString()%>">
<input type="hidden" name="selection" value="">
<input type="hidden" name="nextPage" value="RBIndex.jsp">
<input type="hidden" name="cmd" value="RBICM">
<input type="hidden" name="subcmd" value="advanceFolder">
<input type="hidden" name="InputReasonPage" value="RBInputRejectReason.jsp">
<input type="hidden" name="BPNumber" value="<%=FolderItem.getAttrValue("BP_NUMBER") %>">
<TABLE border="1" width="80%">
    <TBODY>
    <TR>
        <TH colspan="2">Document Routing Actions</TH>
    </TR>
    <TR>
        <TH>Current Step: <%=WPItem.getWorkNodeName()%></TH>
```

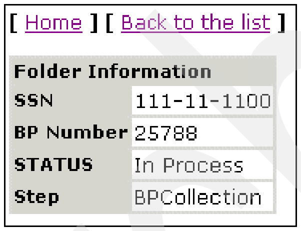
```
        <TD><% String SDesc = (String)
           ((Hashtable)session.getAttribute("StatusDesc")).get( WPItem.getWorkNodeName() );
            if( SDesc == null) SDesc = "No status description available"; %>
         <%=SDesc%>
        </TD>
    </TR>
    <% // If is on the rejected list, display description
    if( WPItem.getWorkNodeName().equals("BPRejected") ) { %>
        <TR>
          <TH>Rejected reason: </TH>
          <TD><%=FolderItem.getAttrValue("BP_REJ_DESC")%></TD>
        </TR>
    <% } %>
        <TR>
           <TD colspan="2" align="center">
           <% if( WPItem.getWorkNodeName().equals("BPInitPkgReview") ) { %>
              <INPUT type="button"
               onClick="setSelection('FinalReview')"
               name="SubmitToRevision"
               value="Folder Completed - Submit to final review">
           <% }
                if( WPItem.getWorkNodeName().equals("BPPkgReview") ) { %>
              <INPUT type="button"
               name="Accept" value="   Accept application package  "
               onClick="setSelection('Accept')">
              <INPUT type="button" name="Reject"
               value="            REJECT             "
               onClick="setSelection('Reject')">
            <% }
                if( WPItem.getWorkNodeName().equals("BPRejected") ) { %>
               <INPUT type="button"
           name="Accept" value="   Submit for new revision  "
           onClick="setSelection('NewReview')">
            <% } %>
          </TD>
        </TR>
    </TBODY>
</TABLE>
</FORM>
</DIV>
</BODY>
</HTML>
```

This JSP file (Example 8-43 on page 286) has the objectives of visualizing the building permit folder details, listing the related documents, selecting the process Document Routing actions, and later, importing new items to the folder. There are three broken link warnings referring to the /IntranetApp/ICMOpenResource servlet; this will be implemented later, so just ignore the warnings for now.

The main object here, retrieved from the session object, is *FolderItem* of the class *CMBItem*. This class is part of the Java Beans APIs, and the method *listFolderItems()* is allowing us to render the folder details in a very simple fashion. Coding the same functionality, using the Information Integrator for Content APIs at the View Layer, could result in many lines of Java Code at our JSP file, something that usually we try to avoid.

```
<%
   CMBItem FolderItem = (CMBItem) BPFolder.elementAt(0);
   CMBWorkPackageICM WPItem = (CMBWorkPackageICM) BPFolder.elementAt(1);
   DKLobICM DKLobs[]  = (DKLobICM[]) BPFolder.elementAt(2);
   Vector ChildItems = FolderItem.listFolderItems();
%>
```

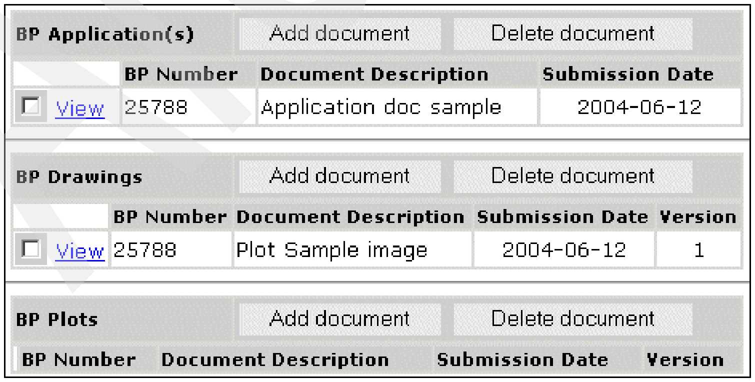
The first part of the JSP renders the BP Folder general information (Figure 8-25).


[Home] [Back to the list]

Folder Information	
SSN	111-11-1100
BP Number	25788
STATUS	In Process
Step	BPCollection

Figure 8-25 BP Folder general information

The next JSP part lists the related documents (Figure 8-26).


BP Application(s) Add document Delete document

	BP Number	Document Description	Submission Date
☐ View	25788	Application doc sample	2004-06-12

BP Drawings Add document Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	25788	Plot Sample image	2004-06-12	1

BP Plots Add document Delete document

BP Number	Document Description	Submission Date	Version

Figure 8-26 BP Folder's documents

And last but not least, these are the Document Routing process action options (Figure 8-27).

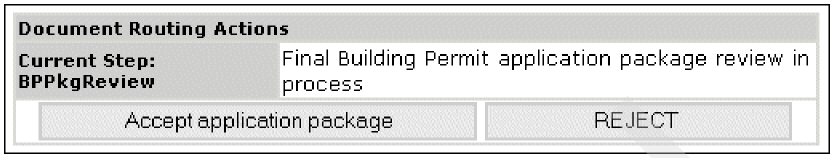

Document Routing Actions
Current Step: BPPkgReview
Final Building Permit application package review in process
Accept application package
REJECT

Figure 8-27 BP Folder's process actions

During the Document Routing process actions, we are using a custom workflow step description; this small table is retrieved from the session object and is initialized at the *RBController.java* file. Open this file and update the *initSessionSetup()* method, and add the following code to the end of the method, but before the "*SessionInit*" attribute definition (Example 8-44).

Example 8-44 RBController.initSessionSetup() update

```
   // Process Status description table
   java.util.Hashtable statusTable = new java.util.Hashtable();

   statusTable.put("BPCollection",(String) Config.getObject("BPCollection"));
   statusTable.put("BPInitPkgReview",(String)
Config.getObject("BPInitPkgReview"));
   statusTable.put("BPPkgReview",(String) Config.getObject("BPPkgReview"));
   statusTable.put("BPRejected",(String) Config.getObject("BPRejected"));
   req.getSession().setAttribute("StatusDesc",statusTable);

   req.getSession().setAttribute("SessionInit","ok");
}
```

The values are taken from the Config.properties file. Open this file and add the lines in Example 8-45 at the end of the file.

Example 8-45 Config.properties file

```
# RedBrook County approval steps description
BPCollection=Application package incomplete. Waiting documents to be submitted.
BPInitPkgReview=Application package completed. Technical review in process
BPPkgReview=Final Building Permit application package review in process
BPRejected=Application package rejected, review comments for details
```

During the BP Folder approval process, the reject action requests the user to input a reject reason description; we use the *RBInputRejectReason.jsp* to implement this task.

Now create a new JSP File at the *WebContent* folder and overwrite the default code with the coding in Example 8-46.

Example 8-46 RBInputRejectReason.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet" type="text/css">
<SCRIPT language="JavaScript">
function VerifyForm() {
   if( document.advanceForm.RejectReason.value == "" ) {
     alert("Please enter a reject reason");
     return;
   }

   document.advanceForm.submit();
}
</SCRIPT>
<BODY>
<H1>Input reject reason</H1>
<HR>
<Form name="advanceForm" action="RBController" method="POST">
<input type="hidden" name="wpPID" value="<%= request.getParameter("wpPID")%>">
<input type="hidden" name="PID" value="<%= request.getParameter("PID")%>">
<input type="hidden" name="selection" value="<%= request.getParameter("selection")%>">
<input type="hidden" name="nextPage" value="<%= request.getParameter("nextPage")%>">
<input type="hidden" name="cmd" value="<%= request.getParameter("cmd")%>">
<input type="hidden" name="subcmd" value="<%= request.getParameter("subcmd")%>">
<input type="hidden" name="InputReasonPage" value="<%= request.getParameter("wpPID")%>">
<input type="hidden" name="BPNumber" value="<%= request.getParameter("BPNumber")%>">
<table>
 <TR>
   <TH>Input reject reason</TH>
 </TR>
 <TR>
   <TD><TEXTAREA rows="3" cols="45" name="RejectReason"></TEXTAREA></TD>
 </TR>
 <TR>
   <TD align="right">
      <input type="button" value=" Continue "
      onclick="VerifyForm()"></TD>
   </TR>
</table>
</FORM>
<hr>
</BODY>
</HTML>
```

This page is called by the controller servlet when the reject action is submitted. The rendered view looks like the image in Figure 8-28.

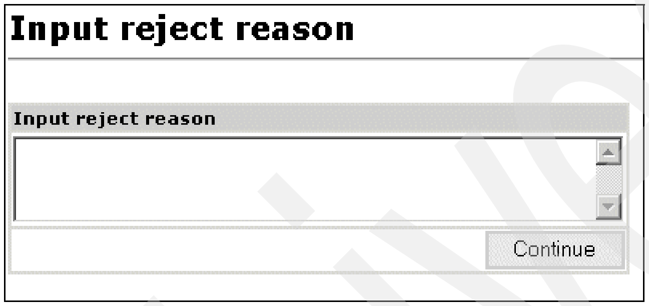


Figure 8-28 RBInputRejectReason.jsp

Test

At this point, the Web sample application is capable of doing almost the whole building permit application approval process. As defined, the folder is initialized at the BPCollection point. In order to get to the next step in the workflow, it is necessary to have at least one document at the BP_APPLICATION, BP_DRAWING, and BP_PLOT item types.

Create a folder using the sample application and use the IBM Content Manager Client for Windows to import the documents to those item types using the sample BP_Number value. The following use case enables our sample application to import the files to the folder.

Use the process action buttons at the sample application to process the BP Folder. Once the BP Folder has been approved, verify that the DB2 legacy System has been updated as well.

Note that if you have implemented the ACL definitions, you use the RBClerk user to define the BP Folder, but to accept or reject the application logged with the RBSupervisor, this is done by the user within the Web sample application.

In order to reset the data, make that sure you:

- Delete the created folders and documents from the IBM Content Manager System.
- Update the itso.building_permit table, and set the certificate value to ‘N’:

```
update itso.building_permit set certificate = ‘N’
```

8.5.6 Document import use case

Now that we are able to create, initialize, and process our building permit folder at the Document Routing process from the Web sample application, we still remain on a third application to import the related BP Folder’s documents.

This use case is the final setup to complete the building permit folder approval process general use case. See Table 8-9.

Description

Table 8-9 Building permit folder approval process general use case

Element	Description
Name	Document import use case.
Description	Add documents such as building permit application, drawings and plots to the building permit application folder to be processed.
Actors	Clerk, Supervisor, IBM Content Manager system.
Inputs	Building permit application folder attributes, item attributes values and content file.
Pre-conditions	The building permit application folder has been created in IBM Content Manager and initialized within the document route approval process.
Steps	1. The clerk/supervisor user logs in to the RedBrook intranet system. 2. Selects process application folders from the menu. 3. Selects one of the available worklists and displays the existing BP Folders to be processed. 4. Opens the BP Folder detail and selects “Add document” from one of the document list table forms. 5. The user inputs the attribute values and selects the file to be uploaded.
Alternatives	

Element	Description
Post-conditions	The new imported file has been linked to the main building permit folder.

Design

We now consider the design aspects of this case (Figure 8-29).

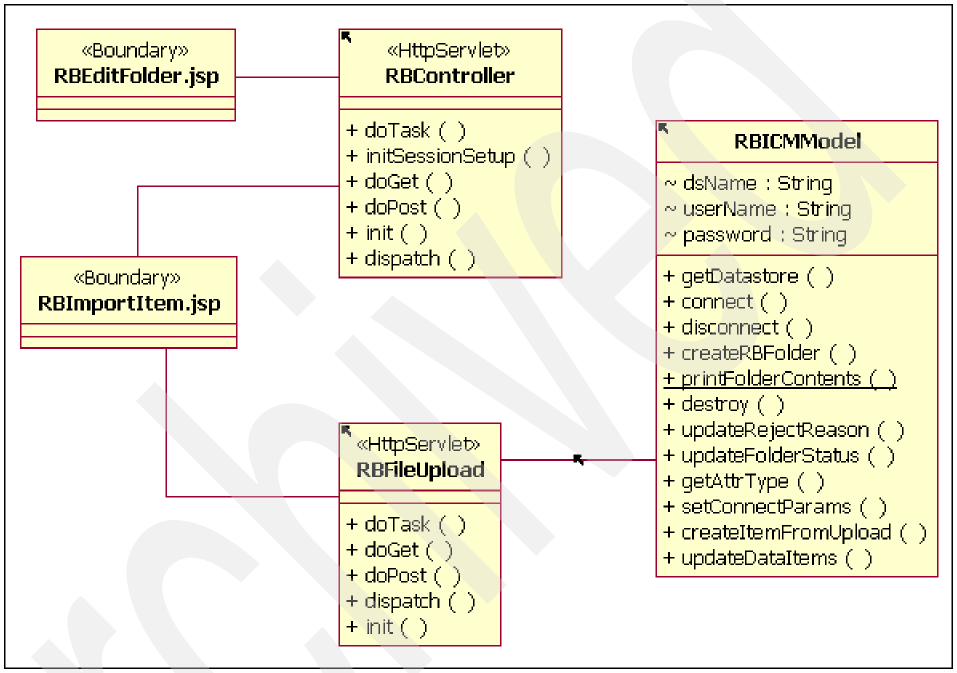


Figure 8-29 Import Document use case - class diagram

RBImportItem.jsp	Input item type attribute's values form. Renders metadata display dynamically using the item type definition.
RBFileUpload.java	Specialized Servlet. Processes the http request defined by the MIME media type "multipart/form-data" in order to retrieve the metadata and the file content. Uses the RBICMModel to create the new item type.

For information about the other classes, please review the past use case definitions.

In order to have a better idea about the collaboration between this objects, review the following sequence diagram and identify the new objects or business methods to be implemented (Figure 8-30).

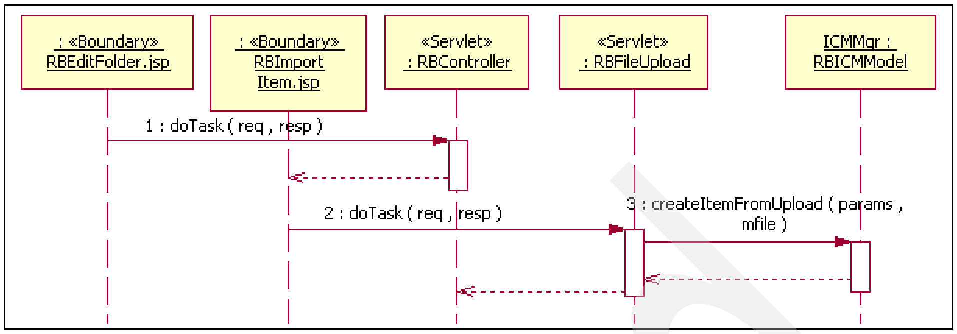

: «Boundary» RBEditFolder.jsp
: «Boundary» RBImport Item.jsp
«Servlet» : RBController
«Servlet» : RBFileUpload
ICMMgr : RBICMModel
1 : doTask (req , resp)
2 : doTask (req , resp)
3 : createItemFromUpload (params , mfile)

Figure 8-30 Import document use case - sequence diagram

Model layer

Once again, update the model layer with the following definitions:

RBICMModel.java:

- createItemFromUpload(Hashtable,byte[]): Creates a new item within IBM Content Manager system.
- updateDataItems(DKDDO,DKItemTypeDefICM, Hashtable): Helper method, retrieves the attribute values from the hash table data structure and casts them to the needed DKDDO Item type definition.
- getAttrType(short): Helper method, translates the attribute type short value to a String value.

Open the RBICMModel.java file and use the following coding examples to define the methods.

First, consider the coding in Example 8-47.

Example 8-47 RBICMModel.createItemFromUpload() method

```
public void createItemFromUpload(Hashtable params, byte[] mfile)
   throws DKException, Exception {

   if(dsICM == null) {
      connect();
   }

   String strEntity = (String) params.get("ItemType");
   String ddoType = (String) params.get("iType"); // Documet or Folder
   DKDDO ddo = null;

   // Create a DKDatastoreDefICM (dsDefICM) object so we can get info
   // about the chosen item type to create the ddo in.
   DKDatastoreDefICM dsDefICM = new DKDatastoreDefICM(dsICM);
   DKItemTypeDefICM entityDef = (DKItemTypeDefICM)
      dsDefICM.retrieveEntity(strEntity);
   switch (entityDef.getClassification()) {

     case DKConstantICM.DK_ICM_ITEMTYPE_CLASS_ITEM :
     case DKConstantICM.DK_ICM_ITEMTYPE_CLASS_RESOURCE_ITEM :
      // For the Item and Resource Item classification, create a
      // DKDDO (ddo) in the passed in Entity Name
      // NOTE: the object 'ddo' has already been declared above so its
      // scope is the entire try block

      ddo = dsICM.createDDO(strEntity, DKConstant.DK_CM_ITEM);
      break;

     case DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL :
               // Set type of item (Document or Folder)
      if (ddoType.equals("Document"))
        // For the doc model, when the user has selecrted "Document",
        // create a DKDDO (ddo) in the passed in Entity Name
        ddo = dsICM.createDDO(strEntity, DKConstant.DK_CM_DOCUMENT);
      else
        // For the doc model, when the user has selecrted "Folder",
        // create a DKDDO (ddo) in the passed in Entity Name
        ddo = dsICM.createDDO(strEntity, DKConstant.DK_CM_FOLDER);
      break;

     default :
      System.out.println("Unknown classification "
         + entityDef.getClassification());
      break;
   } //end switch
   // Call updateDataItems() for this new ddo
   // to update attribute values
   this.updateDataItems(ddo, entityDef, params);

   // Create Item part - ICMBASE
   DKLobICM   base = (DKLobICM)
         dsICM.createDDO("ICMBASE", DKConstantICM.DK_ICM_SEMANTIC_TYPE_BASE);
   base.setMimeType((String) params.get("mimeType"));
   base.setContent(mfile);

   // Get DDO's DKParts Attribute
   DKParts dkParts = (DKParts)

ddo.getData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS));
   short dataid = ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
   if(dataid==0)
      throw new Exception("No DKParts Attribute Found!  DDO is either not " +
       " a CM Document Model Item or Document has not been explicitly retrieved.");

   dkParts = (DKParts) ddo.getData(dataid);
   dkParts.addElement(base);


   // Now that all the data items have their values set, add the ddo to the
   // datastore to persistence store
   ddo.add();

   this.disconnect();

}
```

This method receives the metadata within a hash table data structure and the content file. It creates a new *DKDDO* definition, and sets the attribute values, casting the values following the item type definition. It creates a new *DKPart* and sets the mime type and file content, and also adds the new part to the DDO's part definition.

The method *updateDataItems()* is needed, so define this method also with the Java code in Example 8-48.

Example 8-48 RBICMModel.updateDataItems() method

```
public  void updateDataItems(
  DKDDO ddo,
  DKItemTypeDefICM entityDef,
  java.util.Hashtableparams)
  throws DKException, Exception {

  // A DKDDO, DKItemTypeDefICM object, and the Hashtable object have
  // been passed in to this routine.  With this information, we will update
  // the data items in the ddo from the parameters in the Hashtable
  // object

  // Create a DKSequentialCollection (allAttrs) containing ALL the DKAttrDefICM's for the
  // entity.
  DKSequentialCollection allAttrs = (DKSequentialCollection) entityDef.listAllAttributes();
  dkIterator iter = allAttrs.createIterator();

  while (iter.more()) {
   DKAttrDefICM aDef = (DKAttrDefICM) iter.next();   // Get the next attrDef
   // get the value of the parameter with the same name as the attr name.
   String dataValue = (String) params.get(aDef.getName());

   // Create a short (dataId) and set its value to the number from 1-n
   // of the data item that has the same name as the attribute
```

```
    short dataId = ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR, aDef.getName());

    /* -- Data Attribute Types -------------------------------------------------
      Short            DK_CM_DATAITEM_TYPE_SHORT           java.lang.Short
      Long             DK_CM_DATAITEM_TYPE_LONG            java.lang.Long
      Float            DK_CM_DATAITEM_TYPE_FLOAT           java.lang.Double
      Decimal          DK_CM_DATAITEM_TYPE_DECIMAL         java.math.BigDecimal
      Date             DK_CM_DATAITEM_TYPE_DATE            java.sql.Date
      Time             DK_CM_DATAITEM_TYPE_TIME            java.sql.Time
      Timestamp        DK_CM_DATAITEM_TYPE_TIMESTAMP       java.sql.Timestamp
      Double           DK_CM_DATAITEM_TYPE_DOUBLE          java.lang.Double
    --------------------------------------------------------------------------*/

    // switch based on the type of attribute it is (string, decimal, date, etc)

    switch (aDef.getType()) {
      case DKConstantICM.DK_CM_DATAITEM_TYPE_DATE :
       ddo.setData(dataId, java.sql.Date.valueOf(dataValue)); //"2001-08-12"
       break;
      case DKConstantICM.DK_CM_DATAITEM_TYPE_TIME :
       ddo.setData(dataId, java.sql.Time.valueOf(dataValue));
       break;
      case DKConstantICM.DK_CM_DATAITEM_TYPE_TIMESTAMP : // "2001-08-12 10:00:00.123456"
       ddo.setData(dataId, java.sql.Timestamp.valueOf(dataValue));
       break;
      case DKConstantICM.DK_CM_DATAITEM_TYPE_SHORT : //
       ddo.setData(dataId, java.lang.Long.valueOf(dataValue));
       break;
      case DKConstantICM.DK_CM_DATAITEM_TYPE_LONG : //
       ddo.setData(dataId, java.lang.Long.valueOf(dataValue));
       break;
      case DKConstantICM.DK_CM_DATAITEM_TYPE_FLOAT : //
       ddo.setData(dataId, java.lang.Double.valueOf(dataValue));
       break;
      default :
       // Set the data value as a string
       ddo.setData(dataId, dataValue); // basic string type
    }
  }
}
```

Finally, add the getAttrType(short) method as shown in Example 8-49.

Example 8-49 RBICMModel.getAttrType() method

```
public String getAttrType(short attr) {
  switch (attr) {
   case DKConstant.DK_CM_DATAITEM_TYPE_UNDEFINED :
     return ("Undefined");
   case DKConstant.DK_CM_DATAITEM_TYPE_STRING :
     return ("String");
   case DKConstant.DK_CM_DATAITEM_TYPE_SHORT :
     return ("Short");
   case DKConstant.DK_CM_DATAITEM_TYPE_LONG :
     return ("Long");
   case DKConstant.DK_CM_DATAITEM_TYPE_FLOAT :
     return ("Float");
   case DKConstant.DK_CM_DATAITEM_TYPE_DECIMAL :
     return ("Decimal");
   case DKConstant.DK_CM_DATAITEM_TYPE_DATE :
     return ("Date");
   case DKConstant.DK_CM_DATAITEM_TYPE_TIME :
     return ("Time");
   case DKConstant.DK_CM_DATAITEM_TYPE_TIMESTAMP :
     return ("Timestamp");
   case DKConstant.DK_CM_DATAITEM_TYPE_DOUBLE :
     return ("Double");
   case DKConstant.DK_CM_DATAITEM_TYPE_BYTEARRAY :
     return ("Byte  Array  (Blob)");
   case DKConstant.DK_CM_DATAITEM_TYPE_DDOOBJECT :
     return ("DDO");
   case DKConstant.DK_CM_DATAITEM_TYPE_XDOOBJECT :
     return ("XDO");
   case DKConstant.DK_CM_DATAITEM_TYPE_DATAOBJECTBASE :
     return ("Data  Object  Base");
   case DKConstant.DK_CM_DATAITEM_TYPE_COLLECTION :
     return ("Collection");
   case DKConstant.DK_CM_DATAITEM_TYPE_COLLECTION_DDO :
     return ("DDO  Collection");
   case DKConstant.DK_CM_DATAITEM_TYPE_COLLECTION_XDO :
     return ("XDO  Collection");
   case DKConstant.DK_CM_DATAITEM_TYPE_LINKCOLLECTION :
     return ("Link  Collection");
   case DKConstant.DK_CM_DATAITEM_TYPE_ARRAY :
     return ("Array");
   case 19 :
     return ("Clob");
   default :
     return ("Unknown  (" + attr + ")");
  }
}
```

Now update the controller layer in order to use the defined methods.

Controller layer

For this specific use case, you may have noticed that the RBImportItem.jsp page is not sending the request to the RBController Servlet, instead the RBFileUpload Servlet is receiving the request directly.

The reason is basically that we need to process the request differently, since the data request definition has the Content type definition as: multipart/form-data, needed for the file upload.

Usually, a HTML form definition such as this can be used:

```
<form name=”simpleForm” method=”[POST | GET]” action=””>
<input type=”text” name=”attribute1” value=”valueOne”>
<input type=”text” name=”attribute2” value=”valueTwo”>
<input type=”submit” value=”Submit”>
</form>
```

It generates a string query that can be accessed, depending on whether the request was using the POST o GET methods, using several options, as follows:

```
QueryString = attribute1=valueOne&attribute2=valueTwo
```

In order to have the file upload, change the definition to this:

```
<form name=”fileUploadForm” method=”POST” ENCTYPE="multipart/form-data"
action=””>
<input type=”text” name=”attribute1” value=”valueOne”>
<input type=”text” name=”attribute2” value=”valueTwo”>
<input type=”file” name=”ContentFile”>
<input type=”submit” value=”Submit”>
</form>
```

Refer to this Web site for the following discussion:

http://www.ietf.org/rfc/rfc1867.txt

Our coding generates, as specified by protocol extension of RFC 1867, the following data:

```
Content-type: multipart/form-data, boundary=AaB03x
--AaB03x
content-disposition: form-data; name="attribute1"
valueOne
--AaB03x
content-disposition: form-data; name="attribute2"
valueTwo
--AaB03x
content-disposition: form-data; name="ContentFile"; filename="some.jpg"
Content-Type: image/jpg
... contents of some.jpg ...
--AaB03x--
```

So, in order to keep the code simple, we defined a specific servlet to handle this request. Once the task is done, the request is dispatched to the RBController servlet. In order to help our sample Web application coding, we used third party libraries to process the HTTP request that conform to RFC 1867. These libraries are part of the Jakarta Commons project, specifically, the command FileUpload package. Visit the following site for further information:

http://www.jakarta.apache.org/commons/fileupload/

Add the *commons-fileupload-1.0.jar* library to the Java Build Path within the WebProject. Create a new Servlet class called *RBFileUpload.java* in the *itso.rb.intranet.servle*t Java package. Use the Java code in Example 8-50 for this new servlet.

Example 8-50 RBFileUpload.java servlet definition

```
package itso.rb.intranet.servlet;

import java.io.IOException;

import javax.servlet.*;
import javax.servlet.http.*;
import java.util.*;
import com.ibm.mm.sdk.common.DKException;
import itso.rb.intranet.model.*;
import itso.rb.intranet.bean.*;
import org.apache.commons.fileupload.*;
/**
 * @version 1.0
 * @author
 *
 * RedBrook County System Sample
 * Handles the multipart request in order to import a file
 * to the Content Manager Repository.
 *
 */
public class RBFileUpload extends HttpServlet implements Servlet {
   public void doTask(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

      RBSessionBean sb = (RBSessionBean)
            req.getSession().getAttribute("SessionBean");
      Hashtable params = new Hashtable();

      byte[] mfile = null;
      String mimeType;

      // Upload File
      // Create a new file upload handler
```

```
DiskFileUpload upload = new DiskFileUpload();

try {

   // Parse the request
   List /* FileItem */ items = upload.parseRequest(req);

   // Process the uploaded items
   Iterator iter = items.iterator();
   while (iter.hasNext()) {
      FileItem item = (FileItem) iter.next();

      if (item.isFormField()) {
         params.put(item.getFieldName(),item.getString());
   } else {
         mfile = item.get();
      }
   }

   // Create ITEM and Item Part in CM
   // Get ICM Manager from session
   RBICMModel ICMMgr = (RBICMModel)
            req.getSession().getAttribute("ICMMgr");
   if( ICMMgr == null ) {
      ICMMgr = new RBICMModel();
      ICMMgr.setConnectParams(
         (String)req.getSession().getAttribute("CMSERVER"),
         (String)req.getSession().getAttribute("CMUSER"),
         (String)req.getSession().getAttribute("CMPASSWORD"));

      req.getSession().setAttribute("ICMMgr", ICMMgr);
   }

   // Create Item type and item part using the OO Java model
   ICMMgr.createItemFromUpload(params, mfile);

   // Response message for user
   sb.setGeneralMessage("Document added successfully");
   sb.setGeneralMessage(true);

   this.dispatch(req, resp, "RBController");

} catch (DKException e) {
      e.printStackTrace();
      sb.setErrorOccurred(true);
      sb.setErrorMessage(e.getMessage());
} catch (Exception e) {
      e.printStackTrace();
      sb.setErrorOccurred(true);
```

```
            sb.setErrorMessage("An error occured during item creation.");
        }
    }

    public void doGet(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
        this.doTask(req, resp);
    }

    public void doPost(HttpServletRequest req, HttpServletResponse resp)
        throws ServletException, IOException {
            this.doTask(req, resp);
    }

    /*****************************************************************
    * Dispatches to the next page
    * @param request The incoming request information
    * @param response The outgoing response information
    * @param nextPage The page to dispatch to
    */
    public void dispatch(HttpServletRequest request,
            HttpServletResponse response, String nextPage)
          throws ServletException, IOException {
        RequestDispatcher dispatch = request.getRequestDispatcher(nextPage);
        dispatch.forward(request, response);
    }

    public void init() throws ServletException {
        super.init();
    }
}
```

As you can review at the code, the Apache libraries are quite simple to use. If necessary, because of memory use and file size, tune the Java component according to the library documentation.

At this point we are able to import new documents and the application should be able to retrieve the documents. Actually, the RBEditFolder.jsp has already defined a link reference, pointing to an *ICMOpenResource.java* servlet, using the object's PID as parameter to retrieve the content.

ICMOpenResource.java

Create a new Servlet using the New Servlet Wizard called *ICMOpenResource.* The servlet receives a PID String as a parameter, retrieves the related content, defines the Content-type header for the http response using the content mime type definition, and pushes the content through the output stream to the client (Example 8-51).

Example 8-51   ICMOpenResource.java definition

```
package itso.rb.intranet.servlet;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

import com.ibm.mm.sdk.common.*;
import com.ibm.mm.sdk.server.*;
import java.io.*;
import java.lang.String.*;
import ICMConnectionPool;

import java.util.ResourceBundle;

/**
 * Retrieves an object from CM Resource manager.
 * Input:  PID request parameter
 * Output: set content type header using stored object mime type
 * and push the content. Image content (gif/jpeg) is retrived creating
 * a small dynamic html page and setting the image url
 *
 * The connection parameters are taken from Config.properties
 * s
 */
public class ICMOpenResource extends HttpServlet implements DKConstantICM {
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }

  public synchronized void doPost(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }
  public synchronized void doGet(
    HttpServletRequest request,
    HttpServletResponse response)
    throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();
    try {

      HttpSession session = request.getSession();

     ResourceBundle Config = ResourceBundle.getBundle("Config");
     DKDatastoreICM dsICM = ICMConnectionPool.getConnection(
                (String) Config.getObject("ICMServer"),
                (String) Config.getObject("ICMUser"),
                (String) Config.getObject("ICMPasswd"));

      String inPidString = request.getParameter("PID");

      /// Create a DKLobICM (lob) with the passed in pidstring (inPidString)
      DKLobICM lob = (DKLobICM) dsICM.createDDO(inPidString);

      // Retrieve just enough information to find out its mimetype
      lob.retrieve(DKConstant.DK_CM_CONTENT_NO);

      // Set the string (mimeType) to the mime type of (lob)
      String mimeType = lob.getMimeType();

    // we are retrieving non-gif or jpeg image content
      response.setContentType(mimeType);

        lob.retrieve(DKConstant.DK_CM_CONTENT_ONLY);

        // Setup the storage to hold the object by allocating its length in bytes
        byte[] content = new byte[(int) lob.length()];

        // Put the content of the blob into the storage you just allocated
        content = lob.getContent();

        // write the content out using the mime type
        out.write(content);

      // Return the dsICM Connection back to the ICMConnectionPool
      ICMConnectionPool.returnConnection(dsICM);

    } //end try
    //-------------------------------------------------------
    catch (DKException exc) {
     out.println(exc.name() + "<br>" + exc.getMessage());
     System.out.println(exc.name() + "\n" + exc.getMessage());
      ICMConnectionPool.clearConnections();

      exc.printStackTrace();
    } catch (Exception exc) {
      out.println("Exception message " + exc.getMessage());
      System.out.println(exc.getMessage());
        exc.printStackTrace();
    }
  }
}
```

Tip: As an optional exercise, review the *DKLobsICM.getContentURLs* method at the APIs documentation and design an alternate retrieve content strategy having in mind the IBM Content Manager's flexibility to define distributed solutions.

View layer

The *RBImportItem.jsp* file allows us to input the metadata and select the file to be imported from the IBM Content Manager repository. In order to keep the code simple, the JSP file is making use of the RBICMModel class directly with the objective of retrieving the Item Type definition.

Create a new JSP File called RBImportItem.jsp using the code sample in Example 8-52.

Example 8-52 RBImportItem.jso JSP

```
<%@ page import="com.ibm.mm.sdk.server.*,
         com.ibm.mm.sdk.common.*,itso.rb.intranet.model.*"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
   type="text/css">
<TITLE>RBImportItem.jsp</TITLE>
</HEAD>
<BODY>

   <% // Get Model Bean from Session
      RBICMModel ICMMgr = (RBICMModel) session.getAttribute("ICMMgr");
      if( ICMMgr == null ) {
         ICMMgr = new RBICMModel();
         ICMMgr.setConnectParams(
            (String)session.getAttribute("CMSERVER"),
            (String)session.getAttribute("CMUSER"),
            (String)session.getAttribute("CMPASSWORD"));
         session.setAttribute("ICMMgr", ICMMgr);
      }
      ICMMgr.connect();
      DKDatastoreICM dsICM = ICMMgr.getDatastore();


      // Get ItemType to be imported/created in order to
      // list atributes required dynamically
      String strEntity = request.getParameter("ItemType");
      // Create a DKDatastoreDefICM object (dsDefICM) so we can get info about the
      // chosen item type (strEntity) to create the ddo in.
      DKDatastoreDefICM dsDefICM = new DKDatastoreDefICM(dsICM);
      DKItemTypeDefICM entityDef =
        (DKItemTypeDefICM) dsDefICM.retrieveEntity(strEntity);

    %>

<h1>Import a document to the folder application</h1>
<hr>
<P><BR>
</P>
<P>Enter attribute values to the specified document type </P>
<FORM name="uploadForm" action="RBFileUpload"  method="POST"
         enctype="multipart/form-data">
<input type="hidden" name="iType" value="Document">
<INPUT TYPE="hidden" NAME="cmd" VALUE="RBICM">
<INPUT TYPE="hidden" NAME="subcmd" VALUE="uploadFile">
<INPUT TYPE="hidden" NAME="nextPage" VALUE="RBIndex.jsp">
<INPUT TYPE="hidden" NAME="ItemType"
   VALUE="<%=request.getParameter("ItemType")%>">
<INPUT TYPE="hidden" NAME="WorkList"
   VALUE="<%=request.getParameter("WorkList")%>">
<INPUT TYPE="hidden" NAME="PID"
   VALUE="<%=request.getParameter("PID")%>">
<INPUT TYPE="hidden" NAME="wpPID"
   VALUE="<%=request.getParameter("wpPID")%>">
<INPUT TYPE="hidden" NAME="uix" VALUE="<%=request.getParameter("uix")%>">

<TABLE border=1 width=70% align="center">
    <TR><TH>Attribute</TH><TH>Type</TH><TH>Value</TH></TR>
    <%
      DKSequentialCollection allAttrs =
        (DKSequentialCollection) entityDef.listAllAttributes();
      dkIterator pIter2 = allAttrs.createIterator();
      while (pIter2.more()) {

        DKAttrDefICM aDef = (DKAttrDefICM) pIter2.next();
     %>

          <TR>
            <TH>
```

```
               <%=aDef.getDescription()%>
            </TH>
            <TH>
               <%=ICMMgr.getAttrType(aDef.getType()) %>
            </TH>
            <TD>
               <%if ( aDef.getName().equals("BP_NUMBER") ) { %>
                 <input type="hidden" name="BP_NUMBER"
          value="<%=request.getParameter("BP_NUMBER") %>">
                 <%=request.getParameter("BP_NUMBER") %>
               <% } else { %>
                 <input type="text" size="40" name="<%=aDef.getName()%>">
               <% } %>
             </TD>
           </TR>
    <% } %>
    <TR><TH colspan="3">Date format: yyyy-mm-dd</TH></TR>
    <TR>
       <TH>File: </TH>
       <TD colspan="2" align="right"><input type="file" name="cmfile" value=""></TD>
    </TR>
    <TR><TH>Mime Type: </TH>
        <TD colspan="2">
          <Select name="mimeType">
      <%
         // Get datastore mimetypes manager
         DKMimeTypeMgmtICM MTMgr = new DKMimeTypeMgmtICM(dsICM);

        dkCollection mimeTypesCol = MTMgr.listMimeTypes();
        dkIterator mimeTypes = mimeTypesCol.createIterator();
        for(int i=0; mimeTypes.more() ; i++ ) {
           DKMimeTypeDefICM mime = (DKMimeTypeDefICM) mimeTypes.next();
          %>
        <option SELECTED value="<%=mime.getMimeType()%>">
          <%=mime.getDisplayName()%>
        </option>
      <% } %>
          </Select>
        </TD>
    </TR>
    <TR><TH colspan="3" align="right"><input type="submit" value="Import"></TH></TR>
</TABLE>
<br><br>
</FORM>
</BODY>
</HTML>
<% ICMMgr.disconnect(); %>
```

Open the *RBEditFolder.jsp* file, locate the *Java Script section* at the beginning of the file, and uncomment the submit line at the *setItemType()* Java Script function.

Save the file and close the editor.

Test

Update the WebSphere Testing environment. Add *commons-fileupload-1.0.jar* to the class path definition.

Restart your WebSphere Testing server in order to refresh the configuration descriptor file, so the new servlets and libraries became valid.

8.6 External application

The external application or Internet application implements fewer use cases than the internal application. It is based on the same MVC model framework as the internal application, but the business requirements respond to the property owner's needs.

In the next sections, we briefly describe the use cases that need to be implemented and the files we developed for the sample application. We packaged the sample application in a WAR file and included the instruction to import the WAR file and configure your workbench.

To learn and get the most benefit out of this section, we recommend that you read through the section, install the WAR file onto your environment, configure your system, test the sample application, and then, examine each piece of the code developed for the sample application.

8.6.1 Internet sample application use cases

The Internet sample application consists of the following use cases:

- Query building permit application status use case
- Research on code book use case
- Review properties payment history use case
- Get previous land maps use case

Let us examine each use case briefly.

Query building permit application status use case

Table 8-10 describes the Query building permit application status use case.

Table 8-10 Query building permit application status use case description

Element	Description
Name	Query building permit application status
Description	Display the building permit approval process status to the property owner.
Actors	Property owner
Inputs	Social security number
Pre-conditions	The property owner must be logged at the system and have a valid SSN
Steps	1. The property owner logs in to the Internet application. 2. Selects "List Building permit application status" from the menu. 3. The system displays the related building permit application folders. 4. It opens the BP Folder detail.
Alternatives	
Post-conditions	

To simplify the implementation of the Internet sample application, we designed the application such that it requires a valid SSN in order to for the user to login. There is no password required. The SSN number is used to retrieved all the related folders contained within the BP_FOLDER item type.

Note: To develop your own Internet application, assuming that you do not use LDAP, you should implement the code to provide a default password for the user, request the user to login with the password, and after the user logs in for the first time, force the user to change the password.

Example 8-53 shows a code snippet of the method contained at the ICM Model Java bean that implements the search.

Example 8-53 getBPFolderBySSN method - Code snippet

```
public Vector getBPFolderBySSN(String pSSN) throws DKException, Exception {

   Vector resultTable = new Vector();
   if( dsICM == null)
      this.connect();

   DKResults results = null;
   String queryString = "/BP_FOLDER[@BP_TAX_ID = \""+
                        pSSN + "\"] SORTBY (@BP_TAX_ID ASCENDING)";
   System.out.println(queryString);

   // set up the DKNVPair parns (parms[]} to set a max return value of 50
    DKNVPair parms[] = new DKNVPair[3];
    parms[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "50");
    parms[1] =
      new DKNVPair(
       DKConstant.DK_CM_PARM_RETRIEVE,
       new Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
    // Specify any Retrieval Options desired.  Default is ATTRONLY.
    parms[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);

   // Evaluate the query, seting the results into (results)
   results = (DKResults) dsICM.evaluate(queryString, DKConstant.DK_CM_XQPE_QL_TYPE, parms);

   System.out.println(results.cardinality() + " items found. processing results");
   // put an iterator (iter) on the results collection
   dkIterator iter = results.createIterator();

   System.out.println(results.cardinality() + " items found");

   DKDDO ddo = null;
   while (iter.more()) {
     ddo = (DKDDO) iter.next();
     if (ddo != null) {
      resultTable.add(ddo);
     }
   }

   this.disconnect();
   return resultTable;
}
```

For more information about the parametric search, refer to the *Content Manager Workstation Application Programming Guide*, SC27-1347.

Research on code book use case

Table 8-11 describes the Research code book use case.

Table 8-11 Research code book use case description

Element	Description
Name	Research on code book.
Description	Content search within the code books documents.
Actors	Property owner.
Inputs	Search criteria.
Pre-conditions	
Steps	1. The property owner selects "Research on code books" from the main menu. 2. The system displays the content search input form. 3. The system lists the code books containing the search criteria defined by the user.
Alternatives	
Post-conditions	

Refer to Chapter 9, "Adding text search capability" on page 321 for instructions on how to implemented the text search sample code.

Review properties payment history use case

Table 8-12 describes the Review properties payment history use case.

Table 8-12 Review properties payment history use case description

Element	Description
Name	Review properties payment history.
Description	Display documents related to the payment history.
Actors	Property owner, IBM CM OnDemand.
Inputs	Property number.
Pre-conditions	

Element	Description
Steps	1. The property owner logs in to the Internet application. 2. The property owner selects “Review property payment history” from the menu. 3. The system displays the related property numbers. 4. The user selects a property number and requests the related documents. 5. The system retrieves the related documents from the IBM Content Manager OnDemand repository.
Alternatives	
Post-conditions	

In order to have this sample code working, you must set up sample data for your OnDemand repository. Refer to Appendix A, “Setting up case study infrastructure” on page 487 for details on how to set it up.

Get previous land maps use case

Table 8-13 contains the detailed use case description for Get previous land maps.

Table 8-13 Get previous land map use case description

Element	Description
Name	Search previous land maps.
Description	Search land maps using the BP_Plot item type attributes, retrieving maps only from approved building permit folders.
Actors	Property owner, Content Manager.
Inputs	Search criteria.
Pre-conditions	
Steps	1. The property owner selects “Get previous land maps” from the main menu. 2. The system displays the content search input form. 3. The system lists the available land maps matching the search criteria defined by the user.
Alternatives	
Post-conditions	

We leave this use case as an optional exercise for the reader.

8.6.2 Framework file descriptions

The framework is organized just like the internal sample application with different sets of business logics. We developed the following applications that are included in the WAR file:

- **ICMConnectionPool.java:** Manages connection pool requests.
- **ICMOpenResource.java:** Retrieves a file content within an http request sending the content mime type as header in order to be received by a browser or any requester application.
- **RBDB2Model.java:** Model bean to retrieve data from the RedBrook DB2 legacy system using the Information Integrator for Content APIs.
- **RBICMModel.java:** Model bean to retrieve data from Content Manager using the Information Integrator for Content APIs.
- **RBODModel.java:** Model bean to retrieve data from Content Manager OnDemand system using the Information Integrator for Content APIs.
- **RBOwnerBean.java:** Helper data Java bean.
- **RBSessionBean.java:** Helper bean used to save data within the http session object.
- **RBController.java:** Main controller servlet. Initializes application parameters and dispatches commands to specialized servlets.
- **RBICMServlet.java:** Specialized controller servlet for requests regarding the Content Manager system.
- **RBLogin.java:** Specialized controller servlet.
- **RBODServlet.java:** Specialized controller servlet for requests regarding the Content Manager OnDemand system.
- **RBIndex.jsp:** Main page.
- **RBLogin.jsp:** Login page.
- **RBLogout.jsp:** Logout page.
- **RBCheckStatus.jsp:** Displays Building permit application status.
- **RBODSample.jsp:** Renders OnDemand query results.
- **RBShowMessages.jsp:** Renders error and system messages.

For more information about the framework structure, please refer to 8.2, “Application framework description” on page 209.

8.6.3 Import Internet sample application WAR file

Locate the file *InternetApp.war* from the additional material folder and import the WAR file:

1. With in IBM WebSphere Studio Application Developer, select **File** → **Import** from the menu.
2. From the Select import resource windows, select **WAR File** and click **Next**.
3. Set the absolute file path at the *WAR file* field.
4. Create a new project, click **New**.
5. Set the project name to *InternetApp* and click **Finish**.
6. If a Server configuration window pops up, accept the changes clicking **OK**.
7. Click **Finish**.
8. Update the Java Build path of the new project by adding the following libraries as explained before.

   ```
   cmb81.jar
   cmbcm81.jar
   cmbjdbc81.jar
   cmblog4j81.jar
   cmbsdk81.jar
   cmbutil81.jar
   cmbutilfed81.jar
   cmbutilicm81.jar
   cmbview81.jar
   xalan.jar
   xerces.jar
   ```

9. Verify that no compilation error is shown; otherwise, verify your workbench environment.

We strongly recommend that you go through each file to see what it is doing, then run and test the application to see how it works. Afterwards, make some modifications to see how your changes affect the application.

Important: Please use the provided sample application code *as is*. It is not the responsibility of the IBM Content Manager Technical Support Team and the IBM Redbook organization to provide any type of support for the sample code we provided. If you find any errors in the code or in this IBM Redbook that need correction, please direct your questions to the IBM Redbook organization. We will try our best to correct any mistakes.

8.6.4 Testing

To test, under the Web-Content of the Internet application, right-click **RBIndex.jsp** and select **Run on server** from the context menu. Refer to Appendix A, “Setting up case study infrastructure” on page 487 for descriptions of a sample application page and navigation flow.

8.7 Conclusion

In this chapter, we used Information Integrator for Content non-visual Java beans and Java OO APIs and created a custom Content Manager system. We divided the custom Content Manager system into an internal intranet application and an external Internet application.

We started each section with a use case description, a design, and sample codes that are used to implement the use case. Although we did not show in detail, step-by-step, why we implemented the codes this way, we hope that by following our examples, you have a better understanding of how to approach and develop a Web application like those described. We also encouraged you to deploy the sample applications provided with this redbook, experiment with them, and learn from them.

As you develop your application, refer to the on-line Information Center for Content Manager or Information Integrator for Content to get detailed descriptions for any of the classes you need.

9

Adding text search capability

This chapter focuses on the text search set up of our business application presented in Chapter 8, “Building a case study application” on page 207. We use text search to retrieve the code books contents. This module is part of the external application and requires you to set up the Content Manager environment in order to make its content text searchable.

We cover the following topics:

- Introduction
- Start Content Manager text search service
- System setup
- Integrating the text search module with the Web application

9.1 Introduction

Text search in Content Manager is performed using the DB2 Net Information Extender (NIE), formerly known as Text Information Extender, which was used by Content Manager V8.1. Content Manager allows two types of text search:

- Search against attributes that contain text in components
- Text search against object content.

In each case, content is stored differently. In the first case, the attribute is defined to be text searchable. NIE creates indexes to make the search efficient. This enables text search to be performed for all of its child components down the hierarchy. In the second case, however, when the search is required to be performed against an object's content, NIE uses a reference to the resource object, and uses the reference when creating its indexes.

Additional setup steps are required to make a resource item of document model type text searchable; this is explained in the later sections.

From the client point of view, a setup for the text resource item type view is required in order for the search mechanism to locate the content in the Resource Manager.

9.1.1 Text search syntax

There are two text searches: basic and advanced. In the *basic text search*, the syntax allows many search options, as follows:

- Case insensitive search.
- Phrase search (enclose an entire phrase within quotes).
- The (+) sign, specifying that the term must be included in the document
- The (-) sign, specifying that the term must not be included in the document
- Use of wildcard:
  - The (?) sign is used to replace a single character.
  - The (*) sign is used to replace any number of characters.

In the *advanced text search*, the DB2 Net Search Extender (NSE) text search syntax is used. This allows for more powerful text searches such as proximity search and fuzzy search.

The syntax of both searches are beyond the scope of this redbook. We present only few search examples to give you an idea of the search syntax. To learn more about text search syntax, refer to:

- Chapter 7, "Query language" in *IBM DB2 Content Manager Implementation and Migration Cookbook*, SG24-7051
- *IBM Content Manager for Multiplatforms/IBM Information Integrator for Content, Workstation Application Programming Guide,* C2713471

Let us take a look of some search examples:

- Advanced search:

  The following example shows an advanced search query string:

  ```
  /TextResource[contains-text(@TIEREF," 'Java '&'XML ' ")=1 ]
  ```

  This example performs a text search inside of the documents stored in the Resource Manager. The item type should replace the word TextResource. Inside the contains-text function, the NIE syntax is used.

- Direct search in advanced search:

  As for the resource of type document model, in our example, such as in the sample application, a special precision should be added to the syntax: The keyword `ICMPARTS` should be added to the search term:

  ```
  /Doc [contains-text(.//ICMPARTS/@TIEREF,"'XML '")=1 ]
  ```

  As mentioned earlier, Content Manager has two search functions: `contains-text-basic` is a Content Manager search function, while `contains-text` uses the NSE syntax. This example translates to a direct search.

- Linguistic search in basic search:

  If a fuzzy search is required, the term `contains-text-basic` should be used and does the linguistic matches:

  ```
  /Doc[contains-text-basic(.//ICMPARTS/@TIEREF, " 'XML' ")=1]
  ```

  Note that the Windows client always does a basic search, similar to a linguistic search.

- Linguistic search in advanced search:

  To get the same behavior (linguistic search) in advanced search, use `contains-text`:

  ```
  /Doc[contains-text(.//ICMPARTS/@TIEREF, " FUZZY FORM OF 70 'XML' ")=1]
  ```

  or

  ```
  /Doc[contains-text(.//ICMPARTS/@TIEREF, " STEMMED FORM OF 'XML' ")=1]
  ```

  When you use `contains-text`, you can pass in any valid NSE search syntax. The problem with doing this, however, is that it will only work with DB2. It is better to use basic search, as it works on different databases and text engines.

9.2 Start Content Manager text search service

If you are using DB2 Version 8, DB2 Net Information Extender must be installed for the text search to be enabled in Content Manager. If you are using DB2 Version 7, you should install the Text Information Extender TIE instead.

The installation instruction and the Content Manager product information contains information to make you start the NIE. This requires two steps:

1. Start the service.
2. Enable the Library Service for text search.

If db2text service is not already an automatic service, as shown in Windows services panel, manually start the service from a command line:

```
db2text start
```

You should get an operation completed successfully message or a message indicating the services are already active.

When you install Content Manager, you have an option to enable text search. If you did not enable that, run the following command:

```
db2text enable database for text connect to icmnlsdb
```

Substitute icmnlsdb with the Library Server database name in your environment if you do use icmnlsdb.

If the current log in user does not have the administration authority, re-run the command with Library Server database administration ID and password information:

```
db2text enable database for text connect to icmnlsdb user icmadmin using
lspasswd
```

If you do not remember whether you enabled the text search in Content Manager, running the above command will inform you either that the database is enabled for text search or that it is already enabled.

If you run into any problem during the start of the service or text enabling of the database, refer to the Readme file in the Content Manager application group by selecting **Start** → **Programs** → **IBM Content Manager for Multiplatforms V8.2** → **Readme**. Search for “DB2 NetSearch Extender V8 known problems.”

9.3 System setup

In 2.2, “Data modeling” on page 29, we cover the basic Content Manager data model, including the concept of attributes, item types, and items. In 6.3, “Developing sample code with the Java OOAPIs” on page 110, our sample code covers the data model creation, which includes attribute creation, item type creation, and importing items.

In this chapter, we complete the data model with what is needed for text search. We set the Content Manager system for text search with Content Manager System Administration Client.

In our Redbrook County building permit application, it is required to make the content of the building permit code books available for text search. This means that the Web user can enter a search term or code and be able to retrieve the code book that contains the searched term. As in practical life, the code books may contain a lot of information that cannot be indexed using user-defined attributes. A text search within the content will be helpful in this case.

We need to create an item type to store the text files of any type (PDF, TXT, or DOC) and to create attributes to index these objects for a parametric search.

Creating the attribute

Using Content Manager Administration Client, create the attributes as shown in Table 9-1.

Table 9-1 Attributes associated with code book

Attribute	Attribute type	Max length
BP_CODEBOOK_ID	VARCHAR	12
BP_CODEBOOK_SUB	VARCHAR	200

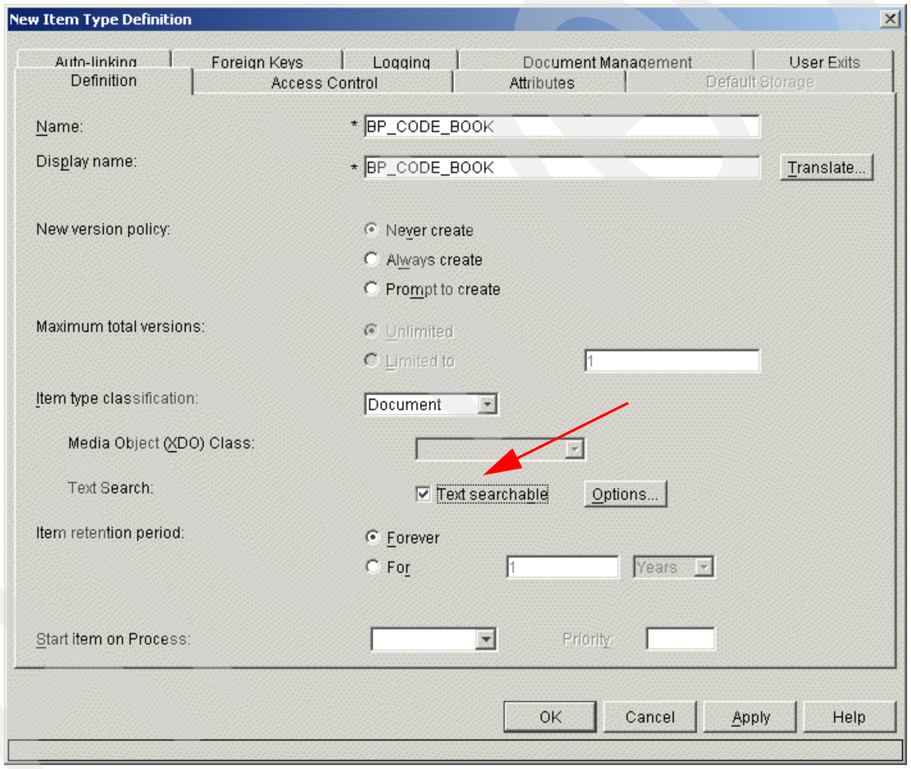
If you run the sample code, CMaddAttr.jsp, presented in Chapter 6, “Quick start in Web application development” on page 95, you should have these attributes already and can thus skip this step.

Creating the item type

Using Content Manager System Administration Client, create a document item type BP_CODE_BOOK. This item type holds parts that are required to be text searchable.

These are the detailed steps for creating the item type:

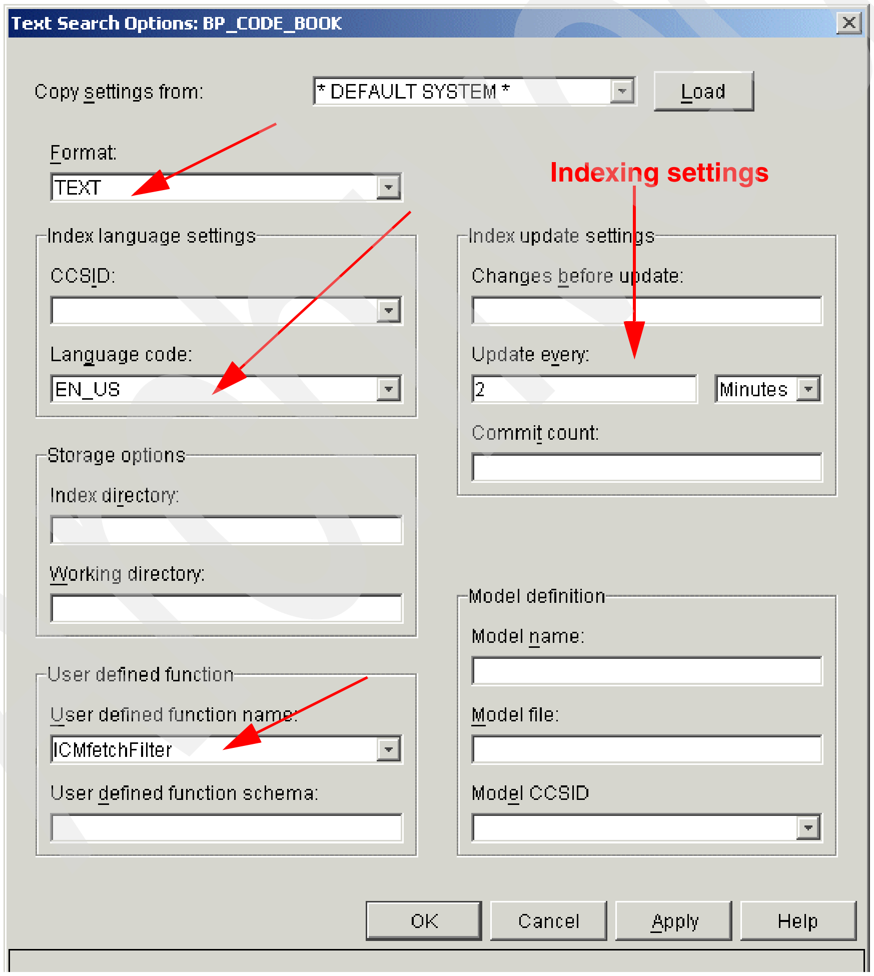
1. Start Content Manager System Administration Client by selecting **Start** → **Programs** → **IBM Content Manager for Multiplatforms V8.2** → **System Administration Client**. Enter your Library Server user ID and password to log on to the System Administration Client's main window.
2. From the left hand side window, expand the Library Server database (default is **icmnlsdb**) → **Data Modeling** → **Item Types**. Right-click **Item Types** and select **New** from its context menu.
3. In the new item type dialog, select **Definition** tab, enter BP_CODE_BOOK in the name field (see Figure 9-1). Select **Text searchable** check box, and click **Options**.



Name: * BP_CODE_BOOK
Display name: * BP_CODE_BOOK
New version policy: [✓ Never create] Always create Prompt to create
Maximum total versions: [✓ Unlimited] Limited to 1
Item type classification: Document
Media Object (XDO) Class:
Text Search: [✓ Text searchable]
Item retention period: [✓ Forever] For 1 Years
Start item on Process:
Priority:

Figure 9-1 Item type definition: name and text search options

4. In the options window (see Figure 9-2), make sure you have specified at least the following four settings:
   - In the Format field, select **TEXT** from the drop-down box.
   - For the Index Languages settings, in the Language code field, select **US_EN** from the drop-down box.
   - For the User Defined Function, in the User defined function name, select **ICMfetchFilter** from the drop-down box.
   - For the index update settings, in the Update every field, enter 2 and select **Minutes** instead of hours from the drop-down box.
5. Click **Apply** to save your settings.



Text Search Options: BP_CODE_BOOK

Copy settings from: * DEFAULT SYSTEM *

Format: TEXT

Index language settings
CCSID:
Language code: EN_US

Storage options
Index directory:
Working directory:

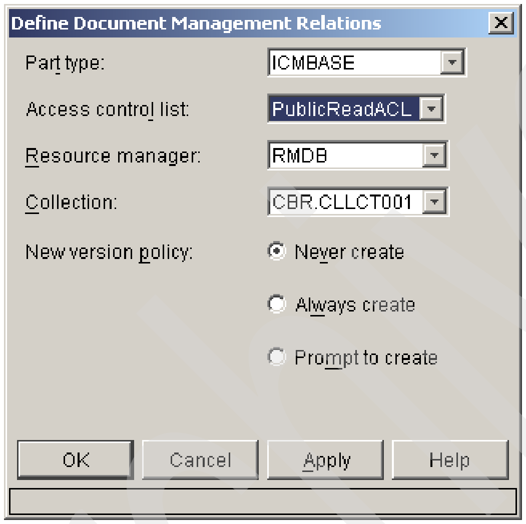
User defined function
User defined function name: ICMfetchFilter
User defined function schema:

Index update settings
Changes before update:
Update every: 2 Minutes
Commit count:

Model definition
Model name:
Model file:
Model CCSID

Figure 9-2 Setting the text search options for an item type

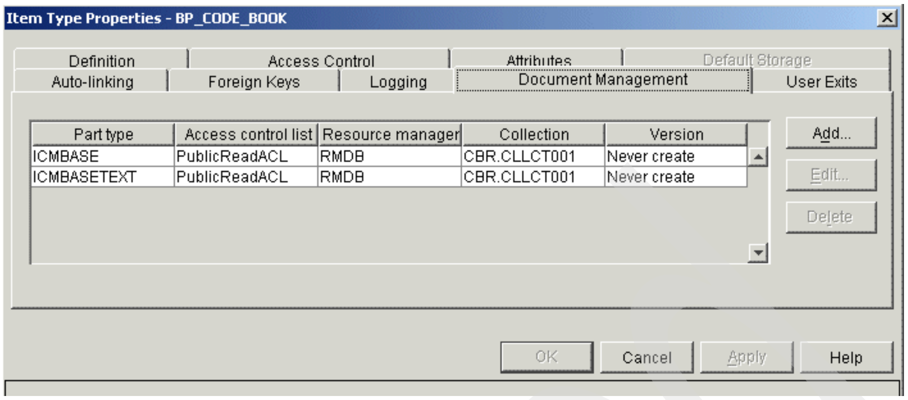
6. Click the **Attributes** tab, select **BP_CODEBOOK_ID** and **BP_CODEBOOK_SUB** from the left panel and click **Add** to add these attributes to the right panel.
7. Select the **Document Management** tab, and add two parts:
   a. Click **Add** to add a part. From the part type drop-down box, select **ICMBASE**, change the Access Control list from the default to **PublicReadACL** (see Figure 9-3), and click **Apply** to save your settings.
   b. Click **Add** again, select **ICMBASETEXT** for Part type, **PublicReadACL** for Access control list, and click **Apply**.



Part type: ICMBASE
Access control list: PublicReadACL
Resource manager: RMDB
Collection: CBR.CLLCT001
New version policy: [✓ Never create] Always create Prompt to create

Figure 9-3 Create item type - Add part in Document Management tab

There should be two parts in the Document Management tab as shown in Figure 9-4.



Part type	Access control list	Resource manager	Collection	Version
ICMBASE	PublicReadACL	RMDB	CBR.CLLCT001	Never create
ICMBASETEXT	PublicReadACL	RMDB	CBR.CLLCT001	Never create

Figure 9-4 Create an item type with 2 parts

8. Click **OK** of the New Item Type Definition window. Click the refresh icon to see the new item type that is created for you.

Importing text resources

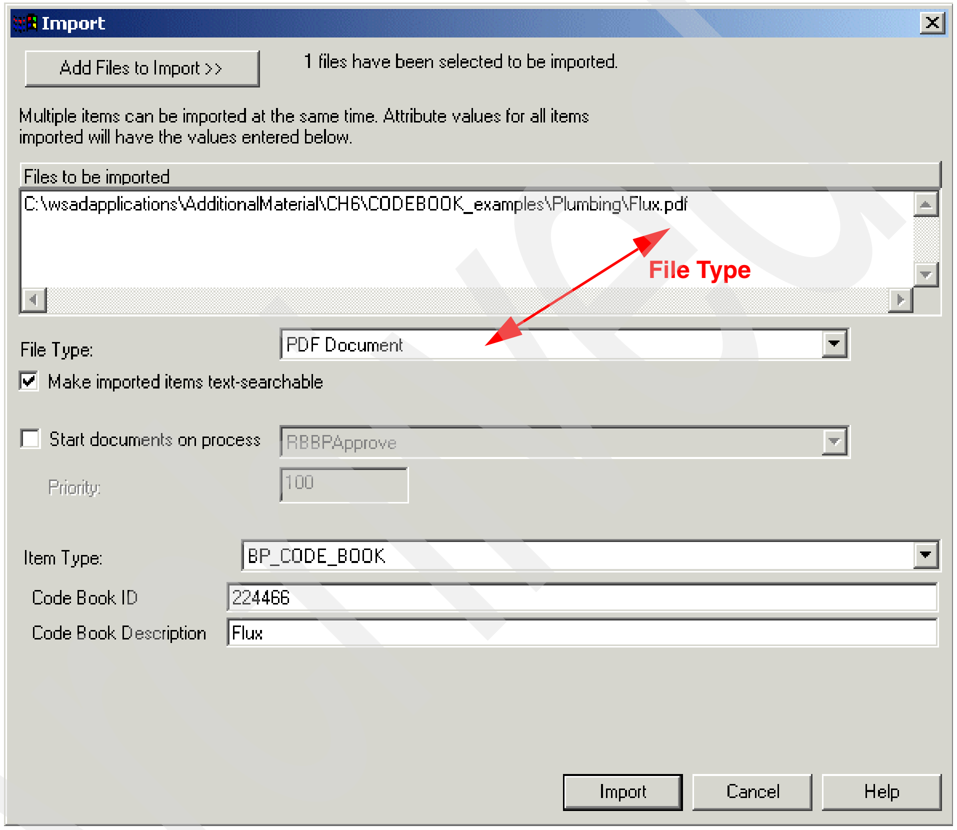
To set up the code book, we need to import the text resources using the Content Manager Windows Client.

To import, do the following steps:

1. Start the Content Manager Windows Client by selection **Start** → **Programs** → **IBM Content Manager V8** → **Client for Windows**. Enter the Library Server user ID and password.
2. Select **Import**.
3. In the import dialog, click **Add Files to Import**. Browse to the location where the additional material provided by this redbook is downloaded. Browse to the folder CH, open the CODEBOOK_examples folder, and select the building folder. Select the building.doc, and click **Open**. The file shows up in the **Files to be imported** field. Make sure to do the following steps:
   – In the File Type field, select **Microsoft Word** from the drop-down box.
   – For the Item Type field, select **BP_CODE_BOOK** that you have created.
   – Check the **Make imported items text-searchable** check box.
   – Fill the fields “Code Book ID” and “Code Book Description” with the values “111111” and “Building” respectively.
4. Click **Import**. You should be able to see the progress bar. Repeat the operation for the other files under the two other folders: Mechanical and Plumbing. Select any 6-digit string for the value of the “Code Book ID” and you can use the filename as a value for the “Code Book Description” field.

Attention: Make sure you select the proper file type when you import the documents. For example, when importing PDF files, remember to select **PDF Document** as File Type (see Figure 9-5).

5. Click **Cancel** when you have imported all the files.


Import

Add Files to Import >>

1 files have been selected to be imported.

Multiple items can be imported at the same time. Attribute values for all items imported will have the values entered below.

Files to be imported

C:\wsadapplications\AdditionalMaterial\CH6\CODEBOOK_examples\Plumbing\Flux.pdf

File Type: PDF Document

[✓] Make imported items text-searchable

[ ] Start documents on process RBBPApprove

Priority: 100

Item Type: BP_CODE_BOOK

Code Book ID 224466

Code Book Description Flux

Figure 9-5 Importing a PDF file

Testing the text search from the Windows Client

To testing the imported text resource, do these steps:

1. Click on the search icon in the Windows Client (see Figure 9-6).

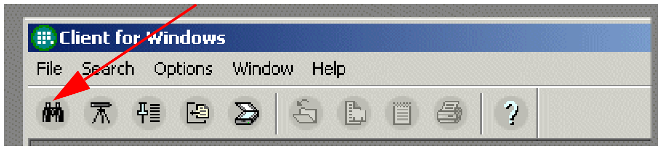
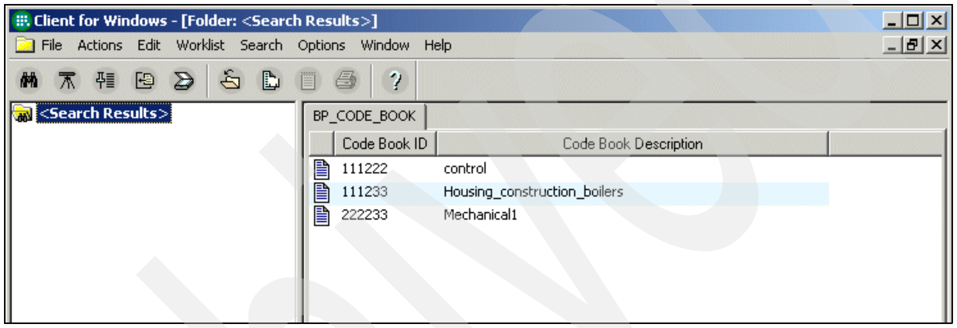


Figure 9-6 Search from the Content Manager Windows Client

2. In the Basic Search dialog, select BP_CODE_BOOK as item type, enter `code` in the Document contents field, and click **OK**.

You should retrieve the three documents listed in Figure 9-7.


Client for Windows - [Folder: <Search Results>]
File Actions Edit Worklist Search Options Window Help
<Search Results>
BP_CODE_BOOK

Code Book ID	Code Book Description
111222	control
111233	Housing_construction_boilers
222233	Mechanical1

Figure 9-7 Text search results

9.4 Integrating the text search module with the Web application

The Redbrook County building permit Web application framework is explained in Chapter 8, “Building a case study application” on page 207. In this section we integrate the text search module to the existing framework.

The Redbrook County building permit Web application consists of two projects, one for the intranet application and one for the external Internet application. We implement the text search module into the external Internet application, RedBrookWeb project.

To integrate, we include all the text search Web resources into the folder WebContent. The start point in the text search JSP file named SearchCodeBook.jsp.

The text search module consists of the files and resources illustrated in Figure 9-8.

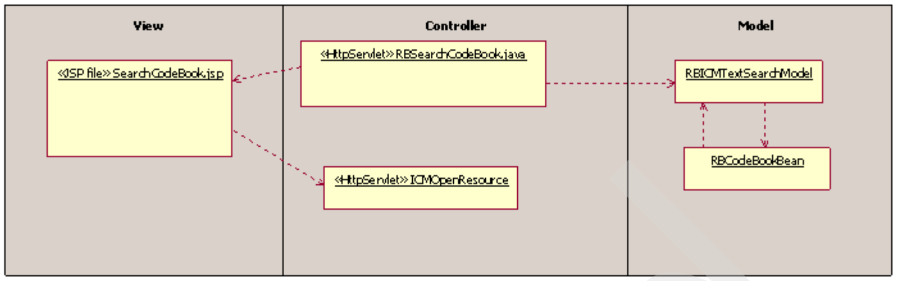

View
Controller
Model
«JSP file» SearchCodeBook.jsp
«HttpServlet» RBSearchCodeBook.java
«HttpServlet» ICMOpenResource
RBICMTextSearchModel
RBCodeBookBean

Figure 9-8   Text search module

These files are:

- ICMOpenResource.java (under JavaSource, with no default package)
- RBSearchCodeBook.java (under JavaSource, itso.redbrook.servlet)
- RBICMTextSearchModel.java (under JavaSource, itso.rb.external.model)
- RBCodeBookBean.java (under JavaSource, itso.redbrook.bean)
- SearchCodeBook.jsp (under WebContent)

You can import all the files in the beginning before continuing with the remainder of the chapter, or you can create them along the way.

SearchCodeBook.jsp

SearchCodeBook.jsp displays a text search form, allowing the user to perform a text search using a word or a sentence and a simple join of AND/OR. This means that a user can search for the occurrence of a word within the contents of a document, and/or the occurrence of a sentence.

Example 9-1 shows a code snippet of the SearchCodeBook.jsp.

Example 9-1   SearchCodeBook.jsp - Code snippet that adds a text search form

```
<Form name="SearchForm" action="../RBSearchCodeBook" method="POST">
<H6>Show Code Books that contains: </H6>
<TR>
<TD>the word</TD>
<TD><INPUT type="text" name="word_value" size="20">  </TD>
</TR>
<TR>
<TD align="left" colspan="2">
<input type="radio" name="join_type" value="AND"> AND
<input type="radio" name="join_type" checked value="OR"> OR</TD>
</TR>
<TR>
<TD>the sentence</TD>
<TD><INPUT type="text" name="sentence_value" size="60"></TD>
```

```
</TR>
<TR>
<TD colspan="2">
<INPUT type="submit" name="search" value="Search">
<INPUT type="hidden" name="nextPage" value="SearchCodeBook.jsp">
<INPUT TYPE="hidden" name="cmd" value="CodeBook">
</TD>
</TR>
</Form>
```

The form contains two text fields for the search, in addition to the **Submit** button:

- word_value
- sentence_value

In addition to a radio button field, there is a field to determine the type of join required:

- join_type

Two more hidden fields are added to ensure the application flow within the framework described in (ref). They are:

- cmd
- nextPage

The form calls the RBSearchCodeBook servlet. All the servlets are located in the package itso.redbrook.servlet. Refer to Chapter 8, "Building a case study application" on page 207 for the description of the application architecture.

Our application consists of three main packages:

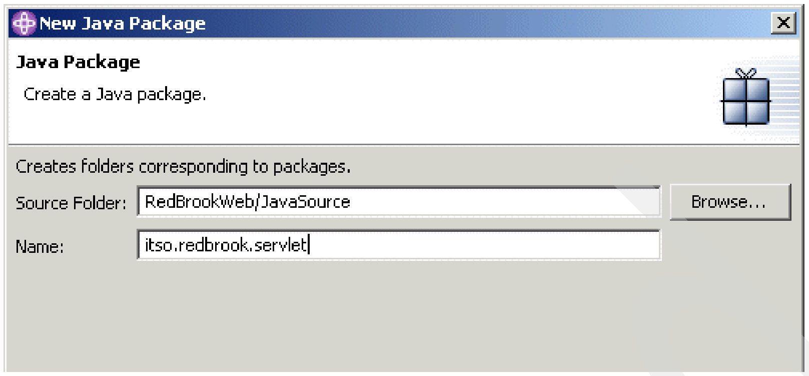
- itso.redbrook.servlet — Contains all the servlets of our application
- itso.redbrook.bean — Contains all the data beans
- itso.rb.external.model — Contains all the classes that implement the model

Adding packages

You can import the three packages downloaded from the additional material provided by this redbook into your project, or create the packages in your Web project.

To create your packages:

1. Select the JavaSource folder under the project.
2. From the context menu, select **New** → **Package**.
3. Enter itso.redbrook.servlet in the Name field and click **Finish**. See Figure 9-9.
4. Repeat the above steps for itso.redbrook.bean and itso.rb.external.model.


New Java Package

Java Package

Create a Java package.

Creates folders corresponding to packages.

Source Folder: RedBrookWeb/JavaSource

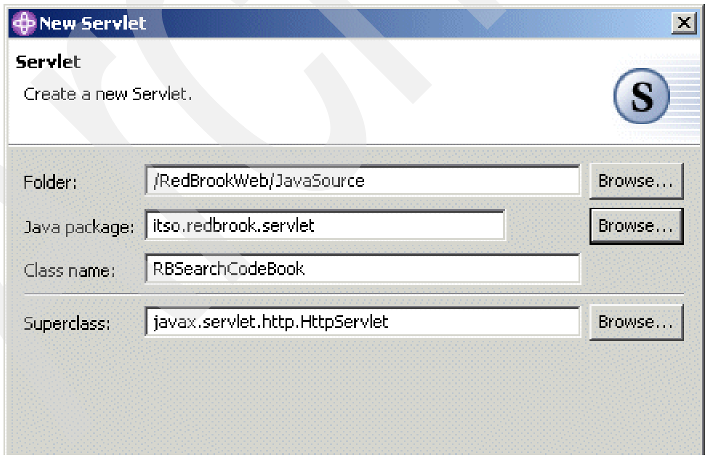
Name: itso.redbrook.servlet

Figure 9-9 Create the itso.redbrook.servlet package

Adding servlets

To add the servlets:

1. Select the itso.redbrook.servlet.
2. From the context menu, select **New** → **Servlet**.
3. Enter `RBSearchCodeBook` in the Class name field. Click **Browse** next to the Package name and select the itso.redbrook.servlet package.
4. Double-click on the servlet to open it in the editor. Replace the existing code with the code in RBSearchCodeBook.java you downloaded from the additional material provided by this redbook.


New Servlet

Servlet

Create a new Servlet.

Folder: /RedBrookWeb/JavaSource

Java package: itso.redbrook.servlet

Class name: RBSearchCodeBook

Superclass: javax.servlet.http.HttpServlet

Figure 9-10 Create the RBSearchCodeBook servlet

The servlet redirects the request to the model, and we explain how the model works in the following section. The result is displayed in the JSP page in the next page request parameter.

In this module, the next page is the same JSP file that contains the text search form. We test a session variable of type vector where the results should be stored. If it is not empty, the results are displayed under the form.

Example 9-2 shows a code snippet of RBSearchCodeBook.java's doTask() method, which handles the main work in this servlet.

Example 9-2 RBSearchCodeBook.java - Code snippet for doTask method

```
public void doTask(HttpServletRequest req, HttpServletResponse resp)
   throws ServletException, IOException {

   String SCriteria = req.getParameter("word_value");
   String nextPage =  req.getParameter("nextPage");

   if( SCriteria == null) {
      this.dispatch(req, resp, nextPage);
   }

   ResourceBundle test = ResourceBundle.getBundle("Config");
   try
   {
   RBICMTextSearchModel textSearchModel = new RBICMTextSearchModel();
   textSearchModel.setConnectParams(
         (String)test.getObject("ICMServer"),
         (String) test.getObject("ICMUser"),
         (String) test.getObject("ICMPasswd"),"");
   textSearchModel.getTSResults(req);
   }
   catch (Exception e)
   {
      System.out.println("exception" + e.getMessage());
   }
   this.dispatch(req, resp, nextPage);
   }
```

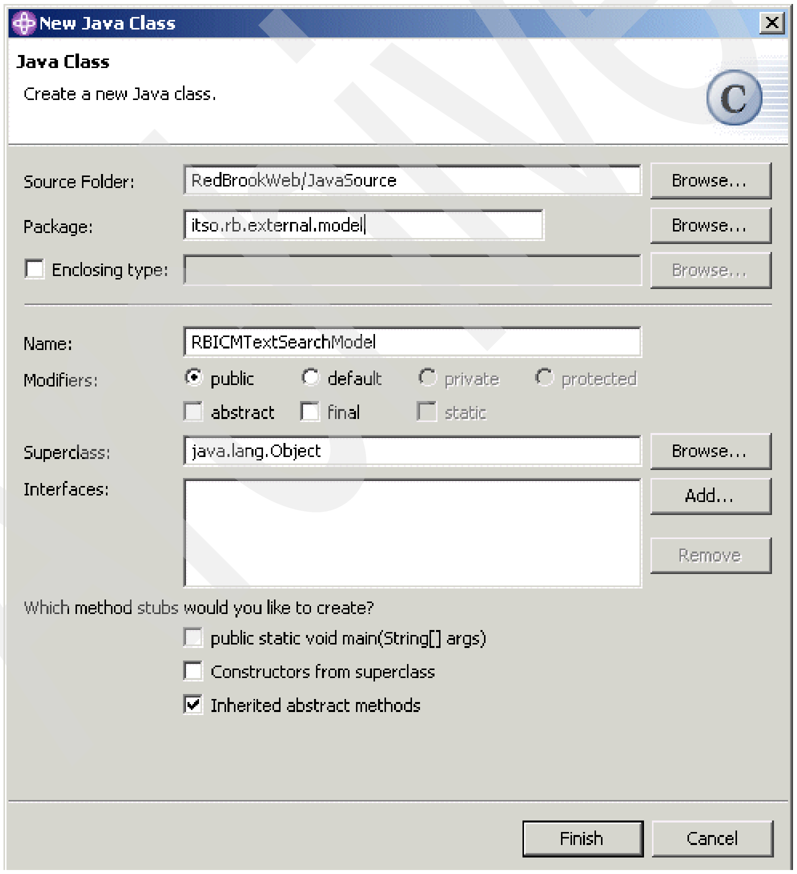
The servlet uses a properties file config to get the connection parameters (such as Library Server database name, user ID, and password) and send them to the model class as parameters before invoking its text search methods getTSResults.

The properties file should be located under the WebContent folder under your Web project. This is an easy way to customize your connection to reflect your environment.

Adding the model bean

To add the model bean:

1. Select the **itso.rb.external.model** created in "Adding packages" on page 333.
2. From the context menu, select **New** → **Class**. Make sure that the package is selected, enter `RBICMTextSearchModel` in the Name field. Accept the default and click **Finish**.
3. Double-click RBICMTextSearchModel class to open it in the editor. Replace the existing code with the code in RBICMTextSearchModel.java from the additional material provided by this redbook.



Create a new Java class.

Source Folder: RedBrookWeb/JavaSource
Package: itso.rb.external.model
[ ] Enclosing type:
Name: RBICMTextSearchModel
Modifiers: [✓ public] default private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods

Figure 9-11 Create the model class RBICMTextSearchModel.java

Examine the code of the model class; it has methods to connect and disconnect to the Content Manager back-end datastore, in addition to the text search functionality which is encapsulated in the method getTSResults. The results are stored in data beans RBCodeBookBean, and added to a session object of type Vector for viewing in a JSP file. (See the downloaded RBCodeBookBean.java from the additional material provided by this redbook for the source code.)

RBICMTextSearchModel.java does the following tasks:

1. Calls the connect method.
2. Gets the request parameters handed by and from the Control Servlet.
3. Constructs the query string.
4. Executes the query using the datastore object and the proper options.
5. Retrieves the results in a cursor object and creates an tolerator.
6. Retrieves the results items and stores information in a data bean, and then to a vector. This data is mainly the contents or value of the two attributes: BP_CODEBOOK_ID and BP_CODEBOOK_SUB (see “Creating the attribute” on page 325 for attribute information), in addition to the part ID of the resource item retrieved as part of the document and its mime type.
7. Stores the results vector in the session object.

Example 9-3 shows a code snippet for getTSResults() method of RBICMTextSearchModel.java.

Example 9-3 RBICMTextSearchModel.java - Code snippet for getTSResults method

```
public void getTSResults(javax.servlet.http.HttpServletRequest req ) throws
DKException, Exception {

   if(dsICM == null) {
      connect();
   }
   String strSearchWord = (String) req.getParameter("word_value");
   String strSearchSentence = (String) req.getParameter("sentence_value");
   String query = " /BP_CODE_BOOK[contains-text( .//ICMPARTS/@TIEREF,\"
'"+strSearchWord+"' \")=1] ";
   DKNVPair options[] = new DKNVPair[3];
   options[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No
Maximum (Default)
   options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,    new
Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
   options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);

   dkResultSetCursor cursor = dsICM.execute(query,
DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
```

```
   System.out.println("Accessing Result Set Cursor...");
   DKDDO  ddo;
   String itemId = "";
   String bookID ="";
   String bookSubject="";

   Vector resultsVector=new Vector();
   String pidxdo=null;
   while((ddo = cursor.fetchNext())!=null)
      {
      String itemElement[] = new String [3];
      itemId = ((DKPidICM)ddo.getPidObject()).getItemId();
      ddo.retrieve(DKConstant.DK_CM_CONTENT_YES);

      bookID = (String)
ddo.getData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_CODEBOOK_ID"));
      bookSubject = (String)
ddo.getData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"BP_CODEBOOK_SUB"));
      RBCodeBookBean cbBean = new RBCodeBookBean();
      cbBean.setId(bookID);
      cbBean.setSubject(bookSubject);
      short dataid =
ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
      if(dataid > 0)
         {
         // if a DKParts collection exists, print contents
         DKParts dkParts = (DKParts) ddo.getData(dataid);
         // obtain the DKParts collection

         dkIterator iter = dkParts.createIterator();
         // Create an iterator to go through Collection
         while(iter.more())
            {
            // While there are children, print list
            DKDDO part = (DKDDO)iter.next();
            part.retrieve();
            DKPidICM PIDx =(DKPidICM) part.getPidObject();
            pidxdo = PIDx.pidString();
            // Move pointer to next part & return that object.
            }//while iter
            cbBean.setPartId(pidxdo);
            cbBean.setMimetype("application/msword");
         }//if dataid>0
         resultsVector.add(cbBean);

         }// end of while

         req.getSession().setAttribute("codeBooks", resultsVector);
```

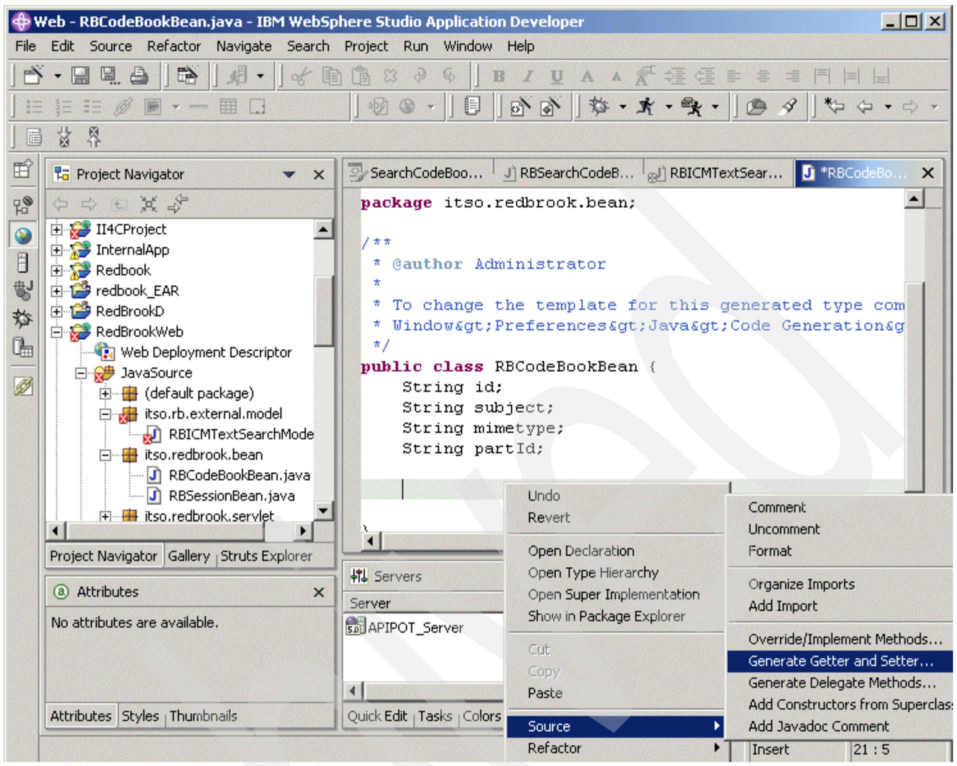
```
            cursor.destroy();
            dsICM.disconnect();
        }
}
```

As mentioned earlier, refer to the corresponding downloaded file, RBICMTextSearchModel.java, from the additional material provided by this redbook, for source code.

Adding the data bean

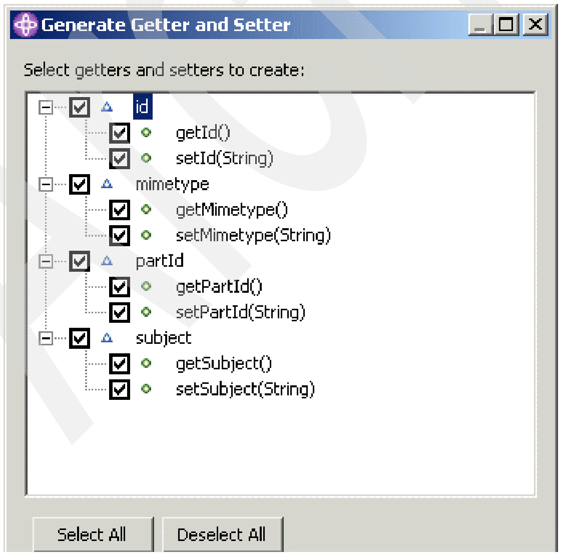
To add the data bean:

1. Select the **itso.redbrook.bean** package.
2. From the context menu, select **New** → **Class**.
3. Enter `RBCodeBookBean` in the Name field.
4. Double-click on the class to open it in the editor, and edit the four class members of type String in the class body.
   - ID
   - subject
   - mimetype
   - partId
5. Click inside the class body again to bring the context menu.
6. Select **Source** → **Generate Setter and Getter** (see Figure 9-12).
7. Make sure you select all the class members in the Generate Getter and Setter dialog window (see Figure 9-13).
8. Click **OK** to save your selections.


Web - RBCodeBookBean.java - IBM WebSphere Studio Application Developer
File Edit Source Refactor Navigate Search Project Run Window Help
Project Navigator
II4CProject
InternalApp
Redbook
redbook_EAR
RedBrookD
RedBrookWeb
Web Deployment Descriptor
JavaSource
(default package)
itso.rb.external.model
RBICMTextSearchMode
itso.redbrook.bean
RBCodeBookBean.java
RBSessionBean.java
itso.redbrook.servlet
Project Navigator | Gallery | Struts Explorer
Attributes
No attributes are available.
Attributes | Styles | Thumbnails
SearchCodeBoo... | RBSearchCodeB... | RBICMTextSear... | *RBCodeBo...
Servers
Server
APIPOT_Server
Quick Edit | Tasks | Colors
Undo
Revert
Open Declaration
Open Type Hierarchy
Open Super Implementation
Show in Package Explorer
Cut
Copy
Paste
Source
Refactor
Comment
Uncomment
Format
Organize Imports
Add Import
Override/Implement Methods...
Generate Getter and Setter...
Generate Delegate Methods...
Add Constructors from Superclas
Add Javadoc Comment
Insert
21 : 5

```
package itso.redbrook.bean;

/**
 * @author Administrator
 *
 * To change the template for this generated type com
 * Window&gt;Preferences&gt;Java&gt;Code Generation&g
 */
public class RBCodeBookBean {
    String id;
    String subject;
    String mimetype;
    String partId;
```

Figure 9-12   Generate setter and getter for text search bean


Generate Getter and Setter
Select getters and setters to create:
Select All
Deselect All

- [✓] id
  - [✓] getId()
  - [✓] setId(String)
- [✓] mimetype
  - [✓] getMimetype()
  - [✓] setMimetype(String)
- [✓] partId
  - [✓] getPartId()
  - [✓] setPartId(String)
- [✓] subject
  - [✓] getSubject()
  - [✓] setSubject(String)

Figure 9-13   Setter and getter dialog

RBCodeBookBean.java provides methods to set and get Id, MIME type, partId, and subject.

Example 9-4 provides a code snippet for getting and setting partId.

Example 9-4 RBCodeBookBean.java - Code snippet for get and set partId methods

```
public String getPartId() {
   return partId;
}
/**
 * @param string
 */
public void setPartId(String string) {
   partId = string;
}
```

Display the results

Following the MVC model, we need to create a JSP file to view the results retrieved from the model class RBICMTextSearchModel. You can create a new JSP and add the code to retrieve the results vector from the session object and iterate trough the vector to display the results.

Or, as in the case of our application, you can import or cut and paste the SearchCodeBook.jsp code from the additional material provided by this redbook.

SearchCodeBook.jsp tests the size of the vector in the session variable. If it is 0, the message `No code books where found. Please enter a search criteria` is displayed in the page; otherwise, it displays the results in a table. An additional link is provided to display the text document retrieved in a new window.

Example 9-5 shows a code snippet of SearchCodeBook.jsp that displays the searched results in a table.

Example 9-5 SearchCodeBook.jsp - Code snippet to display searched results in table

```
  RBCodeBookBean cbBean;
  for( int i = 0; i < codeBooks.size() ; i++) {

   cbBean = (RBCodeBookBean)  codeBooks.elementAt(i);
   System.out.println("cbBean " + cbBean);
%>
   <TR>

      <TD><%=cbBean.getId() %> </TD>

      <TD><%=cbBean.getSubject() %></TD>
```

```
         <TD><%=cbBean.getMimetype() %></TD>
         <TD align="center"><A
href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(cbBean.getPartId())%>"
target="_blank">view</A></TD>

      </TR>
```

Display the documents

ICMOpenResource.java is a servlet we use to display the text document where the search string was found. Based on our model, the text document is a part of a Document Model type.

The new servlet should be created in the default package under the JavaSource folder. The servlet performs the following functions:

1. Retrieves the request parameter (the part ID).
2. Connects to the Content Manager.
3. Retrieves the document using its part ID (`dsICM.createDDO(inPidString)`).
4. Retrieves the document part in a DKLobICM.
5. Gets the contents of the DKLobICM in a byte array.
6. Sets its mime type and displays the content according to the mime type.

Note: The servlet checks the mime type. If it is an image, that is, a mime type of JPEG or GIF, the content is displayed in an html <IMG > tag; otherwise, the content is sent to the servlet output stream to be displayed in the browser using the appropriate plug-ins.

Example 9-6 shows a code snippet of ICMOpenResource.java that displays the content of a document.

Example 9-6 The ICMOpenResource.java - Code snippet to display content of a doc

```
/// Create a DKLobICM (lob) with the passed in pidstring (inPidString)
   DKLobICM lob = (DKLobICM) dsICM.createDDO(inPidString);
      // Retrieve just enough information to find out its mimetype
   lob.retrieve(DKConstant.DK_CM_CONTENT_NO);
      // Set the string (mimeType) to the mime type of (lob)
   String mimeType = lob.getMimeType();
      // If the mimetype is gif or jpeg, we will return some text/html and
      // an IMG SRC tag of the content URL
    if (mimeType.equals("image/gif") || mimeType.equals("image/jpeg"))
        {
        // Set the string (url) to the URL
        //String url = lob.getContentURL(DK_CM_CHECKOUT, -1, -1);
       String url[] = lob.getContentURLs(DK_CM_RETRIEVE ,DK_CM_VERSION_LATEST,
-1, -1, DK_ICM_GETINITIALRMURL  );
        // Return some HTML, and be done
        response.setContentType("text/html");
        out.println("<html><head><title>Open Resource</title></head><body>");
        for(int i=0; i < url.length ; i++ ) {
          out.println("<IMG SRC=\"" + url[i] + "\">");
        }
      } else {
      // we are retrieving non-gif or jpeg image content
      response.setContentType(mimeType);
        lob.retrieve(DKConstant.DK_CM_CONTENT_ONLY);

        // Setup the storage to hold the object by allocating its length in
bytes
        //$$$
        byte[] content = new byte[(int) lob.length()];
        // Put the content of the blob into the storage you just allocated
        content = lob.getContent();

        // write the content out using the mime type
        out.write(content);
      }
```

Test your text search module

At this point, all the components required for the text search module are ready to use.

To summarize, we have:

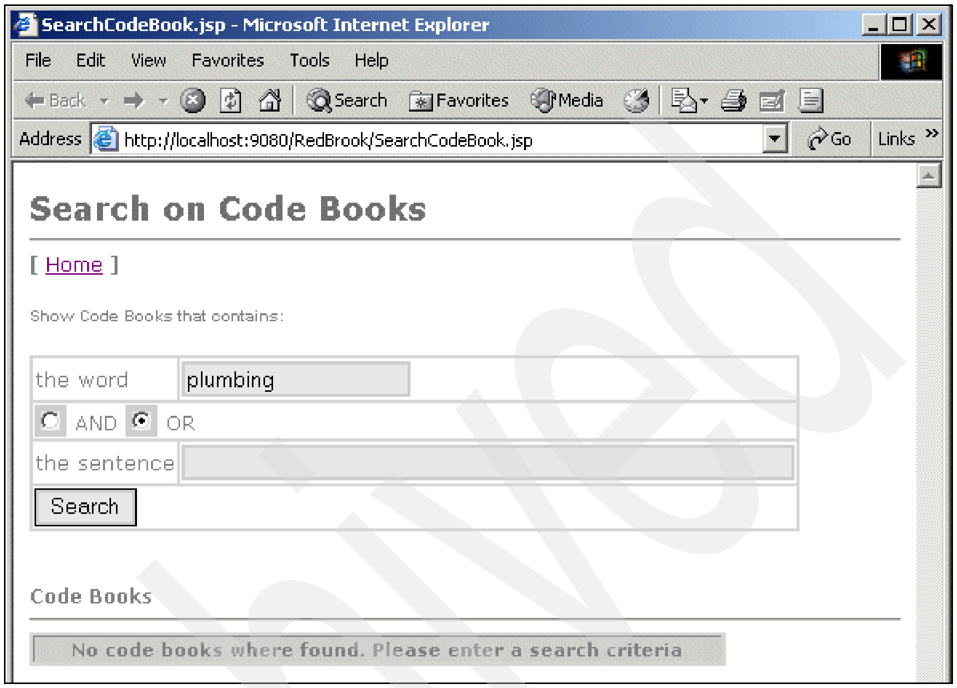
- One JSP file: SearchCodeBook.jsp
- Two servlets: ICMOpenResource.java and RBSearchCodeBook.java
- A Java class that implements our model RBICMTextSearchModel.java and a data bean RBCodeBookBean.java.

To test the code at this stage:

1. Run the SearchCodeBook.jsp on the server.

Note: In order to test the text search code on a server, *the server must have been configured with the appropriate classes.* If you have not done this before, please refer to 6.2, “Development environment setup” on page 97, *and* the Web project must be added to the server project.

2. Figure 9-14 show the text search form. Enter `plumbing` in the word field and click **Search**.


SearchCodeBook.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go
Links

http://localhost:9080/RedBrook/SearchCodeBook.jsp

Search on Code Books

[Home]

Show Code Books that contains:

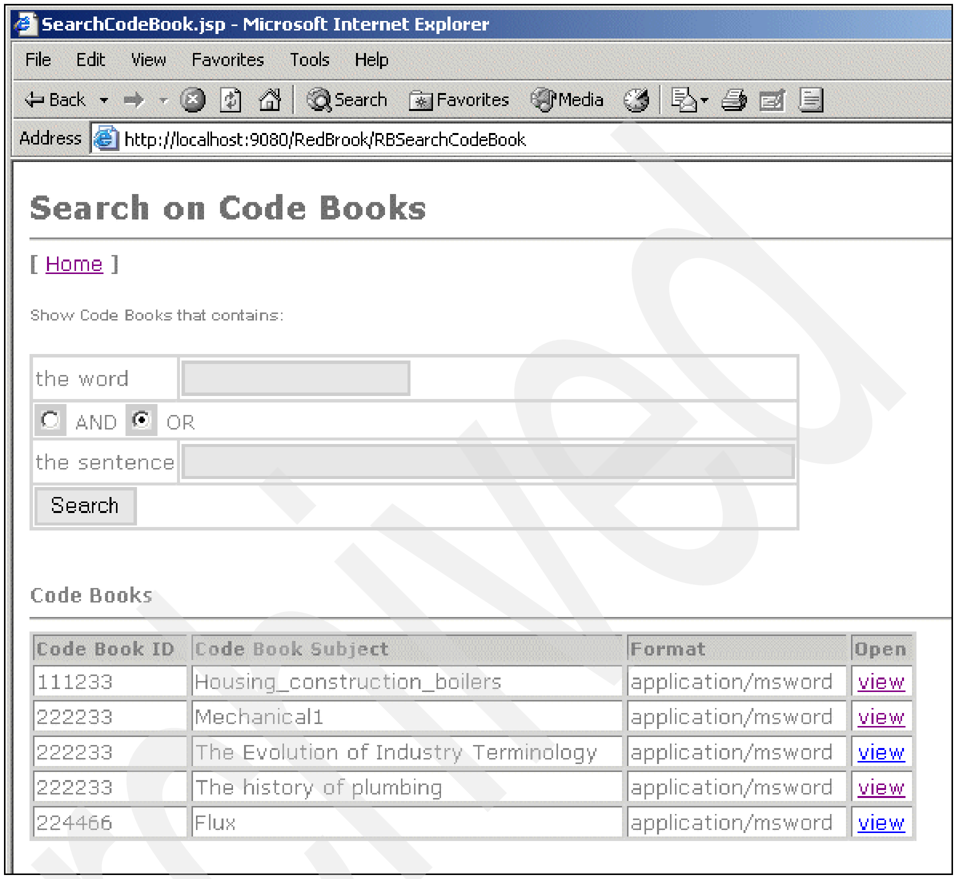
the word	plumbing
AND [✓ OR]	
the sentence	
Search	

Code Books

No code books where found. Please enter a search criteria

Figure 9-14 Text search form

3. Figure 9-15 shows a sample of the searched results.


SearchCodeBook.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/RedBrook/RBSearchCodeBook

Search on Code Books

[Home]

Show Code Books that contains:

the word	
AND [✓ OR]	
the sentence	
Search	

Code Books

Code Book ID	Code Book Subject	Format	Open
111233	Housing_construction_boilers	application/msword	view
222233	Mechanical1	application/msword	view
222233	The Evolution of Industry Terminology	application/msword	view
222233	The history of plumbing	application/msword	view
224466	Flux	application/msword	view

Figure 9-15 Sample output of text search results

4. The results listed show the values of the two user-defined attributes, BP_CODEBOOK_ID and BP_CODEBOOK_SUB, the document mime type and a link to view the document in a new browser window.
5. Click **view** link to view the documents.

9.5 Conclusion

In this chapter, we introduced the text search basics within Content Manager V8.

We illustrated how to set the Content Manager for a text search functionality. If the data model was not created in a previous chapter, we showed you how to manually create an item type and to enable it for text search within its content using the Content manager Administration Client.

We imported code book documents and tested the created item using the Content Manager Windows Client. We also built a text search module to perform the text search on the imported code book documents.

The text module we developed in this chapter provides the ability to search, query and view the text searchable items.

10

Adding document rendering capability

This chapter focuses on rendering documents retrieved from content servers using the Java viewer toolkit. The toolkit's non-visual classes (document services) can be used for document conversion and annotations editing. The toolkit's visual classes can be used to build document viewers. Working again with the Redbrook County building permit application we developed in the previous chapter, we show how to set up the existing project to support viewing functionality.

We cover the following topics:

- Introduction
- Integrating the viewer toolkit

10.1 Introduction

Information Integrator for Content comes with a set of document viewer APIs that help to present content to users.

The *document viewer toolkit* is a set of Java APIs which facilitates how documents are displayed and annotated by users. The toolkit includes a Swing based graphical class. It also includes a set of non-visual classes that provide the document conversion and rendering capabilities.

You can customize the existing graphical interface or build you own. The toolkit includes many document functions, such as rotating, zooming in and out, enhancing, inverting, printing, and annotation. The Content Manager eClient uses this toolkit to provide its viewing functionality.

If a Web client is being developed, you can use the Swing based GUI class that is included within this toolkit to provide an attractive option from having to develop a new document viewer user interface. This class provides the user interface implementation and relies on callback classes (which the developer must implement) in order to provide the document and annotation services functionality.

10.1.1 Viewer architecture

Developing an applet document viewer for a custom Web client consists of two components, an applet and a controller servlet, see Figure 10-1.

The applet and its supporting classes allow for the presentation and rendering of document images. Because the applet runs on the client machines, the processing required to convert and render images are performed on the client machine.

The controller servlet directly communicates with the back-end server (such as updating the annotation parts, obtaining the location of the document and annotations parts, and determining the mime type). In order for the servlet to be able to communicate with a Content Manager Version 8 server, the Content Manager Version 8 connector from Information Integrator for Content and DB2 UDB Runtime must be installed. If a servlet is not used, every client that uses the applet needs to have the Content Manager Version 8 connector and DB2 UDB Runtime client.

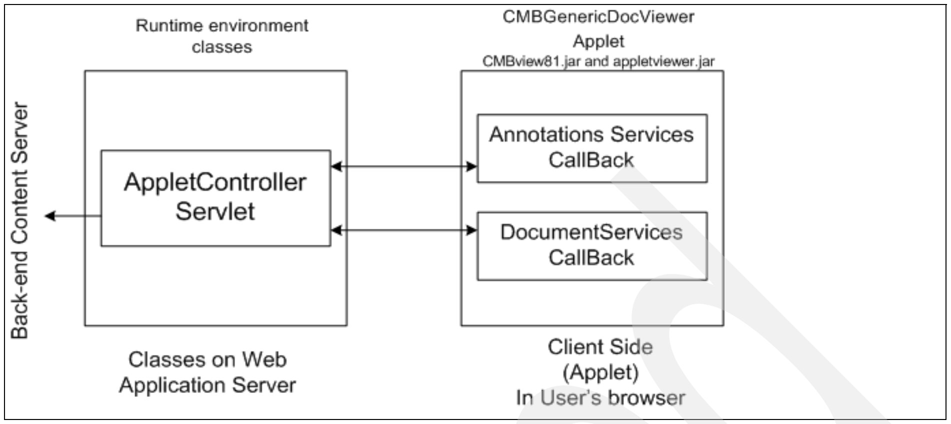

Runtime environment classes
CMBGenericDocViewer Applet
CMBview81.jar and appletviewer.jar
Back-end Content Server
AppletController Servlet
Annotations Services CallBack
DocumentServices CallBack
Classes on Web Application Server
Client Side (Applet) In User's browser

Figure 10-1 Viewer architecture

The applet uses the CMBGenericDocViewer class, which is the JFC based graphical user interface for the document viewer. The class relies on other classes to handle functions requested by users.

The CMBGenericDocViewer class is composed of a number of default toolbars: Operation, Annotation, and Page. Using a viewer configuration file, you can customize the exiting toolbars, and create the new ones.

10.2 Integrating the viewer toolkit

In this section, we integrate the viewer toolkit APIs to our application that we developed in Chapter 8, “Building a case study application” on page 207. The application we developed include two projects, InternetApp and RedBrookWeb. We add the viewer functionality to the InternalApp project, which is designed for RedBrook County personnel. The RedBrookWeb project, which is designed for external internet users, will still use the simple code to view documents without additional facility.

10.2.1 Environment setup

To be able to use the Information Integrator for Content viewer toolkit, the InternalApp project’s build path must include cmbview81.jar.

To set up:

1. Right-click **InternalApp** project, and select **Properties** from the context menu.

2. Select **Java Build Path** from the left panel. Click **Libraries** tab on the right panel. Make sure that the *cmbview81.jar* is there.

   If not, do:

   a. Click Add External JARs.

   b. Browse to <CMBROOT_DIR>\lib, where <CMBROOT_DIR> is the directory where Information Integrator for Content is installed.

   c. Select the cmbview81.jar file.

   d. Click **OK** to save.

   Note that this should have been done in the previous chapter. We highly recommend that you go through the chapters in this redbook in the order that they are presented to you. The chapters in Part 2 of the redbook are highly dependent on each other.

10.2.2 Import the necessary viewer toolkit files

In addition to the cmbview81.jar file, the viewer toolkit requires the following files to be imported into your project:

Table 10-1 Files to be imported to add viewer toolkit functionality

File name	Description	Import directory
AppletController.java	servlet	JavaSource
AppletViewer.jar	applet jar file, used by client to display the object retrieved	WebContent
cmbview81.jar		
AppletViewer.jsp	Contains APPLET tag	
cmv8client.js	Java script file	
CMBViewerConfiguration.properties	viewer toolkit's property configuration file	
rbviewer.store	Certificate files	
rbviewer.cer	Certificate files	

The import directory specifies the directories where these files should be imported to, in the project file.

We create the last two files, rbviewer.store and rbviewet.cer as certificate files. If you want to create your own certificates, you can find the information on how to create certificates from the documentation provided in the AppletViewer.java file. Also refer to 10.2.5, “Viewer toolkit certificates” on page 359 for information concerning the certificates.

To import the viewer toolkit related files:

1. Under the InternalApp project file, right-click **JavaSource**.
2. Select **Import** from the context menu.
3. Select **File system**.
4. Browse to where AppletController.java is downloaded, select the file, and click **Finish**.
5. Repeat the same procedure for the rest of the files, except, import them under WebContent instead of JavaSource.

10.2.3 Implement code changes to use the viewer toolkit

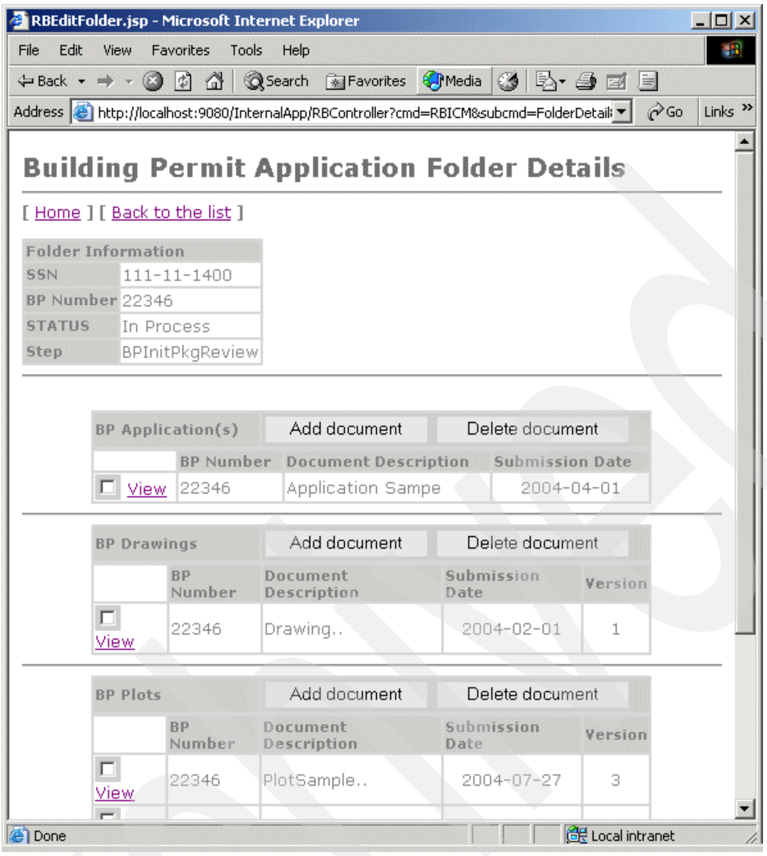
We assume that you have already built the Internal Application code in Chapter 8, “Building a case study application” on page 207. The Internal application’s main functionality is to process building permit applications submitted by a property owner. The application (InternalApp) has many different users performing different roles. The application has the Content Manager Document Routing capabilities to enable the RedBrook county users to perform their tasks over the Intranet using their Web browsers.

Depending on the users’ roles, the InternalApp has worklists that cater to different users. Each user in the county has access to the folders awaiting in the worklist.

When a worklist is displayed, the user can have a link to retrieve the contents of the folder in the worklist. Depending on which worklist is required, the contents of the folder may be of one of three item types:

- BP_APPLICATION
- BP_DRAWING
- BP_PLOT

The JSP file that lists the content of the folders is RBEditFolder.jsp. The list of items found in the folder are displayed according to the item types in three different sections of the page. For each item, there is associated a link to display its content. See Figure 10-2 as an example.


RBEditFolder.jsp - Microsoft Internet Explorer

Address http://localhost:9080/InternalApp/RBController?cmd=RBICM&subcmd=FolderDetail

Building Permit Application Folder Details

[Home] [Back to the list]

Folder Information	
SSN	111-11-1400
BP Number	22346
STATUS	In Process
Step	BPInitPkgReview

BP Application(s) Add document Delete document

	BP Number	Document Description	Submission Date
☐ View	22346	Application Sampe	2004-04-01

BP Drawings Add document Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	22346	Drawing..	2004-02-01	1

BP Plots Add document Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	22346	PlotSample..	2004-07-27	3

Done Local intranet

Figure 10-2 A sample of screen shot for InternalApp

Let us examine the code in the JSP file: RBEditFolder.jsp.

Locate the part of the code shown in Example 10-1. The viewing part is done using the Servlet *ICMOpenResource.java*. The servlet deals with any MIME type for which the browser has plug-ins.

Example 10-1 RBEditFolder.jsp - Code snippet that generates item in folder

```
<DIV align="center">
<TABLE border="0" width="80%">
   <TBODY>
      <TR>
         <TD></TD>
         <TH>BP Number</TH>
         <TH>Document Description</TH>
         <TH>Submission Date</TH>
         <TH>Version</TH>
      </TR>
      <% for( int i=0 ; i < ChildItems.size() ; i++ ) {
           CMBItem CItem = (CMBItem)ChildItems.elementAt(i);
           String pid = CItem.getPidString();
           if( CItem.getEntityName().equals("BP_DRAWING") ) { %>
      <TR>
         <TD><INPUT type="checkbox" name="App" value="1">
         <a
href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(DKLobs[i].getPidObject(
).pidString())%>">View</A>

          </TD>
         <TD><%=CItem.getAttrValue("BP_NUMBER")%></TD>
         <TD><%=CItem.getAttrValue("BP_DOC_DESC")%></TD>
         <TD align="center"><%=CItem.getAttrValue("BP_SUB_DATE")%></TD>
         <TD align="center"><%=CItem.getAttrValue("BP_DOC_VERSION")%></TD>
      </TR>
      <% }
          } %>
   </TBODY>
</TABLE>
</DIV>
```

We need to add an additional piece of code to permit us to use the viewer toolkit functionalities. Because the generic viewer toolkit supports JPEG, GIF and TIFF MIME types, it is important to add a check on the object MIME type before calling the viewer APIs.

Important: Viewer supports only some mime types.

Locate the code in bold in Example 10-1, add the code in Example 10-2.

Example 10-2 Added code in JSP to call the viewer toolkit

```
..
<TR>
<% String MT = DKLobs[i].getMimeType();
   if( MT.equals("image/jpeg") || MT.equals("image/gif") ) { %>
      <a href='javascript:openDoc("<%=pid%>","true")'
title='OpenDocument'>View</a>
<% } else { %>
         <TD><INPUT type="checkbox" name="App" value="1">
         <a
href="ICMOpenResource?PID=<%=java.net.URLEncoder.encode(DKLobs[i].getPidObject(
).pidString())%>">View</A>
<% } %>
          </TD>
```

The added code checks for the object MIME type:

```
<% String MT = DKLobs[i].getMimeType();
   if( MT.equals("image/jpeg") || MT.equals("image/gif") ) { %>
```

If it is an image (image/jpeg or image/gif), a JavaScript, *openDoc* will be called with two arguments, the item ID and a boolean true; otherwise the object will be viewed using our simple servlet ICMOpenResource.

You do not have to add the JavaScript function in the JSP file. It is already included in the cmv8client.js script file that we imported earlier. The function checks the flag sent and determines if the applet is already open, and calls the openAppletViewer function as shown in Example 10-3.

Example 10-3 jcmv8client.js - Code snippet for openAppletViewer method

```
function openAppletViewer( pidString, windowFeatures )
   {
      var appletWinURL = 'AppletViewer.jsp?document=' + pidString;

      appletWin = window.open(appletWinURL, "appletWin", windowFeatures);

      appletAlive = "true";
      setCookie("appletAlive", "true");
   }
```

This script opens a new browser window with the AppletViewer.jsp and passes it the pidString parameter.

The JSP file contains the code to display the Viewer Applet. The <OBJECT> tag is used instead of the <APPLET> tag to be able to download the required plug-in to the browser in order to display the applet. The code base is contained in the two jar files: AppletViewer.jar and cmbview81.jar. Example 10-4 shows APPLET code in AppletView.jsp file.

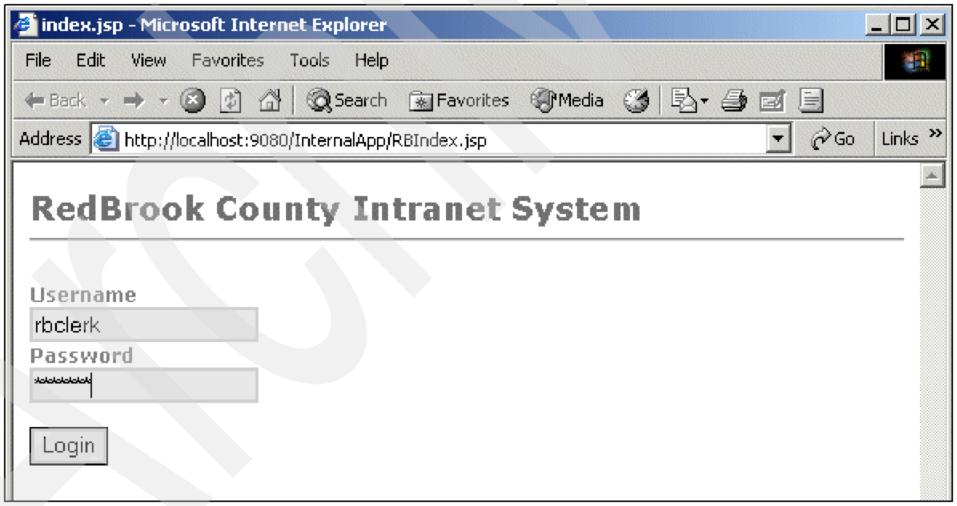
Example 10-4 APPLET code in AppletViewer.jsp

```
<OBJECT CLASSID="clsid:8AD9C840-044E-11D1-B3E9-00805F499D93"
        ID="THEAPPLET"

CODEBASE="http://java.sun.com/products/plugin/autodl/jinstall-1_4_0-win.cab#Ver
sion=1,4,0,0"
        WIDTH="100%"
        HEIGHT="100%">
    <!-- Parameters -->
    <PARAM NAME="code" VALUE="AppletViewer" />
    <PARAM NAME="codebase" VALUE="." />
    <PARAM NAME="type" VALUE="application/x-java-applet;version=1.4"/>
    <PARAM NAME="archive" VALUE="AppletViewer.jar, cmbview81.jar" />
    <PARAM NAME="scriptable" VALUE="true"/>
    <PARAM NAME="mayscript" VALUE="true"/>
   <PARAM  NAME="document" VALUE="<%=document%>">
</OBJECT>
```

10.2.4 Run and test the viewer functionality

After we modify the code to use the viewer toolkit, we need to run and test the modified code:

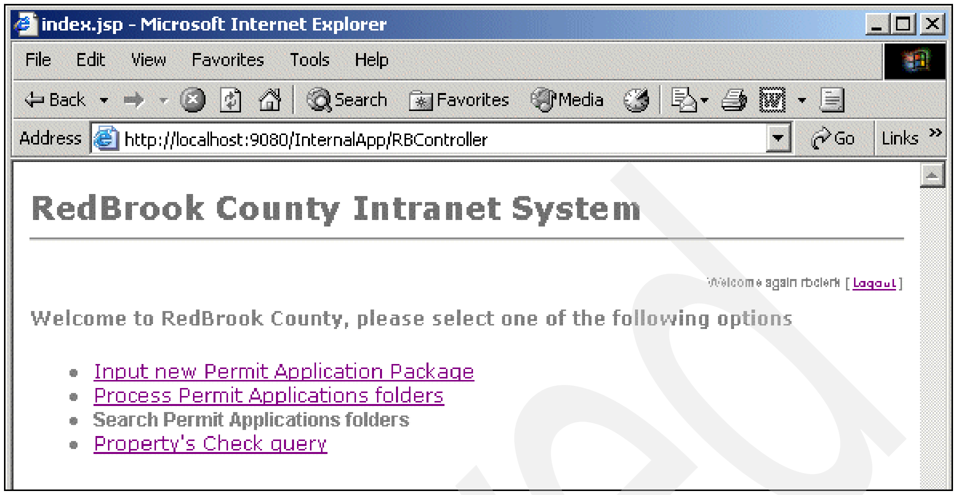
1. Start the InternalApp.
2. Log in. If you follow the application setup, use `rbclerk` for both user ID and password (see Figure 10-3).
3. From the main menu, select **Process Permit Applications folders** (see Figure 10-4).
4. Select **BPInitPkgReview** from the drop-down box and click **Open Folder** (see Figure 10-5).


index.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/InternalApp/RBIndex.jsp
Go Links

RedBrook County Intranet System

Username
rbclerk
Password
****
Login

Figure 10-3 Log in screen


index.jsp - Microsoft Internet Explorer

Address http://localhost:9080/InternalApp/RBController

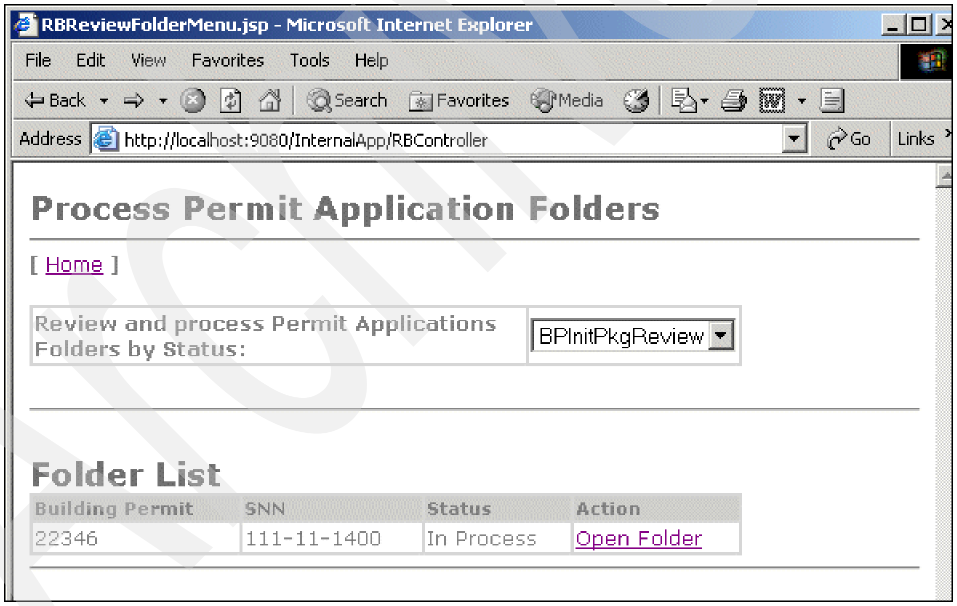
RedBrook County Intranet System

Welcome again rbclerk [Logout]

Welcome to RedBrook County, please select one of the following options

- Input new Permit Application Package
- Process Permit Applications folders
- Search Permit Applications folders
- Property's Check query

Figure 10-4 Internal Application main menu


RBReviewFolderMenu.jsp - Microsoft Internet Explorer

Address http://localhost:9080/InternalApp/RBController

Process Permit Application Folders

[Home]

Review and process Permit Applications Folders by Status:	BPInitPkgReview

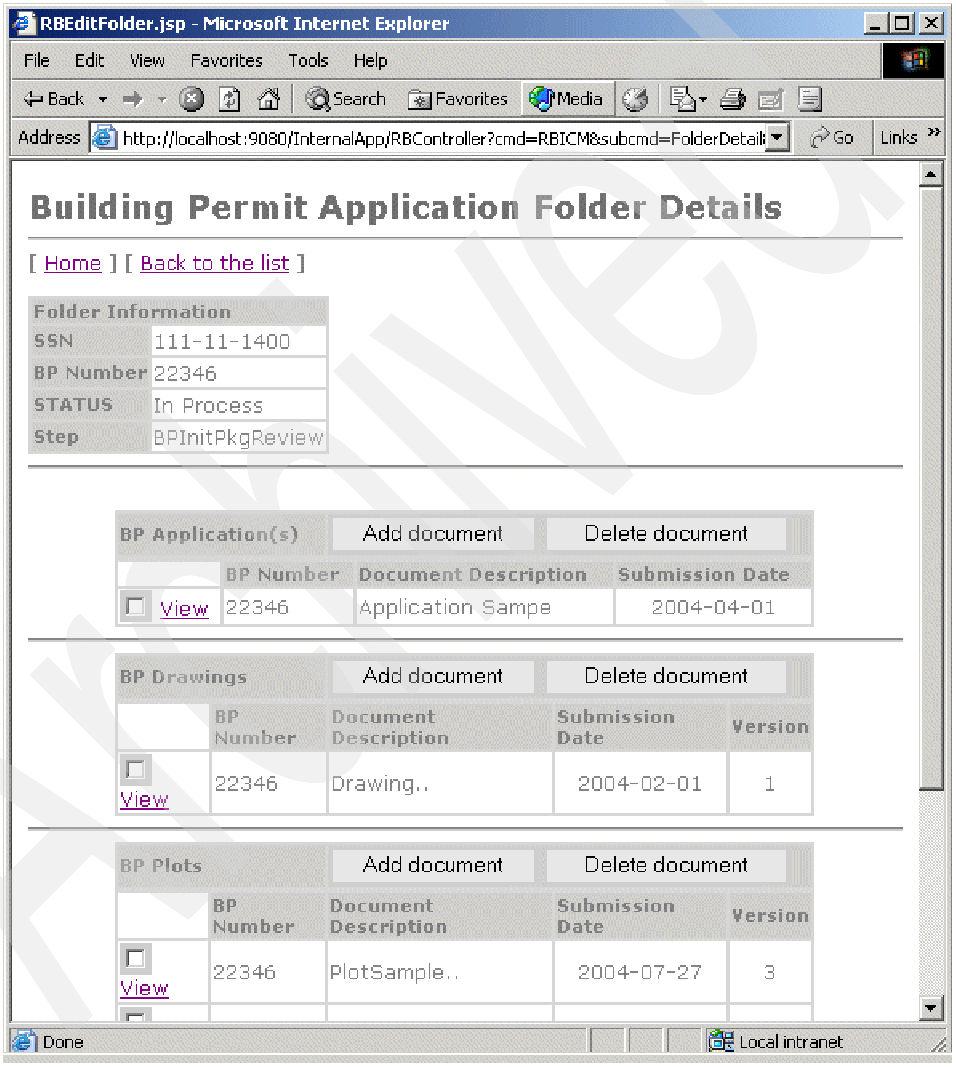
Folder List

Building Permit	SNN	Status	Action
22346	111-11-1400	In Process	Open Folder

Figure 10-5 Select the BPInitPkgReview worklist

5. In the Building Permit Folder Details screen (Figure 10-6), click any of the view links you get. This link is the code that we just added in Example 10-2 on page 353. When invoked, it calls the JavaScript (as explained earlier) and opens a new window with the Viewer applet (see Figure 10-7).

Important: The viewer toolkit supports JDK 1.4 or lower. If you use JDK 1.4.1 or higher, you may have a problem making it work properly. In case you need to develop your application with a higher level of JDK, you should re-compile the code and resolve any issues (such as unsupported security calls) before using the viewer toolkit.


RBEditFolder.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/InternalApp/RBController?cmd=RBICM&subcmd=FolderDetail
Go Links
Done
Local intranet

Building Permit Application Folder Details

[Home] [Back to the list]

Folder Information	
SSN	111-11-1400
BP Number	22346
STATUS	In Process
Step	BPInitPkgReview

BP Application(s) — Add document — Delete document

	BP Number	Document Description	Submission Date
☐ View	22346	Application Sampe	2004-04-01

BP Drawings — Add document — Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	22346	Drawing..	2004-02-01	1

BP Plots — Add document — Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	22346	PlotSample..	2004-07-27	3

Figure 10-6 Edit folder window RBEditFolder.jsp

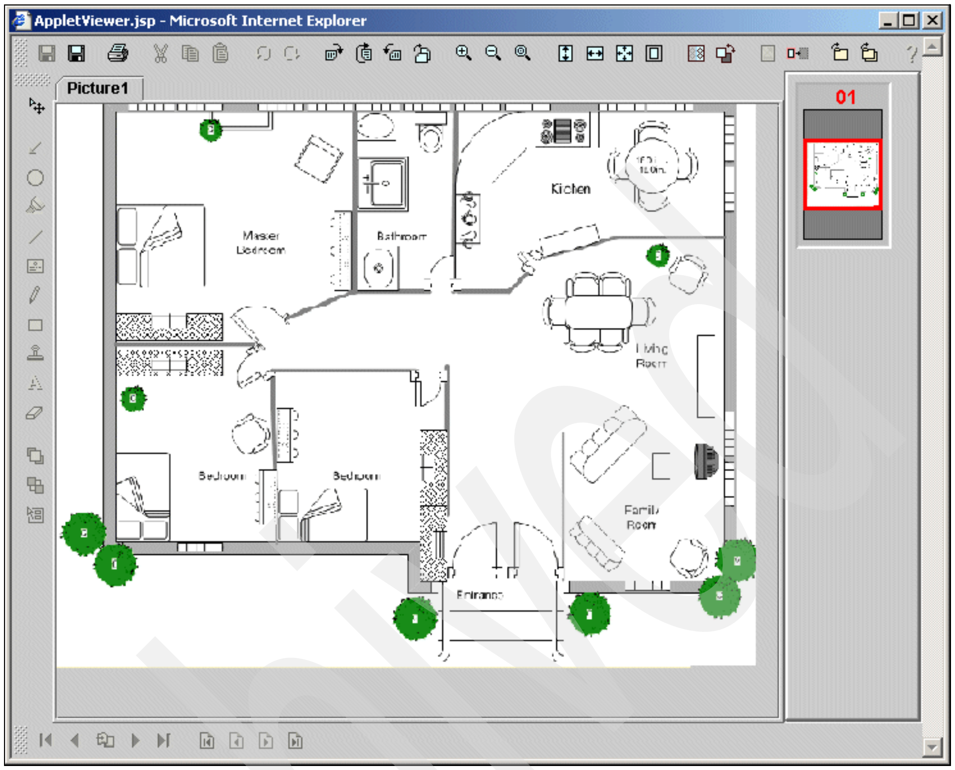


Figure 10-7 Viewer applet

10.2.5 Viewer toolkit certificates

If you decide to modify the files of the AppletViewer.jar file, you can rebuild the jar file and the certificates files by following these instructions.

To create a new jar file:

1. Go to a command window.
2. Change to the <wsad workspace>\InternalApp\Web Content directory and remove the AppletViewer.jar file.

   ```
   cd "<wsad workspace>\InternalApp\Web Content"
   del AppletViewer.jar
   ```

3. Change to the <wsad workspace>\InternalApp\Web Content\WEB-INF\classes directory.

   ```
   cd "<wsad workspace>\InternalApp\Web Content"\WEB-INF\classes"
   ```

4. Use the "jar -cvf" command to create a new AppletViewer.jar file containing the following files:

   – AppletViewer$AnnotationServicesCallbacks.class
   – AppletViewer$LoadedDoc.class
   – AppletViewer$StreamingDocServicesCallbacks.class
   – AppletViewer.class

   Here is the command:

   ```
   jar -cvf AppletViewer.jar *.class
   ```

To sign the new jar file:

1. Go to a command window.
2. Change to the <wsad workspace>\AppletViewer\Web Content directory:

   ```
   cd "<wsad workspace>\InternalApp\Web Content"
   ```

3. Generate a test certificate by entering:

   ```
   keytool -genkey -keystore viewer.store -alias viewercert -validity 365
   ```

4. Export the certificate by entering:

   ```
   keytool -export -keystore rbviewer.store -alias rbviewercert -file
   rbviewer.cer
   ```

5. Sign the jar file by typing:

   ```
   jarsigner -keystore rbviewer.store AppletViewer.jar rbviewercert
   ```

6. You will be prompted to enter a password. Enter the password that you used to create the key.kdb database during the installation of Content Manager.
7. In the WebSphere Studio Application Developer development environment, right-click **WebContent Folder** and select **Refresh** from the context menu.

10.3 Conclusion

In this chapter, we introduced a viewer toolkit, its architecture, and how to integrate it with an existing Web application. The viewer toolkit has the advantage of implementing the major document viewing functionalities that a user needs without having to do any special code.

The functions it supports include document rotation, annotations, placement, and navigating multiple parts or page navigation. A simple integration calls for importing the necessary toolkit files, and adding a few lines of code in your own application to call it. Remember to check the document type before calling the viewer toolkit, as it does not support all document types.

You should also be aware that the viewer toolkit comes with a set of APIs that allow you to develop a special viewer toolkit to fulfill your business requirements. This includes much more than what we presented and used in this chapter.

Part 3

Developing OnDemand Web applications

In this part of the book, we discuss the development of OnDemand Web applications.

11

Web enabling OnDemand

This chapter provides an overview of the products used to Web enable OnDemand. We discuss the API sets, the flow of data between the user, the Web server(s) and OnDemand, choosing the API set for your environment, and where to find information about OnDemand Web Enablement Kit (ODWEK) and Information Integrator for Content.

We cover the following topics:

- Introduction to ODWEK
- ODWEK samples
- Comparing ODWEK to Information Integrator for Content
- Comparing OnDemand clients

11.1 Introduction

There are two IBM products that allow you to interact with IBM Content Manager OnDemand over the Internet: IBM DB2 Content Manager Information Integrator for Content and IBM DB2 Content Manager OnDemand Web Enablement Kit (ODWEK). This chapter gives a short overview of ODWEK and a comparison of the two products.

IBM DB2 Content Manager Information Integrator for Content

Information Integrator for Content enables accessing and managing OnDemand data from the Web. It also works with IBM Content Manager to provide Web users a *federated* view of many back-end repositories such as Content Manager, OnDemand, DB2, Oracle, and other repositories. Information Integrator for Content uses connectors to connect to these back-end systems and give the users a single hit list from all the back-end repositories. Information Integrator for Content also has a tool kit to enable you to develop your own connector to other back-end systems.

Information Integrator for Content provides broad information integration and access to:

- Unstructured digital content such as text, XML and HTML files, document images, computer output, audio, and video
- Structured enterprise information via connectors to relational databases
- Lotus Notes® Domino databases and popular Web search engines using IBM Lotus Extended Search
- Objects within business process workflows

These are some of the benefits of using Information Integrator for Content:

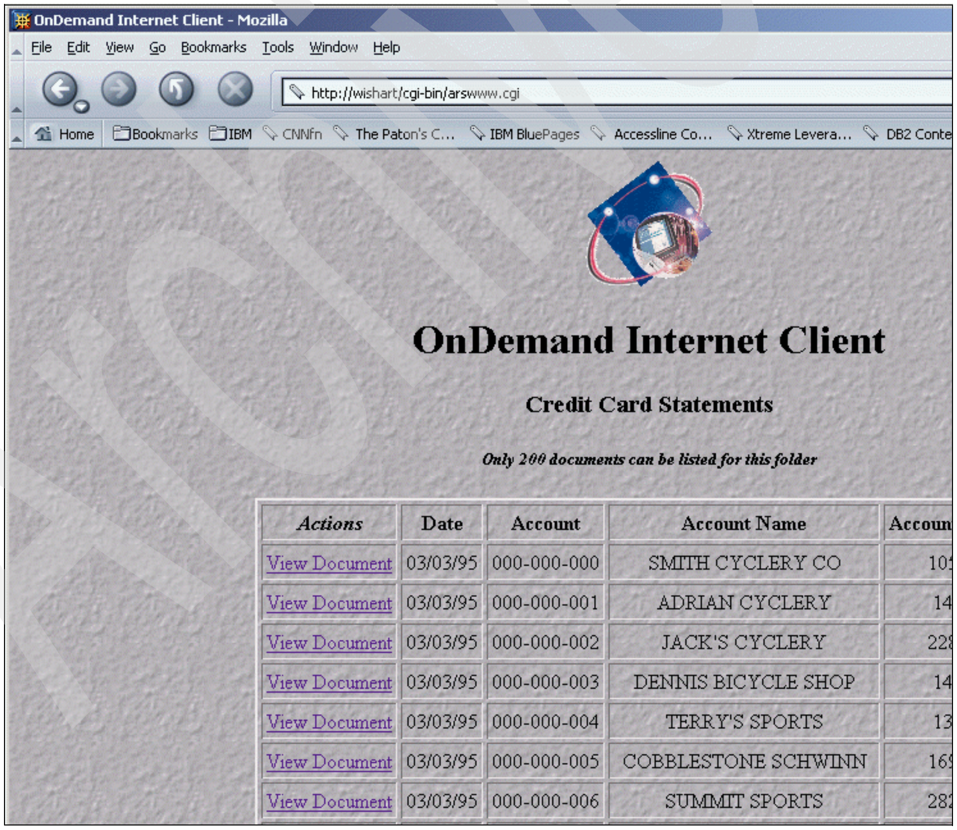
- Users can personalize data queries and search extensively for very specific needs across traditional and multimedia data sources.
- Developers can more rapidly develop and deploy portal applications with the information integration application development toolkit.

To learn more about Information Integrator for Content, see Chapter 5, “Information Integrator for Content programming overview” on page 77.

11.1.1 OnDemand Web Enablement Kit

ODWEK started its life as an IBM services offering called OnDemand Internet Client and later (about Version 2.1.1.9 of OnDemand) became an IBM product called OnDemand Web Enablement Kit, also known as ODWEK. Pronounced “odd-wek” or “odd-wick”. As of this writing, ODWEK is at release 7.1.1.1. Over the years many functions and features have been added. The most important addition was the Java Application Programming Set (APIs) with release 7.1.0.2. ODWEK’s sole purpose in life is to enable OnDemand on the Web.

The OnDemand Web Enablement Kit (ODWEK) is an optional priced feature of OnDemand. ODWEK allows users to access data that is stored in an OnDemand server by using a Web browser or a user-written program. For example, from a browser, a user can log on to an OnDemand server; search a specific folder, and retrieve documents and view them. ODWEK handles user authentication and authorization, searching, retrieval, and Web display of folders and documents. Figure 11-1 shows a partial sample ODWEK screen.


OnDemand Internet Client - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://wishart/cgi-bin/arswww.cgi
Home
Bookmarks
IBM
CNNfn
The Paton's C...
IBM BluePages
Accessline Co...
Xtreme Levera...
DB2 Conte

OnDemand Internet Client

Credit Card Statements

Only 200 documents can be listed for this folder

Actions	Date	Account	Account Name	Accoun
View Document	03/03/95	000-000-000	SMITH CYCLERY CO	10
View Document	03/03/95	000-000-001	ADRIAN CYCLERY	14
View Document	03/03/95	000-000-002	JACK'S CYCLERY	22
View Document	03/03/95	000-000-003	DENNIS BICYCLE SHOP	14
View Document	03/03/95	000-000-004	TERRY'S SPORTS	13
View Document	03/03/95	000-000-005	COBBLESTONE SCHWINN	16
View Document	03/03/95	000-000-006	SUMMIT SPORTS	28

Figure 11-1 Accessing data stored in OnDemand using ODWEK

ODWEK can search for and retrieve documents from OnDemand servers that are running IBM DB2 Content Manager OnDemand for iSeries Common Server, Version5, IBM DB2 Content Manager OnDemand for Multiplatforms, Version7.1, and IBM DB2 Content Manager OnDemand for z/OS and OS/390, Version7.1.

Note: ODWEK can also search for and retrieve documents from an OnDemand for OS/390 Version 2.1 server. See *IBM DB2 Content Manager OnDemand for Multiplatforms: Web Enablement Kit Implementation Guide* for a list of limitations in this environment.

ODWEK contains several components:

- **OnDemand programming interface:** The programming interface uses standard OnDemand interfaces and protocols to access data stored in an OnDemand server. No additional code is needed on the OnDemand server to support ODWEK. You may use one of the following programming interfaces in your ODWEK application:
  - CGI program. The CGI program provides a way to access OnDemand data from a Web browser. It runs on a workstation that is running an HTTP server, such as the IBM HTTP Server.
  - Java™ servlet. The servlet provides a way to access OnDemand data from a Web browser. It runs on a workstation that is running a Java-enabled HTTP server (running as a Java application server) such as the IBM WebSphere Application Server. The servlet requires Java version 1.2.2 or later.
  - Java Application Programming Interface (Java API). The Java API provides a way to access OnDemand data from a user-written program.It requires Java version 1.2.2 or later.
- **IBM OnDemand AFPWeb Viewer:** The AFP Web Viewer lets users search, retrieve, view, navigate, and print AFP documents from a Web browser.
- **IBM OnDemand Image Web Viewer:** The Image Web Viewer lets users search, retrieve, view, navigate, and print BMP, GIF, JPEG, PCX, and TIFF documents from a Web browser.
- **Line Data Java applet:** The Line Data applet lets users view line data documents from a Web browser.
- **AFP2HTML Java applet:** The AFP2HTML applet lets users view the output generated by the IBM AFP2WEB Transform service offering. The AFP2WEB Transform converts AFP documents and resources into HTML files that can be displayed with the AFP2HTML applet. After installing and configuring the AFP2WEB Transform, an administrator enables the use of the AFP2HTML applet by configuring the ARSWWW.INI file.

11.1.2 ODWEK platforms and Web environments

ODWEK can be installed on the following systems:

- AIX 5.1 or later
- HP-UX 11i or later
- Linux kernel
  - 2.4.9 or later (Red Hat)
  - 2.4.19 or later (SuSe)
- Solaris 8 or later
- Windows 2000 and 2003 Servers

ODWEK supports many different HTTP servers and Web application servers:

- If you plan to use the CGI APIs, then you just need an HTTP server.
- If you plan to use the ODWEK servlet, then you need an HTTP server and a Web application server.
- If you plan to use Java APIs to develop a Web application, then you need a Web application server and an HTTP server.

Figure 11-2 presents a graphic view of the configuration setup.

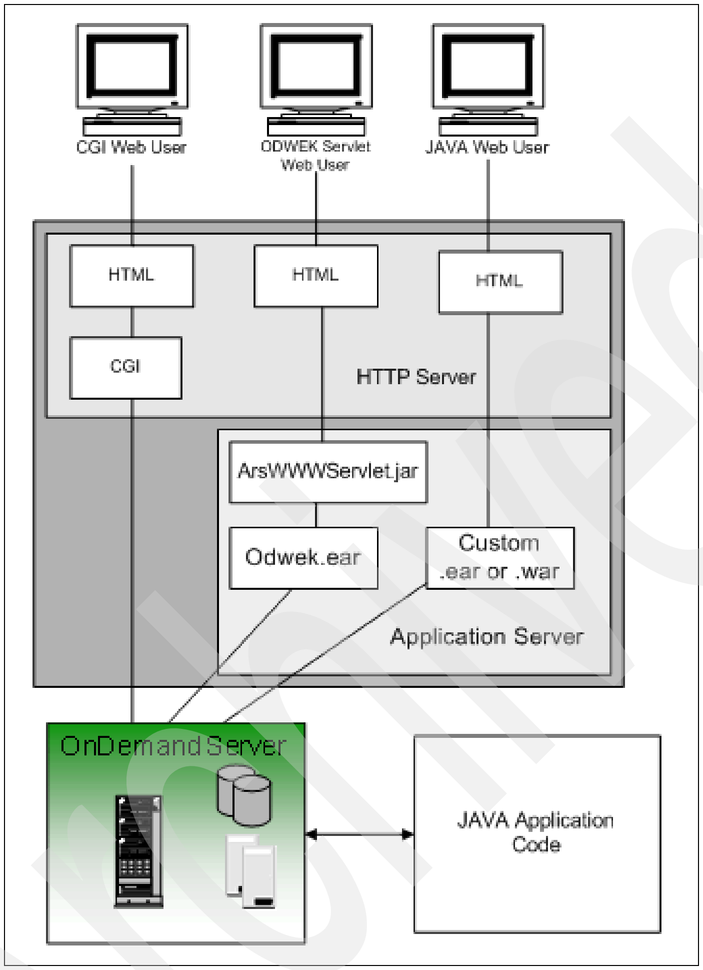


Figure 11-2 CGI, servlet, and Java APIs

11.2 ODWEK samples

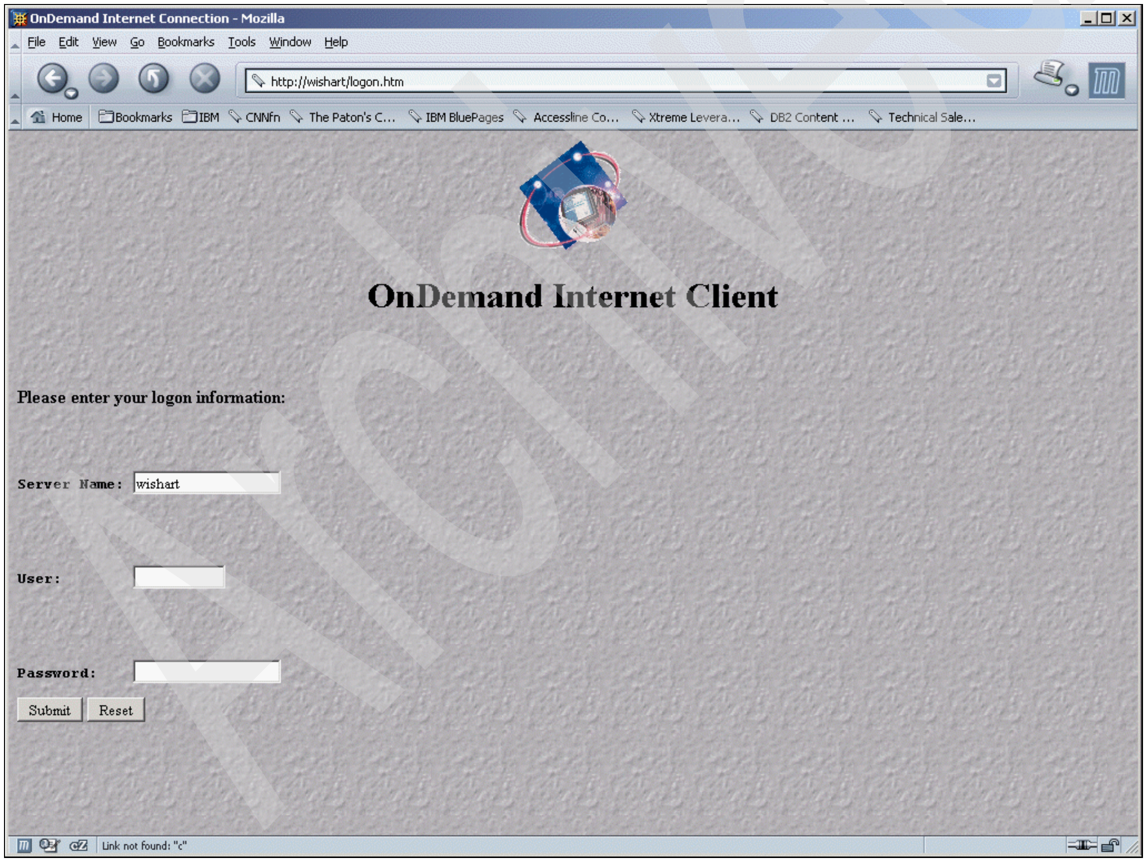
In this section, we discuss samples provided with the ODWEK product.

11.2.1 CGI and servlet HTML samples

When you have successfully installed the CGI or servlet code, you will see the Logon.htm as shown in Figure 11-3. This is the basic, sample HTML code.

In order to see the Logon screen, you need to enter the address of your host (for example, localhost), the directory where you put the Logon.htm file, and finally “logon.htm” on your browser address window:

```
http://yourhost/locationoflogonhtm/logon.htm
```



OnDemand Internet Connection - Mozilla
File Edit View Go Bookmarks Tools Window Help
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Content ... Technical Sale...
Submit
Reset

http://wishart/logon.htm

OnDemand Internet Client

Please enter your logon information:

Server Name: wishart

User:

Password:

Link not found: "c"

Figure 11-3 Sample logon.htm

After typing the Server Name, User ID, and Password, and selecting **Submit**, you should see the Folder Template screen shown in Figure 11-4.

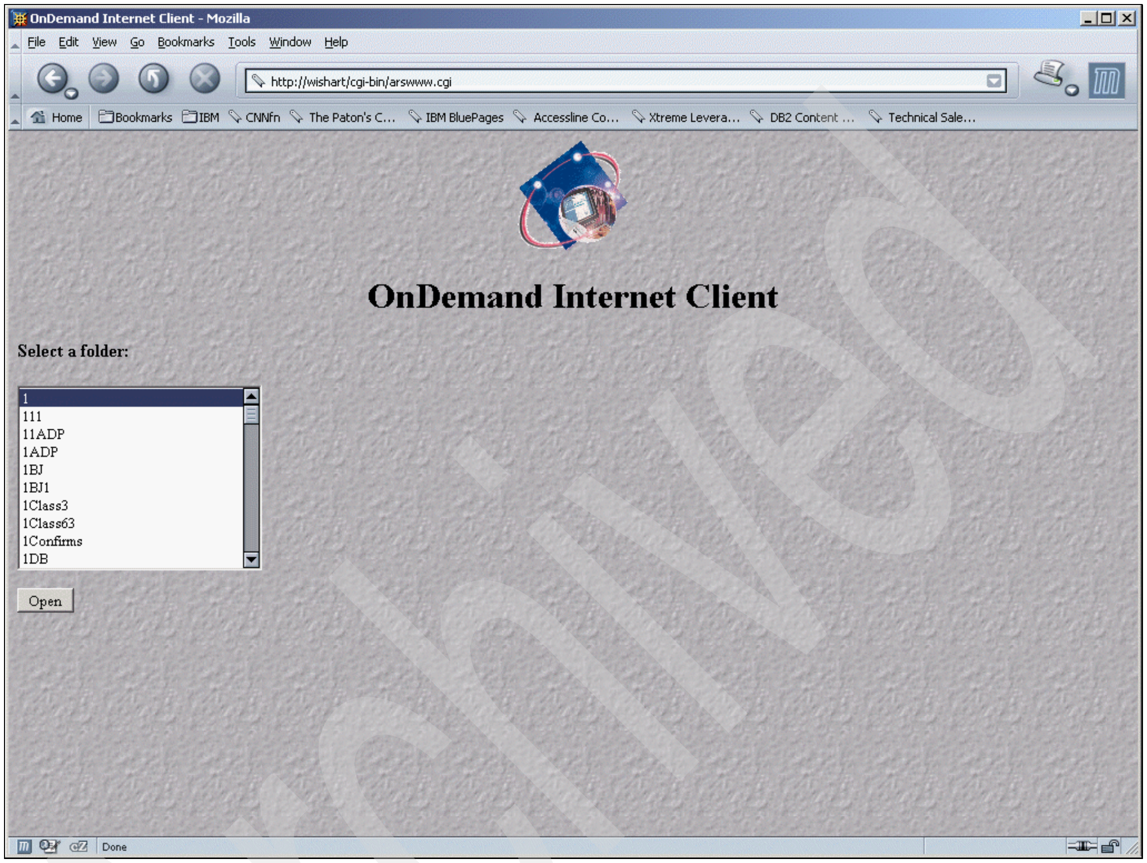
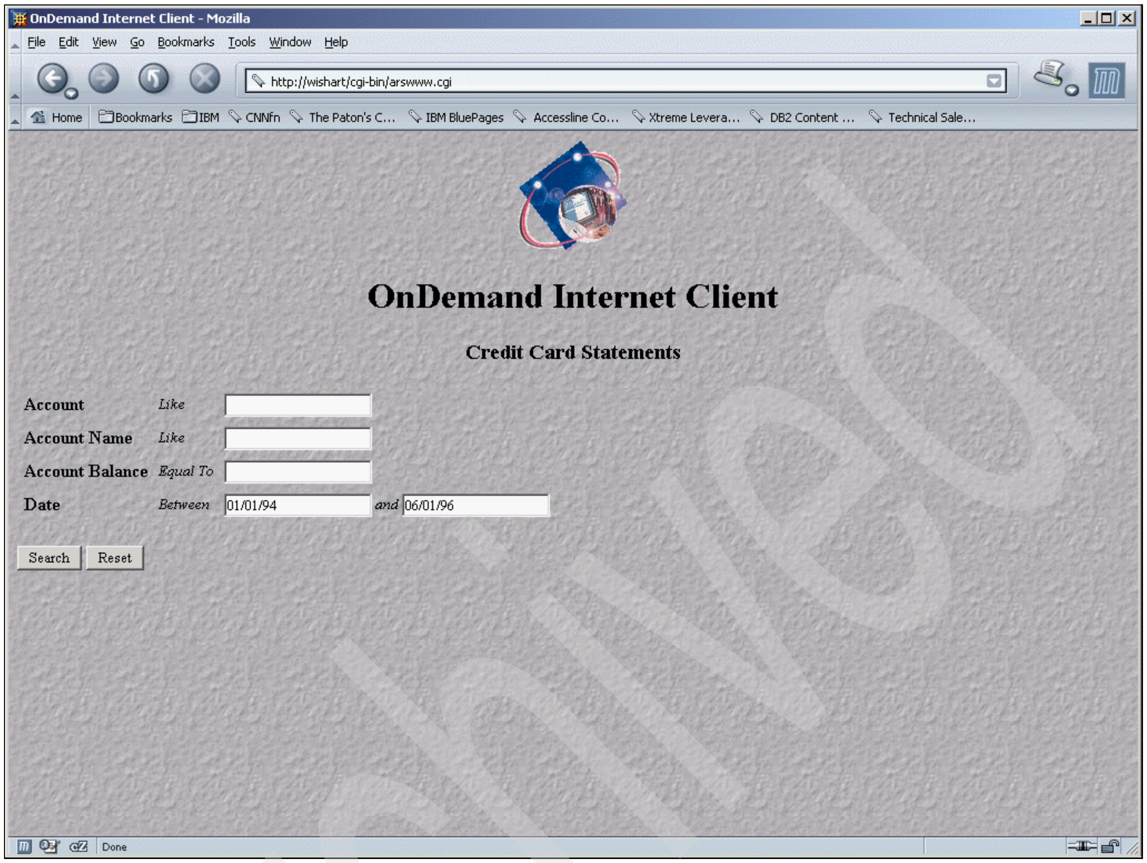

OnDemand Internet Client - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://wishart/cgi-bin/arswww.cgi
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Content ... Technical Sale...
OnDemand Internet Client
Select a folder:
1
111
11ADP
1ADP
1BJ
1BJ1
1Class3
1Class63
1Confirms
1DB
Open
Done

Figure 11-4 Sample folder list

The Folder Screen displays a list of all available folders the logged on user is allowed to see. When an OnDemand System is first installed, the only folder you will see is the System Log folder. The example above has many folders because ODWEK was installed on an existing OnDemand system. After selecting a folder, you click the **Submit** button and you should see the Search Template as shown in Figure 11-5.


OnDemand Internet Client - Mozilla

http://wishart/cgi-bin/arswww.cgi

OnDemand Internet Client

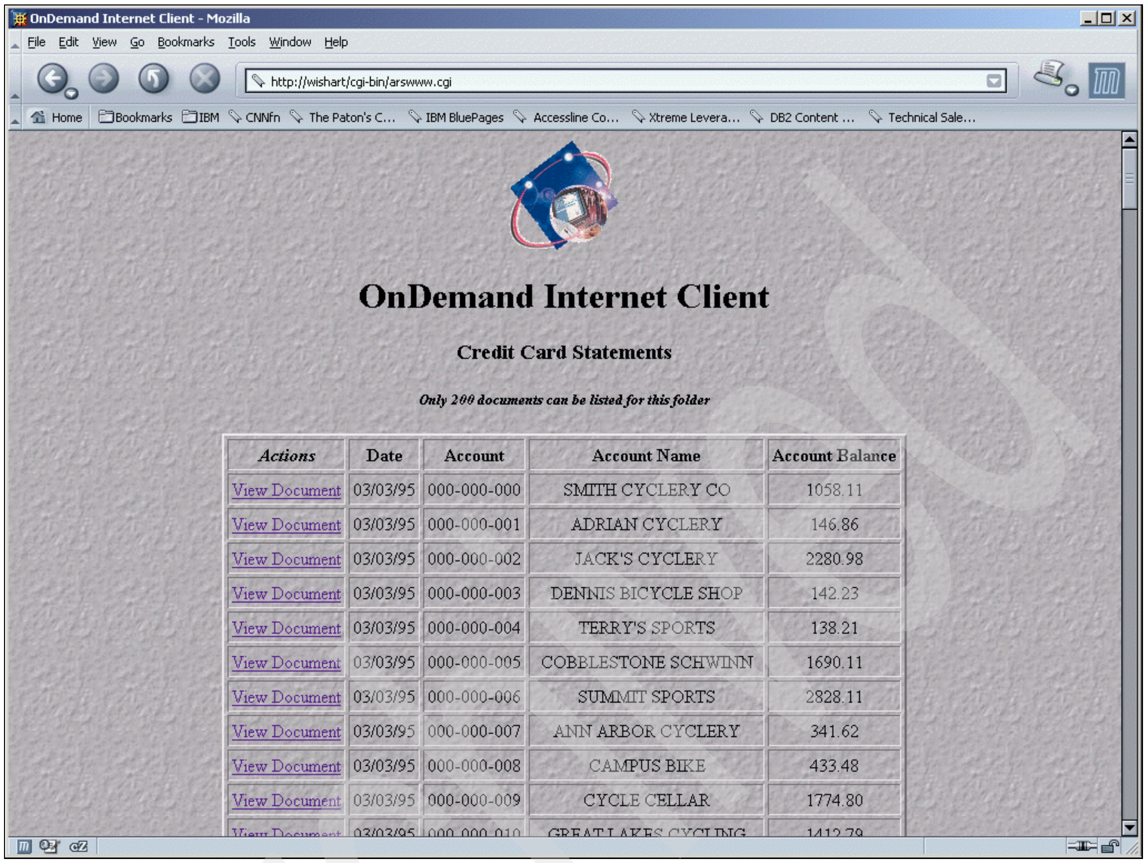
Credit Card Statements

Account	Like			
Account Name	Like			
Account Balance	Equal To			
Date	Between	01/01/94	and	06/01/96

File Edit View Go Bookmarks Tools Window Help
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Content ... Technical Sale...
Search
Reset
Done

Figure 11-5 Sample search template

The Search Template is built from the indexes that were created from your OnDemand Application. In the example above, the indexes to search came from an application already built before ODWEK was installed. If you have not built an OnDemand application yet, then the only search indexes you will see are from the System Log application, which is built when you install OnDemand.


OnDemand Internet Client - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://wishart/cgi-bin/arswww.cgi
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Content ... Technical Sale...

OnDemand Internet Client

Credit Card Statements

Only 200 documents can be listed for this folder

Actions	Date	Account	Account Name	Account Balance
View Document	03/03/95	000-000-000	SMITH CYCLERY CO	1058.11
View Document	03/03/95	000-000-001	ADRIAN CYCLERY	146.86
View Document	03/03/95	000-000-002	JACK'S CYCLERY	2280.98
View Document	03/03/95	000-000-003	DENNIS BICYCLE SHOP	142.23
View Document	03/03/95	000-000-004	TERRY'S SPORTS	138.21
View Document	03/03/95	000-000-005	COBBLESTONE SCHWINN	1690.11
View Document	03/03/95	000-000-006	SUMMIT SPORTS	2828.11
View Document	03/03/95	000-000-007	ANN ARBOR CYCLERY	341.62
View Document	03/03/95	000-000-008	CAMPUS BIKE	433.48
View Document	03/03/95	000-000-009	CYCLE CELLAR	1774.80
View Document	03/03/95	000-000-010	GREAT LAKES CYCLING	1412.79

Figure 11-6 Sample hit list

In Figure 11-6 you should see the results of ODWEK returning the hit list from OnDemand. The example above shows data loaded into an OnDemand application that existed before installing ODWEK. If you have not loaded any of your applications yet, you will see the OnDemand System Log hit list.

There are two HTML samples that come with ODWEK:

- CREDIT.HTML, which supports searching a specific folder you define.
- FRAMES.HTML, which is a different version of CREDIT.HTML that supports frames.

11.2.2 ODWEK documentation

To correctly install and work with ODWEK, you should have all of the following documentation:

- *Web Enablement Kit Implementation Guide*, SG27-1000
- *IBM DB2 Content Manager OnDemand Guide*, SG24-6915
- The readwek.txt and the ReleaseNotes.pdf files associated with OnDemand fixes

11.2.3 Viewing and transforming documents

To view other types of documents that are stored in OnDemand, you must obtain and install the appropriate viewer. For example, to view Adobe Portable Data Format (PDF) documents, we recommend that you obtain the Adobe Acrobat viewer for the browsers that are used in your organization.

To use the viewers and applets, the browser must be Netscape Navigator Version 4.7 or later, or Internet Explorer Version 5.5 or later on a Windows workstation. To convert AFP documents that are stored in OnDemand into PDF documents that you can view with the Adobe Acrobat viewer requires either the AFP2PDF (from IBM Printing Systems) transform or the Xenos d2e Platform.

See Table 11-1 or consult your IBM representative for more information about these optional priced transforms.

Table 11-1 Comparing AFP2WEB to Xenos

	IBM Printing Systems AFP2WEB	**Xenos d2e**
Platforms	NT, Win2K, XP, AIX, HP-UX, Sun Solaris, Linux (x86), OS/400® (PASE), OS/390 or zOS USS	NT, Win2K, XP, AIX, HP-UX, Sun Solaris, Linux, OS/390 and zOS USS
APIs or interfaces	Command line, C++, Java	Java, ODWEK Java APIs multithreaded, Corba, .NET

	IBM Printing Systems AFP2WEB	**Xenos d2e**
Transforms	AFP2PDF AFP2HTML AFP2XML Xerox2AFP - (AIX, OS/390, zOS) PS2PDF PCL2PDF	*Parses:* AFP Line data Xerox Metacode Xerox XES (UDK) PCL TIFF *Generates:* PDF Templates Merge TIFF AFP Metacode PS PCL HTML/CSS XML WML
Indexing	Yes - AFP with Visual Indexer	Yes
Works with	ODWEK or II for Content	ODWEK or II for Content
Pre-reqs	None	REXX (supplied)
Additional Programs	AFP Visual Indexer	d2e Developer Studio d2e Vision

11.3 Comparing ODWEK to Information Integrator for Content

In this section, we compare ODWEK and Information Integrator for Content (II for Content) from various areas.

11.3.1 API differences

ODWEK Java APIs and the OnDemand portion of the II for Content APIs and II for Content OnDemand connector share the same code base. They all provide similar functionalities, with some exceptions. The following list shows the capabilities present in ODWEK Java APIs that are missing from the II for Content APIs and the II for Content OnDemand connector:

- Folders:
  - In Wild card search, you can retrieve a partial list of folders, instead of having to build the entire list of folders.
  - Obtain the number and the names of application groups for a specified folder. Typically this is only used to build customized SQL statements.
- Print:
  - Retrieve list of server printers.
  - Print documents to server printer.
- Retrieval:
  - Cancel retrieve operation.
  - Bulk document retrieval.
- Search:
  - Client cancel search operation.
  - Database segment search when using SQL search. For example, a date range may be specified to narrow the search. The Information Integrator for Content OnDemand connector supports the cancel of the search operation with a user written callback object. The Information Integrator for Content OnDemand connector tool kit includes sample code of a callback class.
  - Single application group search:
    - If the application group is presented as a choice type criteria, the Information Integrator for Cotnent OnDemand connector supports that.
    - If the folder is mapped to multiple application groups, you can force single application group search.
  - Specify whether or not search criteria should be OR'd or AND'd:
    - In Information Integrator for Content, it supports AND only.
    - In Information Integrator for Content OD connector, it supports OR or AND type search.
  - Obtain document location (cache, media, external, and unknown).
- CRUD (Create, read, update, and delete) operations:
  - Update criteria values for a hit, for example, the database fields.
  - Add-hoc document storage.
- Conversion:
  - Xenos converter integration.
  - Application group/Application specification during document conversion (AFP2WEB, Xenos).

11.3.2 ODWEK

ODWEK has the following viewing components and characteristics for its interfaces.

Viewing components:

- AFP Web Viewer (ActiveX/Plug-in) Windows environment
- OnDemand Image Web Viewer (ActiveX/Plug-in)
- Line data Applet:
  - Version 1 requires Java 1.1.8 or greater. No future enhancements planned.
  - Version 2 requires Java 1.4.1 or greater. This is the most current version and it is required for certain types of printing such as DBCS.
- AFP2HTML Applet:
  - Version 1 requires Java 1.1.8 or greater. No future enhancements planned.
  - Version 2 requires Java 1.4.1 or greater. This is the most current version. It is required for certain types of printing such as DBCS.

CGI, servlet or Java interfaces run on AIX, HP-UX, Linux, Sun Solaris, Windows, OS400 (iSeries), OS390 (zSeries®):

- It works with:
  - OnDemand for AIX
  - OnDemand for HP-UX
  - OnDemand for Sun Solaris
  - OnDemand for Windows
  - OnDemand for iSeries (OS400 common server only)
  - OnDemand for zSeries (OS390 V7 and V2).
- It *does not* work with:
  - OnDemand for OS400 (non-common server, Spool File Archive).

11.3.3 eClient/Information Integrator for Content

eClient/Information Integrator for Content has the following viewing components and characteristics for its interfaces.

Viewing components:

- AFP Web Viewer (ActiveX/Plug-in)
- Line data Applet:
  - Does not support the Version 1 applet
  - Version 2 requires Java 1.4.1 or greater

OnDemand Connector APIs provides Windows support (Java and C++ interfaces), AIX support (Java, C++ Interfaces), and Sun Solaris support (Java Interface):

- It works with:
  - OnDemand for AIX
  - OnDemand for HP-UX
  - OnDemand for Linux- when available
  - OnDemand for Sun Solaris
  - OnDemand for Windows
  - OnDemand for iSeries (OS400, including both servers, common server and spool file archive)
  - OnDemand for zSeries (OS390, V2 and V7).

11.3.4 Viewing technology differences

AFP Web Viewer (ActiveX/Plug-in) has the following standard functions:

- Zoom
- Rotate
- Reset View
- Hide Images
- Copy to clipboard
- Select rectangle
- Select text
- Goto page, table of contents
- Find
- Find Next
- Navigation (begin, end, page next, page end)
- Print (All pages, current page, page range)

Both solutions (ODWEK and eClient/II for Content V8.2 Fixpack 1) now provide the same AFP Web Viewer. Additional functions are provided by the enhanced version of the AFP Web Viewer which is available only through ODWEK/II for Content (the free and unsupported version on the IBM Printing Systems Web site does not provide the additional functions):

- It maintains default OD logical view (zoom, rotate, paper width/length, image intensity, image color, background color)
- All data (including AFP resources) sent from ODWEK is compressed (up to 10:1), the plug-in decompresses the document, therefore network bandwidth is reduced as well as providing quicker document display.

  For large objects, only the first segment is sent to a plug-in. The plug-in then controls when page/segment boundaries are crossed in order to get the next segment from ODWEK/II for Content. All of this is seamless to the end-user.

11.3.5 EBCDIC or transaction/line reports

Both solutions now provide the same line data applet:

- All line data is converted to Unicode (UTF-8) and sent to the applet, so it is completely NLS enabled.
- It provides navigation (goto page, begin, end, next page, previous page)
- It maintains OD logical view (locked headers/columns)
- It offers the following capabilities:
  - Print (all pages, current page, page range)
  - Find/find next (including across large objects)
  - Annotations (view and add)
  - Change font, font size
  - Copy to clipboard (select text)
  - Copy document pages to file (all pages, current page, page range)
- All data sent from both solutions is compressed (up to 30:1), and then the applet decompresses the document; therefore, network bandwidth is reduced as well as quicker document display.
- For large objects, only the first segment is sent to the applet. The applet then controls when page/segment boundaries are crossed in order to get the next segment. All of this is seamless to the end-user.
- The line data applet caches documents so that it avoids having to go back to ODWEK/II for Content to retrieve data already sent

11.3.6 Viewing image data stored in OnDemand

To view image data stored in OnDemand, ODWEK has the following characteristics:

- It uses any existing viewing application (MIME the application type).
- It provides OnDemand Image Web Viewer (ActiveX/Plug-in).

- It uses Pixel Translations, similar to that of OnDemand Windows Client, which includes:
  - Scale
  - Rotate
  - Contrast/brightness
  - Scale to gray
  - Copy to clipboard
  - Select rectangle
  - Navigation (begin, end, page next, page end)
  - Print (All pages, current page, page range)

To view image data stored in OnDemand, eClient/Information Integrator for Content has the following characteristics:

- It uses any existing viewing application (MIME the application)
- It *does not* provide OnDemand Image Web Viewer.
- All of the various data types are passed through the Java beans. If the data stream is image, it can take advantage of the image transform beans available. The image can then be viewed with the eClient (by default the conversion is to GIF):
  - Show/hide/add annotations
  - Provides navigation (goto page, begin, end, next page, previous page)
  - Scale
  - Rotate

11.3.7 Xenos transforms

The Xenos transforms can be used to index data for loading into OnDemand as well as transforming data while retrieving it from OnDemand using ODWEK or II for Content. The following transforms are supported for data retrieval:

- AFP2PDF
- APF2HTML (template)
- AFP2XML (template)
- META2AFP
- META2PDF
- META2HTML (template)
- META2XML (template)
- LINE2PDF
- LINE2HTML (template)
- LINE2XML (template)

Note: OnDemand Large Object support is only valid for a stored data type of AFP or line data. If data is stored in OnDemand as PDF, large objects do not apply.

PCL2PDF is the Xenos transforms that is only applicable to the OnDemand loading process. Data is stored as native PDF in OnDemand. If you choose to store native PCL within OnDemand (by using the Generic Index File Format and User Defined Data Types), then this transform can be used to convert PCL to PDF under ODWEK.

Xenos transform in ODWEK includes:

- AFP2PDF, LINE2PDF, META2PDF:

  The entire document (including large objects) is converted to PDF and sent to the Adobe PDF Reader. Data is sent in its entirety, there is no concept of compression.

- META2AFP:

  The entire document is converted to AFP and sent to the AFP Plug-in; there is no compression, and no default OD logical view.

- AFP2HTML, AFP2XML, LINE2HTML, LINE2XML:

  The entire document (including large objects) is converted to a template driven HTML or XML and sent to the browser.

- META2HTML, META2XML:

  The entire document is converted to a template driven HTML or XML and sent to the browser.

Xenos transforms with eClient/II for Content includes the following capabilities:

- Any Xenos transform can be converted in the II for Content document conversion engine. This code is provided directly from Xenos.
- The resulting output will be viewed by the mime-type associated viewer (that is, PDF to Adobe PDF Reader).

11.3.8 IBM printing systems AFP2WEB transforms

AFP2WEB transforms are an optional services offering that can be used for *retrieval only*. They do not provide any supported loading transforms.

AFP2PDF works with ODWEK and eClient/II for Content:

- Both solutions do exactly the same thing:

  - The entire document (including large objects) is converted to PDF and sent to the Adobe PDF Reader.
  - Data is sent in its entirety, there is no concept of compression.
- For eClient/II for Content, the document conversion engine is used.

AFP2HTML applet works with ODWEK as follows:

- It provides navigation (goto page, begin, end, next page, previous page).
- It offers print (handled by the browser print button).
- It supports annotations (view and add).
- It has a return to hit-list button (the browser Back button is not sufficient).
- All data sent from ODWEK is compressed (up to 10:1), applet decompresses the document, therefore network bandwidth is reduced as well as quicker display.
- For large objects, only the first segment is sent to the applet. The applet then controls when page/segment boundaries are crossed in order to get the next segment from ODWEK. All of this is seamless to the end-user.
- Applet caches document so that it avoids having to go back to ODWEK to retrieve data already sent

eClient/II for Content *restrictions* include:

- OD AFP2HTML applet does not work in this environment.
- It uses the II for Content document conversion engine.
- The browser provides the viewer.
- Print and View Annotations are restricted.
- Only a single page at a time is delivered to the browser.

  When subsequent pages are requested, the viewer calls II for Content, which calls AFP2HTML to convert the needed page and sends it to the browser.
- It provides navigation (goto page, begin, end, next page, previous page).

  This action must be performed by II for Content; therefore, the page to be retrieved is sent/resent to the browser.

11.4 Comparing OnDemand clients

Table 11-2 compares the clients in logon, folder selection, document searching, document hit list, document viewing, AFP document viewing, line data report viewing, and image viewing.

Table 11-2 OnDemand clients - functionalities comparison

	OD Client	ODWEK	e-Client
Logon			
Server Name	List	Default	List
User Id, Password	Type in	Type in	Type in
Logon action	OK button or Enter Key	Submit button	Logon button
Reset button to clear all field	No	Yes	Yes
Update Servers list button	Yes	No	No
Help button	Yes	No	Yes
Change Password button	No	No	Yes
Folder Selection			
Open Folder	Open button or Enter Key	Open button	OK button
Search through folder list	Yes	No	No
Help button	Yes	No	No
Search for a document			
Customizable toolbar	Yes	No	No
Customizable Shortcut	Yes	No	No
Customizable Font	Yes	No	No
Move between Open Folders	Yes	No	No
Advanced Search pull down	Yes	No	Yes
Search Annotation text	Yes	No	Yes
Logical AND / OR	Yes	No	No
Append to Hit List	Yes	No	No
Save, Restore Search query	Yes	No	No
Document Hit List			
View Document location	Yes	No	No
View if Annotation exist	Yes	No	No
View, Add Annotations	No	Yes	Yes

	OD Client	ODWEK	e-Client
Viewing Document			
AFP document	Yes	Plug-in	Plug-in
Line Data report	Yes	Applet	Applet
Image	Yes	Plug-in	Browser
PDF document	Yes- OLE requires Adobe (Approver or Acrobat). Non OLE can use Adobe Reader	Yes-Adobe Reader	Yes-Adobe Reader
With Acrobat Readers	Separate window	Plug-in	Plug-in
AFP2HTML	n/a	Applet	Browser
AFP2PDF	n/a	Plug-in	Plug-in
Xenos transform	n/a	Yes	No
View an AFP Document			
Plug-in	n/a	Yes	Yes
Copy text or image to clipboard	Yes	Yes	Yes
Find/Find next/Goto	Yes	Yes	Yes
Zoom/Rotate/Print	Yes	Yes	Yes
Turn off image (logo)	Yes	Yes	Yes
Next/Previous Page	Yes	Yes	Yes
Reset view	Yes	Yes	Yes
Add/view note annotation	Yes	Yes	Yes
Graphical annotation	Yes	No	No
Logical views	Yes	Yes	No
Support compress data	Yes	Yes	Yes
Support large object	Yes	Yes	No

	OD Client	ODWEK	e-Client
View a Line data report			
		Applet	Convert to ASCII
Create/View Annotation	Yes	Yes	No
Find	Yes	Yes	No
Go to / Print	Yes	Yes	No
Entire Report is a single page	No	No	Yes
Full navigation	Yes	By Applet	No
Copy to Clipboard	Yes	Yes	Yes
Copy to File	Yes	Yes	No
Support compress data	Yes	Yes	No
Support large object	Yes	Yes	No
View an Image BMP, GIF, JPEG, PCX, TIFF			
Plug-in	n/a	Yes	No
Data types passed thru Java beans	n/a	n/a	Yes
Zoom	Yes	Plug-in	By Server
Rotate	Yes	Plug-in	By Server
Image brightness	Yes	Yes	No
Image Scale to Grey	Yes	Yes	No
First/Last/Next/Previous Page	Yes	Plug-in	By Server
Copy to clipboard	Yes	Yes	No
Print	Yes	Yes	1 page only
Show/Hide Annotation	Yes	No	Yes

So which one should I choose?

If you need a *federated search* in real time across multiple repositories, you should consider eClient/Information Integrator for Content, since ODWEK does not have such a client.

If you currently have Content Manager or plan to implement a Content Manager system, we recommend using Information Integrator for Content.

For an existing OnDemand customer who simply needs access to the OnDemand data streams, ODWEK provides the most comprehensive platform support and provides native access (non-federated) support to OnDemand servers.

Although ODWEK does not supply an out-of-box “thin client”, it does provide sample code for setting up CGI or servlet Web sites and Java APIs for developing your own Java client or program to program interface. If you currently only have OnDemand, have no plan of using or working with another repository, have no plan of using Content Manager, and have only CGI skills, your choice would be using the CGI version of ODWEK to access OnDemand. If you have Java developers and will only need to manage OnDemand data, you can choose ODWEK.

12

ODWEK installation and configuration

This chapter provides updated installation instructions for installing ODWEK for CGI, servlets, and Java APIs on a Windows platform. We start out with a simple installation of CGI to get you used to ODWEK. Then we build up to the servlet and install a sample ODWEK Java client. We finish by showing you a simple method to have all three components installed for testing purposes.

We cover the following topics:

- Installing CGI ODWEK on Windows
- Line data applets
- Installing the ODWEK servlet on Windows

12.1 Installing CGI ODWEK on Windows

The simplest implementation of ODWEK is through Common Gateway Interface (CGI). This is because the CGI setup does not require a Java application server; only an HTTP server is needed (see Figure 12-1). If you are new to ODWEK, a quick way to get started is to implement ODWEK using the CGI Application Programming Interface (APIs) and HTTP. This simple implementation can help you understand the structure of ODWEK and see how all of the parts interact. We go through all the API sets in the following sections from A-Z as if you are new to the Web and ODWEK. If you are familiar with some of the areas, please jump ahead to the next section.

CGI is a way of allowing users to be “interactive” on the Web. Usually CGI scripts are written in some programming language such as Perl. In this section, we show some ODWEK supplied CGI scripts which allow users to be interactive with OnDemand over the Web.

The *Hypertext Transfer Protocol* (HTTP) is an application-level protocol for distributed, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for tasks, such as a Web server.

A feature of HTTP is the display and negotiation of data. It allows systems to be built independently of the data being transferred. HTTP has been in use by the World-Wide Web initiative since 1990.

These are the software requirements for CGI implementation of ODWEK:

- Windows 2000 Server
- IBM HTTP Server Version 1.3 or higher (We use Version 1.3.28)
- ODWEK Version 7.1.1.1
- OnDemand Version 7.1.1.1
- JRE Version 1.3.1 or higher (we use Version 1.4.2)
- Supported browsers:
  - Netscape Navigator 4.06 or higher
  - Microsoft Internet Explorer 4.01 or higher

Here are the assumptions for our scenario:

- OnDemand is installed on a server called *ODServer1*.
- OnDemand server is called *wishart*.
- HTTP is installed on a server called *congo*.
- A line data report is loaded in OnDemand.
- A valid user ID is created for OnDemand.
- D:\ drive is used for installation.

Figure 12-1 shows the ODWEK implementation using CGI in our scenario. Note that we refer to the specific system names and directories in the remainder of this section for easier presentation. Substitute these values with your system specific setup.

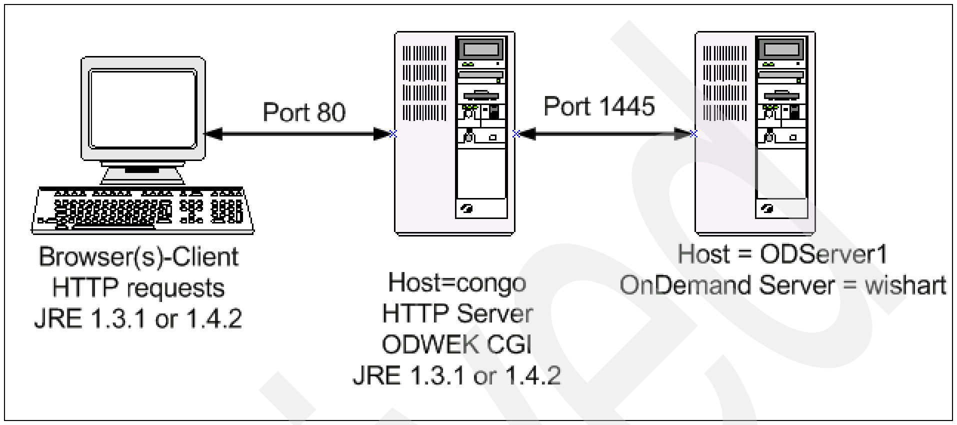


Figure 12-1 ODWEK implementation using CGI

There are several steps needed to install CGI ODWEK. We summarize them as follows:

1. Install Java Runtime Environment (JRE).
2. Install IBM HTTP Server.
3. Install ODWEK.
4. Install ODWEK modifications and Program Temporary Fixes (PTF).
5. Customize ODWEK and HTTP.
6. Test ODWEK with one or more browsers.
7. Troubleshoot.
8. Test CGI.

In the following sections, we describe each step in detail.

12.1.1 Installing Java Runtime Environment (JRE)

In this section, we describe the step-by-step details of installing the JRE.

If you have both JRE Version 1.3.1 (or higher) *and* IBM HTTP server Version 1.3 (or higher), you can go directly to 12.1.3, "Installing ODWEK base code" on page 396.

If you already have JRE Version 1.3.1 or higher, but do not have IBM HTTP server installed or have a lower version of the HTTP server, proceed to the next step, 12.1.2, "Installing IBM HTTP server" on page 392.

Note that ODWEK Version 7.1 or later comes with JRE Version 1.3.1. If you plan to install ODWEK Version 7.1 or later, *and* you already have IBM HTTP server Version 1.3.1 installed, you have a choice of installing JRE manually as described in this section, or let the installation of ODWEK install JRE for you.

Note: If you do not have JRE or the right version of JRE, we highly recommend that you install it manually before proceed further.

If you want to run the latest line data applet in ODWEK which we highly recommend, you need to install JRE 1.4.1 or later.

Determining your installed Java version, if any

To determine if you have Java installed, do the following steps:

1. Open a DOS command window.
2. Enter the following command:

```
Java -fullversion
```

The level should read 1.3.1 or higher. If you get a message that Java is an invalid command, then you need to follow this section to install Java before continuing.

Downloading JRE

JRE Version 1.3.1 or higher can be downloaded from:

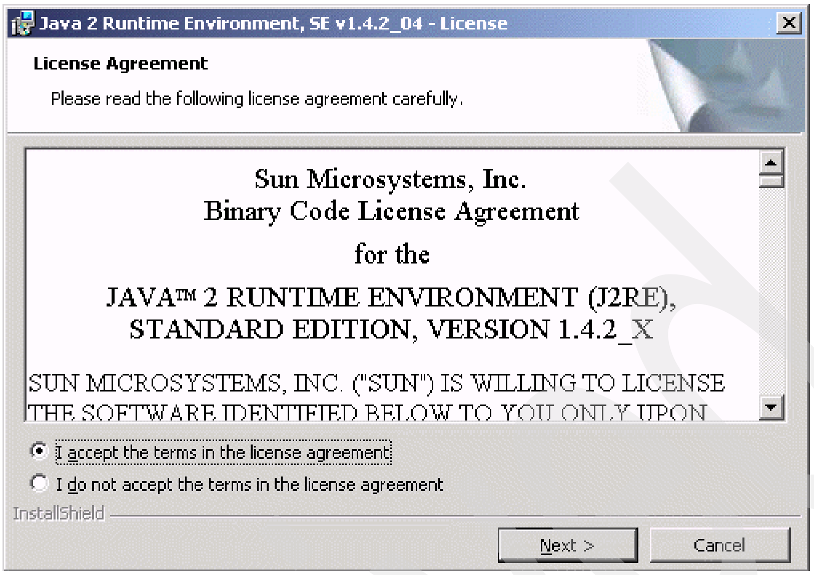
http://www.java.sun.com

In our scenario, we downloaded JRE Version 1.4.2 You can download either the Software Development Kit (SDK) or JRE.

Installing JRE

Once you have downloaded the JRE, install it on your system:

1. Double-click the downloaded executable file. The install screen followed by the license agreement screen appears, as shown in Figure 12-2.


Java 2 Runtime Environment, SE v1.4.2_04 - License

License Agreement

Please read the following license agreement carefully.

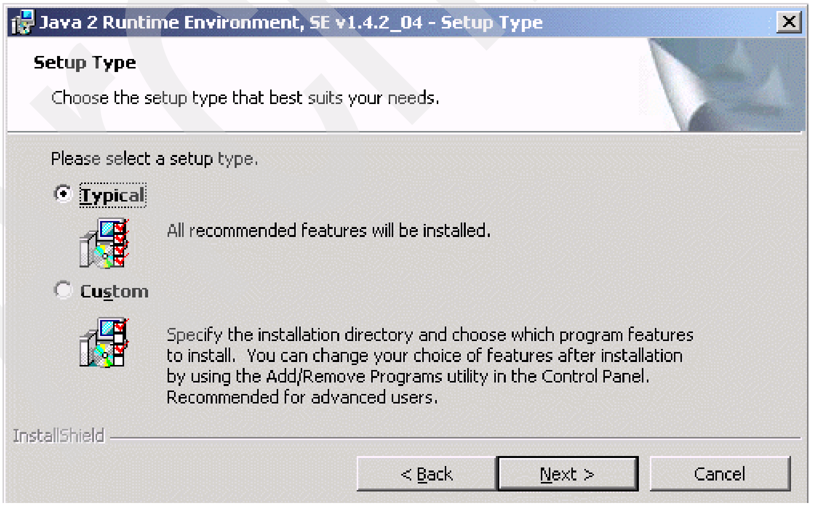
Sun Microsystems, Inc.
Binary Code License Agreement
for the
JAVA™ 2 RUNTIME ENVIRONMENT (J2RE),
STANDARD EDITION, VERSION 1.4.2_X

SUN MICROSYSTEMS, INC. ("SUN") IS WILLING TO LICENSE
THE SOFTWARE IDENTIFIED BELOW TO YOU ONLY UPON

[✓ I accept the terms in the license agreement]
I do not accept the terms in the license agreement

Figure 12-2 Installing Sun Java

2. Click **I accept the terms in the license agreement**, and then click **Next**.
3. The next screen (Figure 12-3) asks you to select setup type. Select **Typical**. If you want additional language and font support, you may optionally select Custom. Click **Next**.


Java 2 Runtime Environment, SE v1.4.2_04 - Setup Type

Setup Type

Choose the setup type that best suits your needs.

Please select a setup type.

[✓ Typical]
All recommended features will be installed.

Custom
Specify the installation directory and choose which program features to install. You can change your choice of features after installation by using the Add/Remove Programs utility in the Control Panel. Recommended for advanced users.

Figure 12-3 Java setup screen

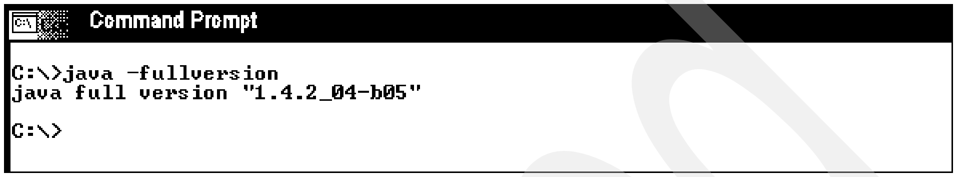
4. Click **Finish** when the installation is completed.

Verifying the installation

To verify your JRE installation:

1. Open a DOS command window.
2. Enter **`java -fullversion`**.

You should see the version of JRE you have just installed (see Figure 12-4).



```
C:\>java -fullversion
java full version "1.4.2_04-b05"

C:\>
```

Figure 12-4 testing Java version

12.1.2 Installing IBM HTTP server

In this section, we give you a step-by-step procedure for installing IBM HTTP server on your system.

If you already have IBM HTTP server Version 1.3 or higher installed on your system, proceed to the next step, 12.1.3, "Installing ODWEK base code" on page 396.

If you do not have JRE Version 1.3.1 or higher installed on your machine, go back to the previous step, 12.1.1, "Installing Java Runtime Environment (JRE)" on page 390.

ODWEK requires an HTTP server. You can install the IBM HTTP as presented in our scenario, or you can include the HTTP server as part of the WebSphere installation (see 12.3.2, "Installing WebSphere Application Server 5.0 for Windows" on page 424).

Downloading IBM HTTP server software

IBM HTTP server Version 1.3 or later software can be downloaded from:

`http://www.ibm.com/software/webservers/httpservers`

Installing IBM HTTP server

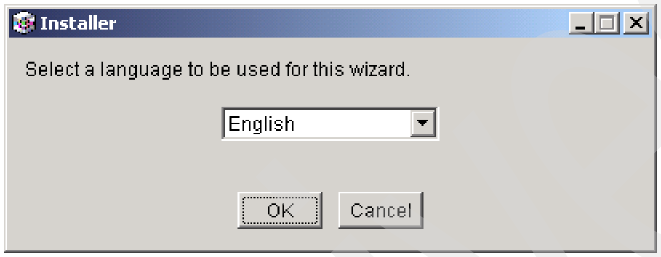
Once you have downloaded the IBM HTTP server software, install it on your system:

1. Launch the downloaded software:
   a. Create a temporary directory and save the HTTP server software in this new directory. In our scenario, we use `D:\HTTPServer`.

   b. Rename the downloaded file to have only a zip extension. The downloaded file name will be similar to HTTPServer.win.1382.zip.bqy. Rename it to HTTPServer.win.1382.zip.
   c. Unzip the file to the temporary directory using a Zip tool.
   d. Change the directory to where you unzipped the file, and issue the following command from a DOS command window:

      ```
      Java -jar setup.jar
      ```

2. After a short time, you will see the Select a language screen (see Figure 12-5). Click **OK** to accept English (or choose language of preference).


Installer

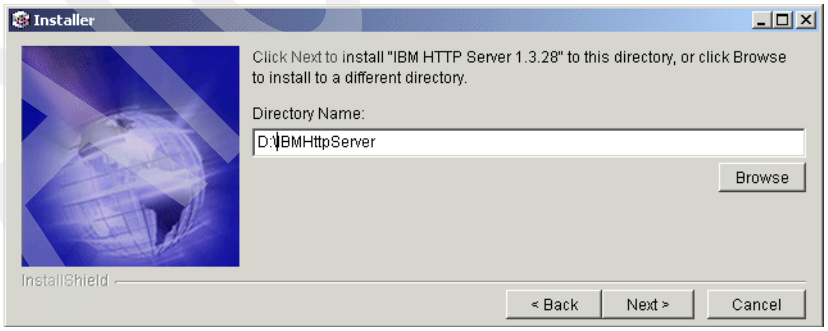
Select a language to be used for this wizard.

English

Figure 12-5 HTTP, selecting language

3. On the Welcome screen, click **Next**.
4. On the License Agreement screen, click **I accept the terms of the license agreement** and then click **Next**.
5. On the next screen (Figure 12-6), enter `D:\IBMHTTPServer` or `C:\IBMHTTPServer` instead of the default install directory. Click **Next**.

   The short directory makes working with the HTTP server easier.


Installer

Click Next to install "IBM HTTP Server 1.3.28" to this directory, or click Browse to install to a different directory.

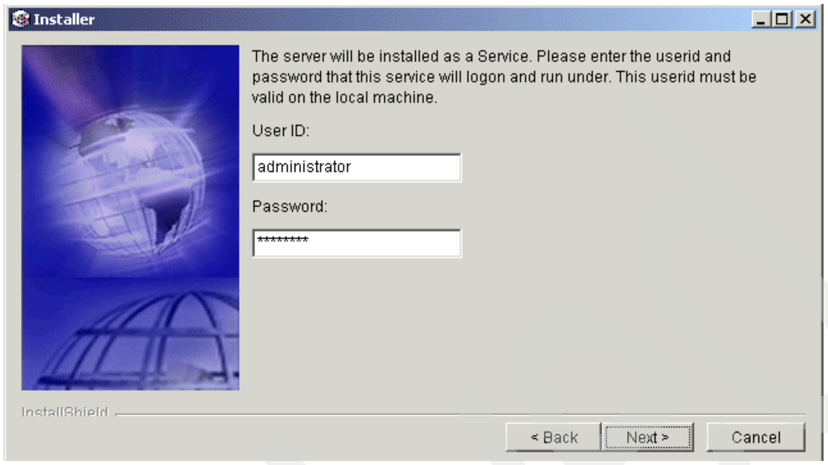
Directory Name:

D:\IBMHttpServer

Figure 12-6 HTTP Install directory

6. On the Choose the setup type screen, select **Typical**. Click **Next**.

7. Enter the user ID and password (see Figure 12-7). Make sure this is a valid user ID in your system. Click **Next**.


Installer

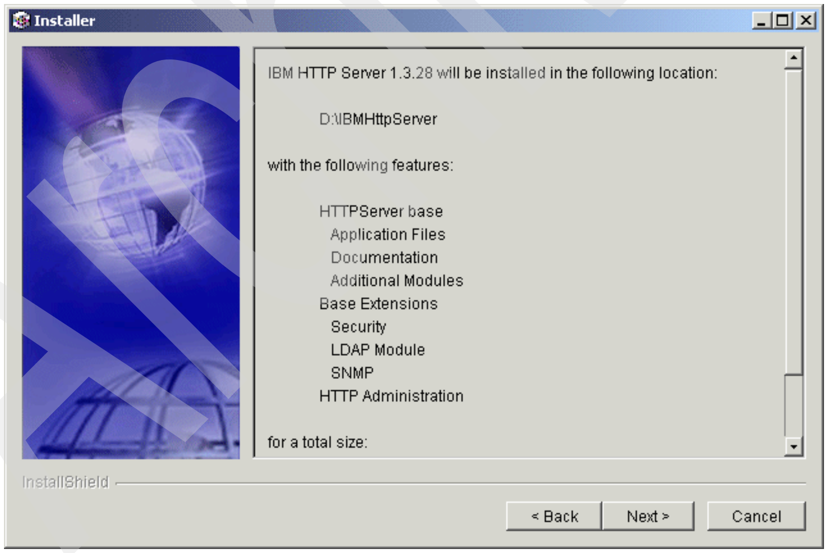
The server will be installed as a Service. Please enter the userid and password that this service will logon and run under. This userid must be valid on the local machine.

User ID: administrator

Password: *********

Figure 12-7 HTTP user ID and password screen

8. The HTTP install program lists all the directories and features you have selected (see Figure 12-8). If all the input is correct, click **Next**.


Installer

IBM HTTP Server 1.3.28 will be installed in the following location:

D:\IBMHttpServer

with the following features:

HTTPServer base
Application Files
Documentation
Additional Modules
Base Extensions
Security
LDAP Module
SNMP
HTTP Administration

for a total size:

Figure 12-8 HTTP install choices

9. After the installation is complete, click **Finish**.

Verifying installation

To verify your IBM HTTP server installation:

1. Open a Web browser.
2. Enter the following URL:

   `http://localhost`

   If you installed the IBM HTTP server on one machine (in our scenario, congo), and you launched the browser in another machine, enter:

   `http://congo`

   Substitute `congo` with whatever machine name you installed the HTTP server on. `Localhost` only works if you launched the browser on the same machine where you installed your HTTP server.

A Welcome to IBM HTTP Server screen should appear (see Figure 12-9).

If you get an invalid page message, go to the Windows Services menu and ensure that the HTTP server is running. If it is not running, start it manually. Make sure it is set to start automatically. If you still have problems, check the `IBMHTTPServer\Logs\error.log` file for details.

Important: If you have Microsoft IIS running, this may cause a port conflict. Stop the IIS service or correct port numbers so they do not conflict.

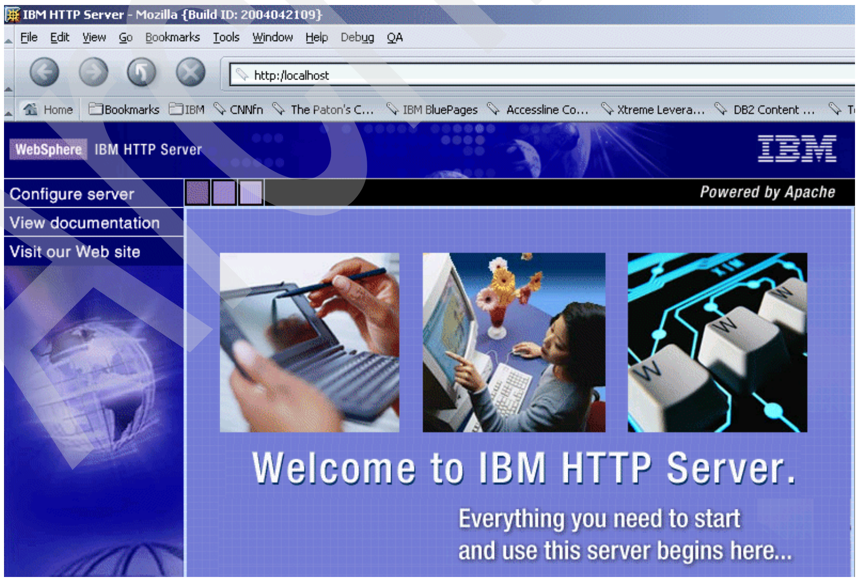

IBM HTTP Server - Mozilla {Build ID: 2004042109}
File Edit View Go Bookmarks Tools Window Help Debug QA
http://localhost
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Content ...
WebSphere IBM HTTP Server
IBM
Configure server
View documentation
Visit our Web site
Powered by Apache
Welcome to IBM HTTP Server.
Everything you need to start and use this server begins here...

Figure 12-9 Welcome to IBM HTTP Server screen

12.1.3 Installing ODWEK base code

Good documentation for installing ODWEK is provided in the product manual, *Web Enablement Kit Implementation Guide*, SG27-1000. We recommend that you read it.

Note: IBM delivers software in four (4) levels: versions, releases, modifications, and fixes, also known as PTFs (Program Temporary Fixes). At the time of writing this redbook, OnDemand is at Version 7, Release 1, Modification 1, and PTF 1, or ODWEK Version 7.1.1.1

In order to get to ODWEK Version 7.1.1.1 you must install the base ODWEK CD Version 7.1.0.0 or the newer base code ODWEK CD Version 7.1.1.0.

If your are installing from the ODWEK CD Version 7.1.0.0 (the older code base), you need to install ODWEK in the following order:

1. ODWEK Version 7.1.0.0 base code
2. ODWEK Version 7.1.1.0 modification level
3. ODWEK Version 7.1.1.1 PTF

In this section, we start with installation of ODWEK Version 7.1.0.0 base code.

If you are installing from the (newer base) ODWEK CD Version 7.1.1.0, you need to install ODWEK in the following order:

1. ODWEK Version 7.1.1.0 base code
2. ODWEK Version 7.1.1.1 PTF

See "Installing ODWEK Version 7.1.1.0 base code" on page 398.

Downloading OnDemand and ODWEK fixes

OnDemand and ODWEK fixes can be downloaded from:

`ftp://service.software.ibm.com/software/ondemand/fixes/`

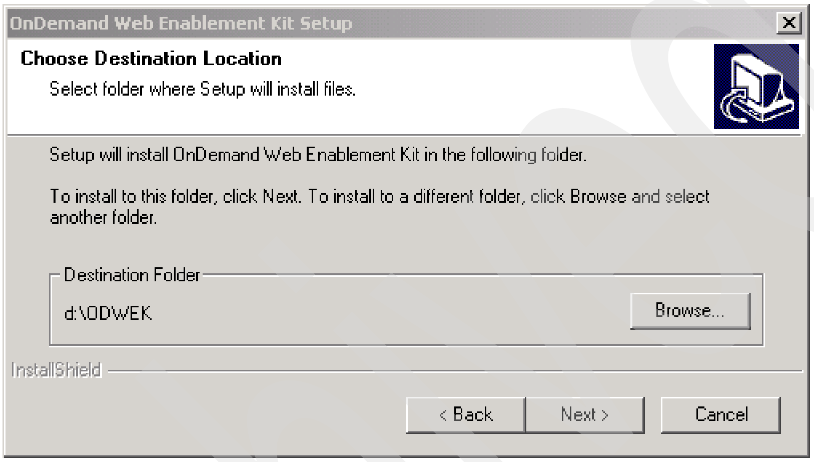
Installing ODWEK Version 7.1.0.0 base code

To install ODWEK Version 7.1.0.0 base code:

1. Gather all the code needed to do a complete ODWEK install. This includes base code and any needed fixes. Download the fixes to a temporary directory on your system (Example: C:\ODFixes).

2. Place the base code CD in your CD ROM drive. Change to that drive. Go to the WEK\Windows sub-directory.

3. Double-click **setup.exe**.
4. On the Welcome screen, click **Next**.
5. On the License Agreement, click **I accept** and then click **Next**.
6. On the next screen (Figure 12-10), change the default directory (C:\Program Files\IBM\OnDemand Web Enablement Kit) to a simpler directory. We use D:\ODWEK. This can be done by clicking **Browse** and enter `D:\ODWEK` or `C:\ODWEK`. Click **Next** to continue.


OnDemand Web Enablement Kit Setup

Choose Destination Location

Select folder where Setup will install files.

Setup will install OnDemand Web Enablement Kit in the following folder.

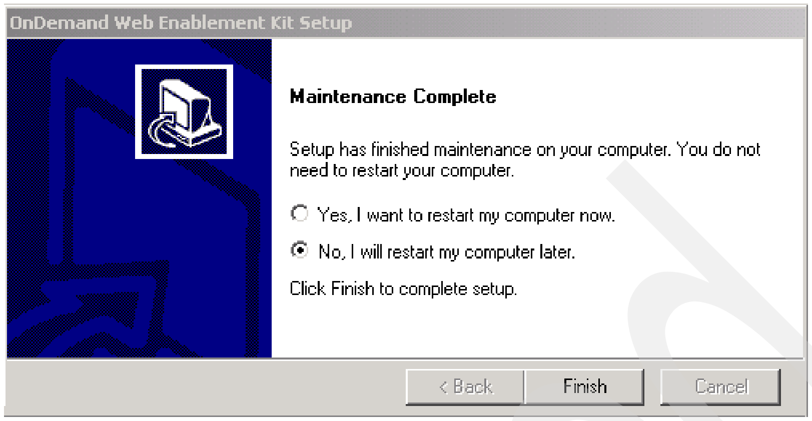
To install to this folder, click Next. To install to a different folder, click Browse and select another folder.

Destination Folder

d:\ODWEK

Figure 12-10 ODWEK Choose Destination Location screen

7. Click **Next** to confirm the setup. The system will begin installing ODWEK.
8. Uncheck **View Readme file**, and click **Finish** to complete the installation.
9. On the Maintenance Complete screen (Figure 12-11), click **No, I will restart my computer later**, and then click **Finish**.


OnDemand Web Enablement Kit Setup

Maintenance Complete

Setup has finished maintenance on your computer. You do not need to restart your computer.

○ Yes, I want to restart my computer now.

[✓ No, I will restart my computer later.]

Click Finish to complete setup.

Figure 12-11 ODWEK base code complete screen

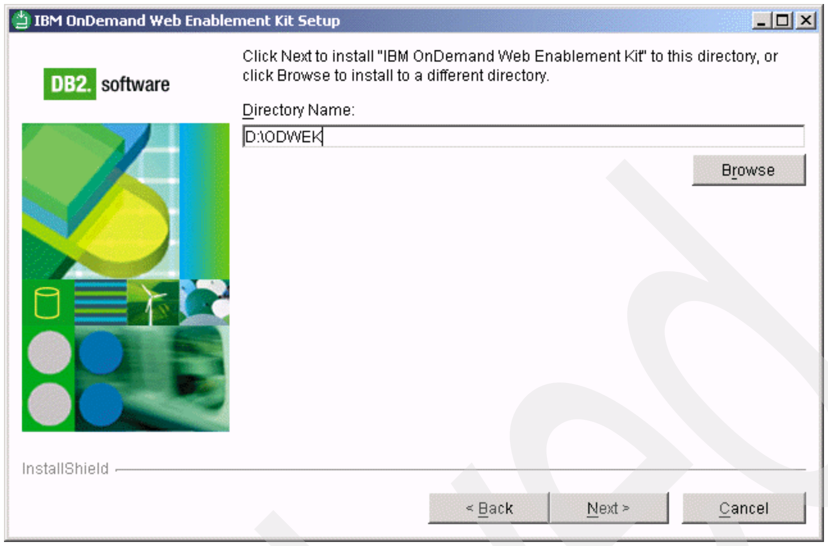
This completes the installation of the ODWEK Version 7.1.0.0 base code.

To continue the installation of modification and PTFs, proceed to 12.1.4, “Installing ODWEK releases, modifications, and PTFs” on page 400.

Installing ODWEK Version 7.1.1.0 base code

To install ODWEK Version 7.1.1.0 base code:

1. Gather all the code needed to do a complete ODWEK install. This includes base code and any needed fixes. Download the fixes to a temporary directory on your system (Example: C:\ODFixes).
2. Place the base code CD in your CD ROM drive. Change to that drive. Go to the WEK\Windows sub-directory.
3. Double-click **setup.exe**.
4. On the Welcome screen, click **Next**.
5. On the License Agreement screen, click **I accept** and then click **Next**.
6. On the next screen (Figure 12-12), enter `D:\ODWEK` or `C:\ODWEK`. Click **Next**.


IBM OnDemand Web Enablement Kit Setup

Click Next to install "IBM OnDemand Web Enablement Kit" to this directory, or click Browse to install to a different directory.

Directory Name:

D:\ODWEK

Figure 12-12 ODWEK installation directory screen

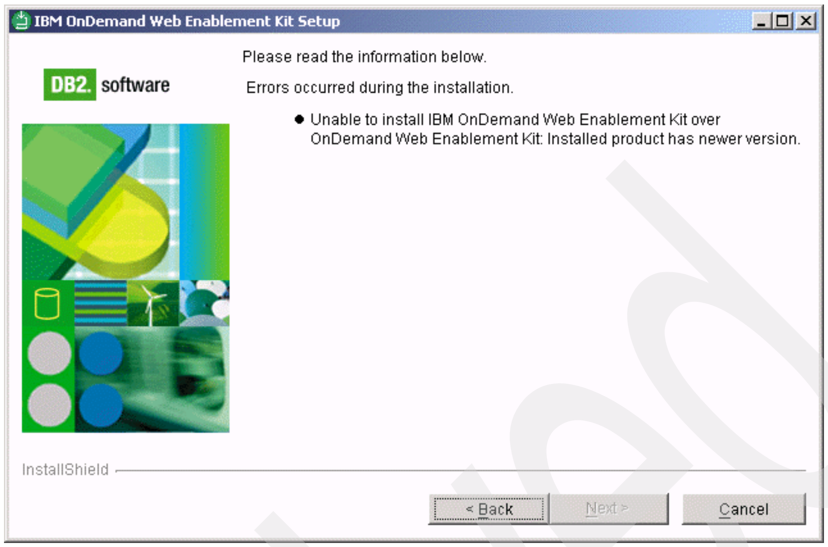
7. Click **Next** to confirm the setup. The system will begin installing ODWEK.
8. Uncheck **Display product ReadMe file**, and click **Next**.
9. Click **Finish** when the installation is done.

This completes the installation of the ODWEK Version 7.1.1.0 base code.

Attention: ODWEK's uninstall program may not clean up your system completely. If you uninstall ODWEK (because the setup was not done correctly or you just want to repeat your installation) and then try to re-install it on the same non-default directory (such as D:\ODWEK), you may encounter an error message as shown in Figure 12-13.

To resolve this issue, you can either install ODWEK in a new directory, or use `regedit` to search for any references of the ODWEK and delete them before re-installing ODWEK on the same directory.

Caution: You must be familiar with `regedit` to use it to clean up your system. Manually cleaning the registry setting without sufficient knowledge on how the registry works can be dangerous!



Please read the information below.

Errors occurred during the installation.

- Unable to install IBM OnDemand Web Enablement Kit over OnDemand Web Enablement Kit: Installed product has newer version.

Figure 12-13 ODWEK install error message

Proceed to install ODWEK Version 7.1.1.1 PTF as described in the next section.

12.1.4 Installing ODWEK releases, modifications, and PTFs

In this section, we describe the step-by-step procedures for installing ODWEK releases, modifications, and PTFs.

Downloading ODWEK releases, modifications, and PTFs

ODWEK releases, modifications, and PTFs can be downloaded from:

`ftp://service.software.ibm.com/software/ondemand/fixes/`

Installing modification

If you installed ODWEK Version 7.1.0.0 base code, you need to install ODWEK Version 7.1.1.0 modification level. If you installed ODWEK Version 7.1.1.0 base code, go to "Installing PTF" on page 401.

To install an ODWEK modification:

1. Change to the directory where you downloaded the modification maintenance code (for example, C:\OD7110). Double-click **odwekwin.exe** file.
2. On the Welcome screen, click **Next**.
3. On the License Agreement screen, click **I agree**, and then click **Next**.

4. On the confirmation screen, click **Next** to confirm the setting. The system will start installing the modification.
5. On the next screen, click **No** for reviewing Redme.txt file, and then click **Next**.
6. Click **Finish** to complete the installation.

You have now completed the installation for ODWEK 7.1 and 7.1.1.

Installing PTF

In this section, we describe how to add any PTFs when you need to upgrade the PTF level. In our scenario, we need to install ODWEK Version 7.1.1.1 PTF.

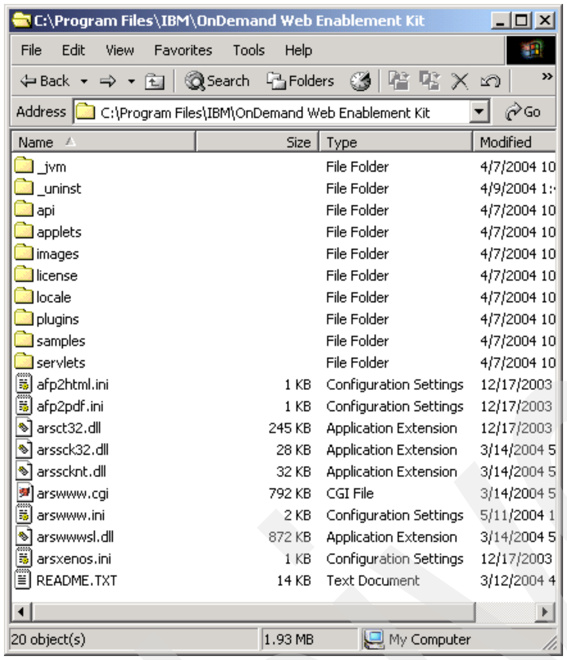
To install PTF:

1. Change to the directory where you downloaded the PTF code. Double-click **odwekwin.exe**.
2. Follow the same steps as given for installing the base code and the modification level until you have completed the installation of the PTF level.

Now your system is at ODWEK Version 7.1.1.1.

You can re-boot your server now.

When you have complete the installation of ODWEK and ODWEK Maintenance, your ODWEK directory should look similar to Figure 12-14.


C:\Program Files\IBM\OnDemand Web Enablement Kit
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
C:\Program Files\IBM\OnDemand Web Enablement Kit
Go
20 object(s)
1.93 MB
My Computer

Name	Size	Type	Modified
_jvm		File Folder	4/7/2004 10
_uninst		File Folder	4/9/2004 1:
api		File Folder	4/7/2004 10
applets		File Folder	4/7/2004 10
images		File Folder	4/7/2004 10
license		File Folder	4/7/2004 10
locale		File Folder	4/7/2004 10
plugins		File Folder	4/7/2004 10
samples		File Folder	4/7/2004 10
servlets		File Folder	4/7/2004 10
afp2html.ini	1 KB	Configuration Settings	12/17/2003
afp2pdf.ini	1 KB	Configuration Settings	12/17/2003
arsct32.dll	245 KB	Application Extension	12/17/2003
arssck32.dll	28 KB	Application Extension	3/14/2004 5
arsscknt.dll	32 KB	Application Extension	3/14/2004 5
arswww.cgi	792 KB	CGI File	3/14/2004 5
arswww.ini	2 KB	Configuration Settings	5/11/2004 1
arswwwsl.dll	872 KB	Application Extension	3/14/2004 5
arsxenos.ini	1 KB	Configuration Settings	12/17/2003
README.TXT	14 KB	Text Document	3/12/2004 4

Figure 12-14 ODWEK installed files

ODWEK directories explanation

Table 12-1 lists all the ODWEK directories and their descriptions.

Table 12-1 ODWEK directories - explanation

Directory	**Description**
_jvm	Contains the Java Virtual Machine (JRE code)
_uninst	Contains ODWEK uninstallation code
api	Holds the Java APIs and documentation
applets	Contains the line data and AFP2HTML applets
images	Contains the images for the sample HTML and icons used in the applets
license	Contains ODWEK license text files
locale	National language support directories

Directory	Description
plugins	Plug-ins for: Advanced Function Presentation (AFP) National Language Support (NLS) version AFP (US version) Image plug-in for viewing multiple image formats.
samples	Sample html (htm) code files to support the installation of the CGI or servlet. There are three samples: Logon: Contains example of most ODWEK CGI and servlet functions Credit: Contains example for simple access to Credit Card application Frames: Contain example of ODWEK using frames
servlets	Holds the ODWEK Java code (jar and class files) for the Servlet and Java client.

ODWEK files explanation

Table 12-2 lists major ODWEK files and their descriptions.

Table 12-2 ODWEK major files and their descriptions

File name	Description
afp2html.ini	Setup and configuration information for AFP to HTML transform. Requires the afp2html optional package from IBM Printing Systems.
afp2pdf.ini	Setup and configuration information for the AFP to PDF transform. Requires the optional afp2pdf package from IBM Printing Systems.
arsct32.dll arssck32.dll arsscknt.dll	ODWEK code file.
arswww.cgi	CGI script.
arswww.ini	Configuration file for CGI, servlet and Java client. Controls interface between OnDemand, ODWEK and browsers.
arswwwsl.dll	ODWEK shared library code file.
arsxenos.ini	Configuration and setup file for working with the optional Xenos transforms.
README.TXT	Contains the latest information about the ODWEK release.

12.1.5 Configuring HTTP and ODWEK

In this section we configure the HTTP server and ODWEK environment to run a CGI implementation of ODWEK.

Note: This is just one way to set up ODWEK. There are many other ways to deploy ODWEK. In a later section, to help you understand ODWEK better, we show another way that enables you to have the CGI, servlet, and Java client of ODWEK deployed.

Let us get started deploying the CGI ODWEK. Here is a summary of the steps we need to follow to deploy the CGI ODWEK:

1. Copy files to the HTTP server.
2. Edit the HTTP httpd.conf file.
3. Edit the ODWEK arswww.ini file.
4. Edit ODWEK HTM logon file.
5. Set up environment path.
6. Test the logon.htm.
7. Troubleshoot if necessary.

We assume the following system setup:

- Two physical servers:
  - *Congo* is where ODWEK and the HTTP server are installed and run.
  - *Wishart* is where the OnDemand system is installed and run.

  Can you install everything in one physical machine? Yes; however, having everything on one physical server may impact performance. We *strongly* discourage this configuration.
- ODWEK is installed at D:\ODWEK on congo.
- HTTP server is installed at D:\IBMHTTPServer on congo.
- OnDemand is installed on a host server called ODServer1.
- The OnDemand host server is call wishart.
- JRE, HTTP, ODWEK are properly installed.

In this section, we refer to the specific directories and machines as mentioned above. Substitute the instruction with the actual server names and directories in your environment.

Copying files to HTTP server

To copy files to HTTP server:

1. Copy the following files from D:\ODWEK to D:\IBMHTTPServer\cgi-bin:

   ```
   arswww.cgi
   arswww.ini
   afp2html.ini
   afp2pdf.ini
   arsxenos.ini
   arsct32.dll
   arsscknt.dll
   ```

   Substitute the D:\ODWEK directory with the directory where you installed ODWEK. Substitute D:\IBMHTTPServer with the directory where you installed IBM HTTP server.

   Note: The arswww.cgi file is a CGI script that needs to be in a directory where HTTP server can see it. By default, the IBM HTTP server looks in its cgi-bin directory for scripts. This is why we place the arswww.cgi file in the D:\IBMHTTPServer\cgi-bin directory.

2. Create the necessary subdirectories under D:\IBMHttpServer\cgi-bin:

   ```
   applets
   images
   templates
   MyODWEK
   tmp
   tmp\cache
   ```

3. Create the following subdirectory under D:\IBMHTTPServer\htdocs\en_US:

   ```
   MyODWEK
   ```

   Your directory structure should look similar to Figure 12-15.

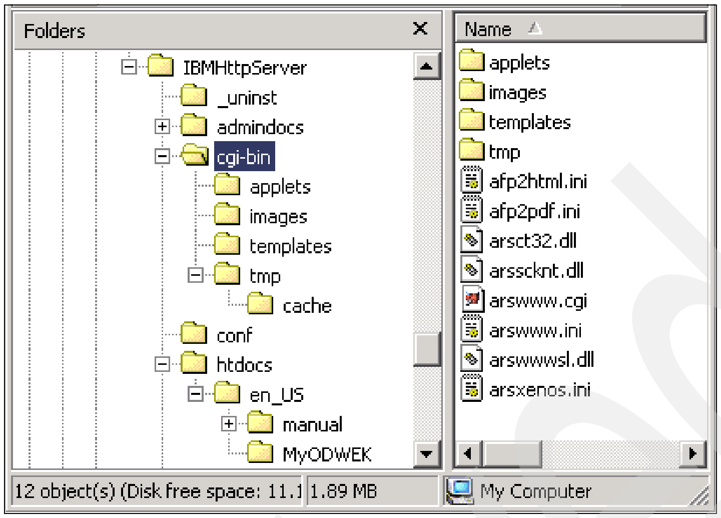


Figure 12-15 Directory structure of CGI ODWEK

4. Copy the files from the ODWEK subdirectories to HTTP subdirectories:
   - Copy D:\ODWEK\applets\*.* to D:\IBMHTTPServer\cgi-bin\applets
   - Copy D:\ODWEK\images\*.* to D:\IBMHTTPServer\cgi-bin\images
   - Copy D:\ODWEK\samples\*.* to D:\IBMHTTPServer\cgi-bin\templates
   - Copy D:\ODWEK\samples\logon.htm to D:\IBMHTTPServer\htdocs\en_US\MyODWEK.

Important: Remember to copy your files to the correct drive. In our scenario, we use the D:\drive. However, you might use the C:\ drive in your setup.

Editing ODWEK arswww.ini

The ODWEK arswww.ini file holds all the setup communication parameters for ODWEK, OnDemand and the browsers.

To edit the file:

1. Go to the directory, D:\IBMHttpServer\cgi-bin.
2. Copy arswww.ini to arswww.ini.backup. It is important to back up the file before you modify it!
3. Open arswww.ini with a text editor such as Notepad or Wordpad.

4. Modify the arswww.ini file. Example 12-1 shows the original arswww.ini file. Update the following three lines as reflected in Example 12-2 in bold text:

```
[@SRV@_gunnar]
HOST=gunnar
TemplateDir=/home/httpd/docs/templates
```

Example 12-1 Original arswww.ini file

```
[@SRV@_gunnar]
HOST=gunnar
PROTOCOL=0
PORT=1445
.
.
[configuration]
TemplateDir=/home/httpd/docs/templates
AppletDir=/applets
TempDir=/tmp
CacheDir=/tmp/cache
```

Example 12-2 Edited arswww.ini file

```
[@SRV@_wishart]
HOST=ODServer1
PROTOCOL=0
PORT=1445
.
.
[configuration]
TemplateDir=/templates
AppletDir=/applets
TempDir=/tmp
CacheDir=/tmp/cache
```

Note: ODServer1 is the name or IP address of the OnDemand host server, and Wishart is the name of the OnDemand server in our scenario. Substitute these names with the servers in your environment.

5. Save the changes in the arswww.ini file.

Updating httpd.conf

To update the httpd.conf file:

1. Go to the D:\IBMHttpServer\conf directory.
2. Copy httpd.conf to httpd.conf.backup. It is important to back up the file before you modify it!
3. Open httpd.conf for editing using a text editor such as Notepad or Wordpad.
4. Locate the following line in the httpd.conf using **Edit** → **Find** or something similar in your text editor tool:

   ```
   Alias /icons/ "D:/IBMHttpServer/icons/"
   ```

   Add the following lines immediately after the above line:

   ```
   Alias /images/ "D:/IBMHttpServer/cgi-bin/images/"
   Alias /templates/ "D:/IBMHttpServer/cgi-bin/templates/"
   Alias /applets/ "D:/IBMHttpServer/cgi-bin/applets/"
   ```

Tips:

- We use all forward slashes "/" in the httpd.conf file. If you are working in the Windows environment, you can use back slashes also. ODWEK works with other platforms such as AIX, HP, Solaris and Linux. It is safer to use all forward slashes "/" since this works with any platform.
- It is very important when defining an Alias to make sure your statement ends with a forward slash "/" or it will not work.
- Anytime you make changes to the httpd.conf file, you must *stop* and *re-start* the HTTP Server to have the change take effect!

5. While you are still in the httpd.conf file, search through the file until you find the `documentroot statement`. It may appear in the file multiple times. The statement says:

   ```
   DocumentRoot "D:\IBMHttpServer/htdocs/en_US"
   ```

   This is what the IBM HTTP server uses as its root directory. Every file it looks for is relative to this location. This directory is important because this is where we place the logon.htm file.
6. Save the httpd.conf file.
7. Go to the Windows Services menu. Stop and restart the HTTP Server.

Editing ODWEK logon.htm

The logon.htm file is the file that HTTP displays first and points to the CGI code, arswww.cgi, which picks up the arswww.ini parameters we defined earlier.

The sample logon.htm files are provided with ODWEK for you to use and change to suit your needs, or you can create your own HTML files later. ODWEK also has other sample files which we do not cover here. If you need more information on these files, please refer to the *Web Enablement Kit Implementation Guide*, SG27-1000.

To edit the ODWEK logon file:

1. Change to the D:\IBMHttpServer\htdocs\en_US\MyODWEK directory where you copied the logon.htm file.
2. Copy logon.htm to logon.htm.backup.
3. Open logon.htm using a text editor such as Notepad or Textpad.
4. Change the following line

   ```
   <FORM METHOD=POST ACTION="/scripts/arswww.cgi">
   ```

   To something similar to the following lines:

   ```
   <FORM METHOD=POST ACTION="../../../cgi-bin/arswww.cgi">
   ```

Note: The FORM line has a METHOD and ACTION, which can be considered as "how" and "where". The METHOD is either a get or a post. A get tacks the information onto the URL and a post sends it separately. Each method has its pros and cons. For most of the part, you will want to use post. The ACTION points to "where" the arswww.cgi file is located. If you installed the IBM HTTP server in a different directory, then you may have to change this line to point to the location where your cgi-bin directory is installed.

   If we did not change the FORM line, the logon.htm file would have searched in one subdirectory down looking for a "scripts" subdirectory with the files arswww.cgi and arswww.ini. As we know, it would not have found the "scripts" directory, since we placed the arswww.cgi file in the D:\IBMHttpServer\cgi-bin directory. This is where we need to point the HTTP server to find the arswww.ini file. You need to tell the logon.htm to back up three directories higher ../../../, and then find the cgi-bin directory.

5. Change the following two lines (optional) which provide the default host name and user ID for the logon screen:

   ```
   <b>Server Name:</b> <input type=text name=_server value=everest>
   <b>User:</b>        <input type=text name=_user size=10>
   ```

   To something similar to the following lines:

   ```
   <b>Server Name:</b> <input type=text name=_server value=wishart>
   <b>User:</b>        <input type=text name=_user size=10 value=rpaton>
   ```

   Note that the modified values are in bold text.

   Substitute the host name and user ID to the values in your environment.

Your logon.htm file should look similar to Example 12-3.

Example 12-3 Sample logon.htm for the CGI

```
<head>
<title>OnDemand Internet Connection</title>
</head>
<body background="/images/grytxtr4.jpg" bgcolor="#c2c2c2">
<p align="center"><img src="/images/odcol2in.gif" width="100" height="100"></p>
<h1 align="center">OnDemand Internet Client</h1>
<br>
<!---
    - Here is the form definition to gather logon information from the
    - user and submit it to the Internet client script. An important
    - thing to note here is that the ACTION item must include a fully
    - qualified domain name.
---->
<FORM METHOD=POST ACTION="../../../cgi-bin/arswww.cgi">
<H4>Please enter your logon information:</H4>
<P>
<br>
<pre>
<b>Server Name:</b> <input type=text name=_server value=wishart>
<br>
<br>
<b>User:</b>        <input type=text name=_user size=10 value=rpaton>
<br>
<br>
<b>Password:</b>    <input type=password name=_password>
</pre>
<input type=hidden name=_function value=logon>
<input type=hidden name=_html value=template.htm>
<input type=submit value="Submit">
<input type=reset value="Reset">
</FORM>
</body>
</html>
```

Setting up the system path

In “Copying files to HTTP server” on page 405, you copy some DLL and INI files to the D:\IBMHttpServer\cgi-bin directory. You need to let the system know how to find them. You do this by updating the Window’s system path:

1. In Windows 2000, go to the desktop, right-click the **My Computer** icon, and select **Properties** from the context menu.
2. When the System Properties menu comes up, click the **Advanced** tab, and you see the Environment Variables button in the middle of the menu.

3. Click **Environment Variable**.
4. On the Environment Variables screen, there are two windows; the one on top is the *User variables* and the one on the bottom is the *System variables*. Click **System Variables**.
5. Scroll down the window until you see Path. Double-click **Path**.
6. When the edit window appears, add your path D:\IBMHTTPServer\cgi-bin in the end of the path variable. Click **OK**. This allows the system to access the DLLs you added from ODWEK.
7. Click **OK** and **OK** again to save your changes and get out of the System Environments screen.

Important: When adding D:\IBMHTTPServer\cgi-bin in the end of the existing path variable, be sure to add a "**;**" *before* D:\IBMHTTPServer\cgi-bin.

Testing logon.htm

After HTTP and ODWEK configuration, let us test the logon.htm:

1. Start your browser.
2. If the browser is launched on the same machine where you installed HTTP server and ODWEK, enter the following URL:

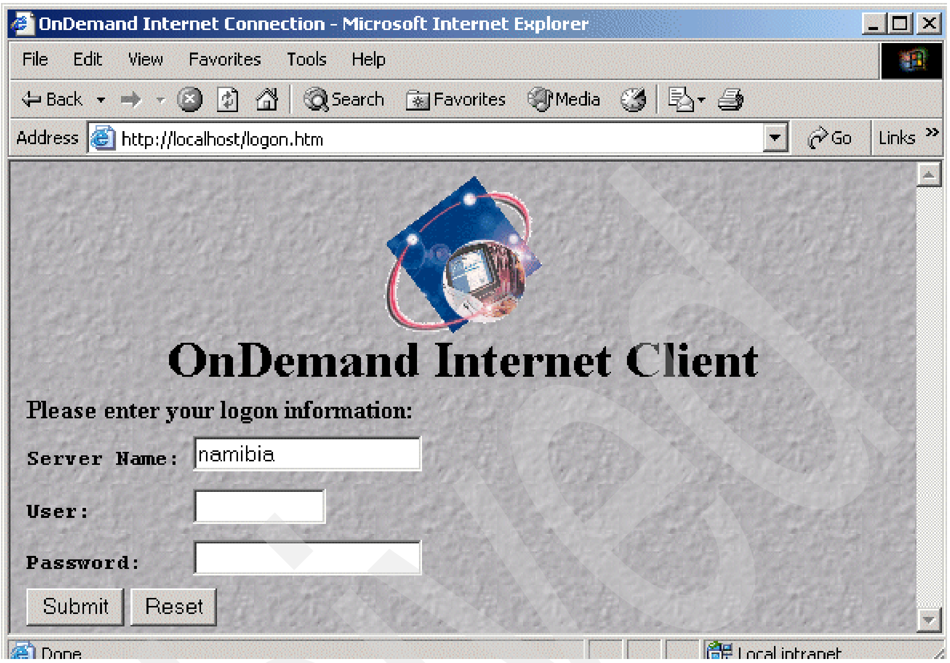
   ```
   http://localhost/MyODWEK/logon.htm
   ```

   If the browser is launched on a machine other than the server where you installed HTTP server and ODWEK, enter the following URL:

   ```
   http://congo/MyODWEK/logon.htm
   ```

   Substitute congo with the actual server name you installed your HTTP server and ODWEK. Figure 12-16 shows a sample logon screen.

.


OnDemand Internet Connection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/logon.htm
Go Links
Submit
Reset
Done
Local intranet

OnDemand Internet Client

Please enter your logon information:

Server Name: namibia

User:

Password:

Figure 12-16 ODWEK CGI logon screen

Congratulations! Your HTTP server ODWEK configuration is successful. The system is able to locate the logon screen and all the associated images in the /images directory. If this is not the case, then, let us troubleshoot the problem.

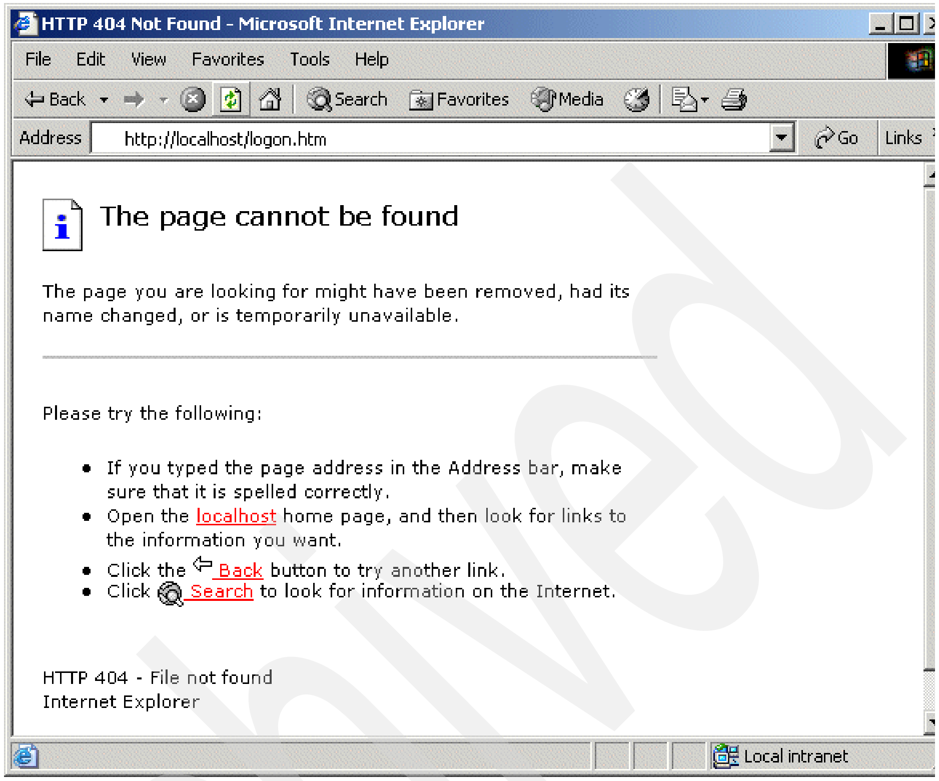
Trouble shooting

There could many reasons why you do not see the screen as shown in Figure 12-16:

- If you get the message `The page cannot be found`, with the error "`HTTP 404 - File not found`" (see Figure 12-17), this means the HTTP server cannot locate the logon.htm file. Check to be sure the logon.htm file is spelled correctly and that it is in the correct directory.

  Logon.html file should be in D:\IBMHTTPServer\htdocs\en_US\MyODWEK directory. (Substitute D:\IBMHTTPServer with the home directory where you installed your HTTP server.)

  If you do not have the file, copy it from D:\ODWEK\samples directory.


HTTP 404 Not Found - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/logon.htm
Go Links
Local intranet

The page cannot be found

The page you are looking for might have been removed, had its name changed, or is temporarily unavailable.

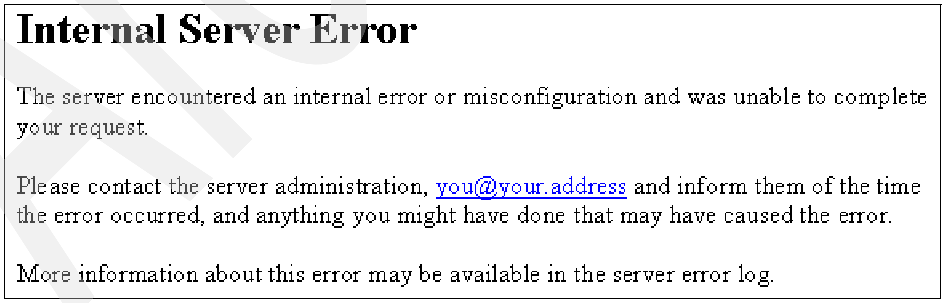
Please try the following:

- If you typed the page address in the Address bar, make sure that it is spelled correctly.
- Open the localhost home page, and then look for links to the information you want.
- Click the Back button to try another link.
- Click Search to look for information on the Internet.

HTTP 404 - File not found
Internet Explorer

Figure 12-17 Logon.htm can not be found

- What if you get the message, `Internal Server Error`, as shown in Figure 12-18?



Internal Server Error

The server encountered an internal error or misconfiguration and was unable to complete your request.

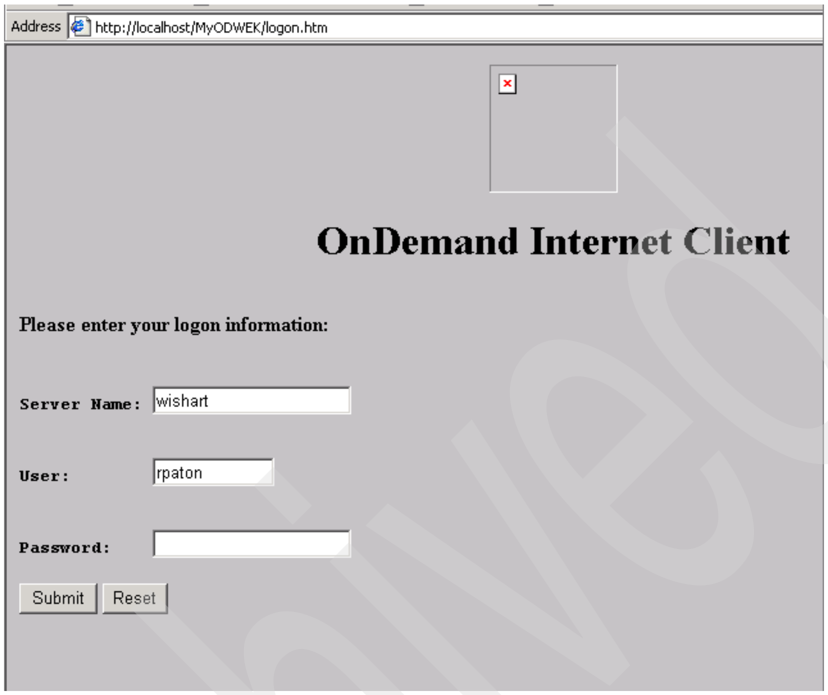
Please contact the server administration, you@your.address and inform them of the time the error occurred, and anything you might have done that may have caused the error.

More information about this error may be available in the server error log.

Figure 12-18 Internal server error

This may be caused by an incorrect setup for the Path variable to the DLLs or Shared Libraries. Check the system environment variables. See “Setting up the system path” on page 410.

- What if you see the logon screen, but no image appears as in Figure 12-19?


Address http://localhost/MyODWEK/logon.htm

OnDemand Internet Client

Please enter your logon information:

Server Name: wishart

User: rpaton

Password:

Figure 12-19 Logon screen but no images

The logon.htm file is pointing to the wrong directory to look for the images. This may be caused by one of the following three reasons:

– The logon.htm file is not pointing to the image directory.

  In the logon.htm, check the directory in bold text:

```
<body background="/images/grytxtr4.jpg" bgcolor="#c2c2c2">
<p align="center"><img src="/images/odcol2in.gif" width="100"
height="100"></p>
```

– The httpd.conf file does not point to the correct Alias directory.

  In the httpd.conf, check the directory in bold text:

```
Alias /images/    "D:/IBMHttpServer/cgi-bin/images/"
```

– The arswww.ini file, Images=/images is incorrect.

  In the arswww.ini, check the directory in bold text:

```
ImageDir=/images
```

Recheck all three files and correct the values as required.

12.1.6 Testing CGI ODWEK

This next step is to test the CGI script. So far you have only accessed the logon.htm file and images. When you enter an OnDemand server, user ID, and password, and then click **Submit,** the CGI script gets involved.

The CGI uses the template.htm located in the templates directory, connects to the OnDemand server, checks the users authority, and returns all the folders the user is authorized to see. You will know the CGI is working when you see the Select a folder screen (Figure 12-20).

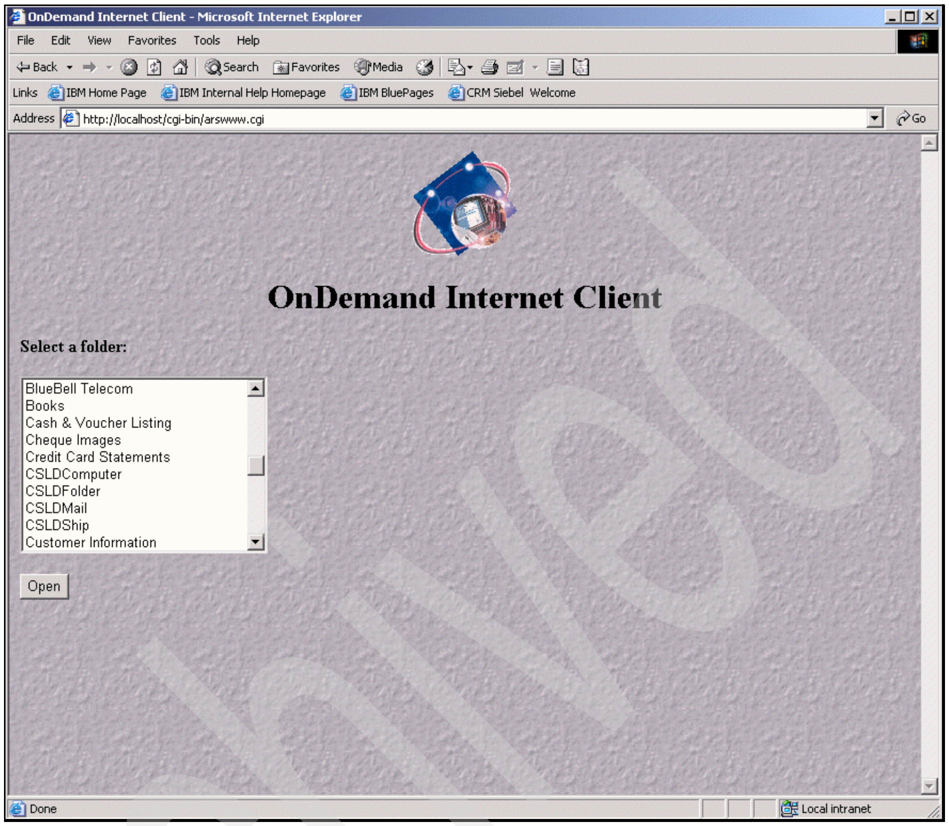
To test the CGI ODWEK:

1. From the logon screen, enter Server Name (in our scenario, it is `wishart`). Enter user ID and user password.

   Note that you must have a valid OnDemand user ID and password.
2. Click **Submit**.

If you see the Select a folder screen (Figure 12-20), then the installation of the CGI ODWEK is successful.

If you do get the logon screen and the images, but do not get the Select a folder screen, then the HTTP server cannot find the CGI script, or there may be a problem with the arswww.ini file. Make sure you have copied all the needed files into the /cgi-bin directory. Re-check the arswww.ini file and be sure you have created the subdirectories for images, applets, templates, tmp and tmp/cache.


OnDemand Internet Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Home Page IBM Internal Help Homepage IBM BluePages CRM Siebel Welcome
Address http://localhost/cgi-bin/arswww.cgi
Go
Done
Local intranet

OnDemand Internet Client

Select a folder:

BlueBell Telecom
Books
Cash & Voucher Listing
Cheque Images
Credit Card Statements
CSLDComputer
CSLDFolder
CSLDMail
CSLDShip
Customer Information

Open

Figure 12-20 ODWEK select a folder screen

12.2 Line data applets

ODWEK ships with two line data applets. A *line data applet* is a Java applet that enables you to view line data on your Web browser as line data, instead of having the server converting it to HTML and displaying as HTML.

Both line data applets are doing the same thing. The new one, Version 2, is written based on Swing technology.

Swing, introduced in Java Version 1.2, is a powerful graphical toolkit that enables the developers to write applications with rich graphical user interfaces (GUI). Swing gives Java applications the professional look and feel that has long been shared by their C++ and Visual Basic counterparts, and goes even further with a range of new components and controls, allowing a customizable “look-and-feel”.

You do have choices when it comes to viewing line data. You can view the line data on a Web browser using one of the applets or view the line data as ASCII or native mode by setting the LineView parameter in the arswww.ini file to ASCII, applet, or native.

```
LineViewing=[ascii,applet,native]
```

There are two line data applets in the /applets directory:

```
ODLineDataViewer.jar
ODLineDataViewer2.jar
```

Important: You must have JRE 1.4.1 or higher to use the new line data applet, ODLineDataViewer2.jar.

ODWEK uses the older line data applet by default. To use the Version 2 applet, edit the arswww.ini file and add the following line in the line to the Default Browser section of the arswww.ini file:

```
[default browser]
ODApplet.version=2
```

To launch new line data applet:

1. Launch a browser and go to the ODWEK home page.

   If you launch the browser on the same machine where you install HTTP server and ODWEK, enter the following URL:

   ```
   http://localhost/MyODWEK/logon.htm
   ```

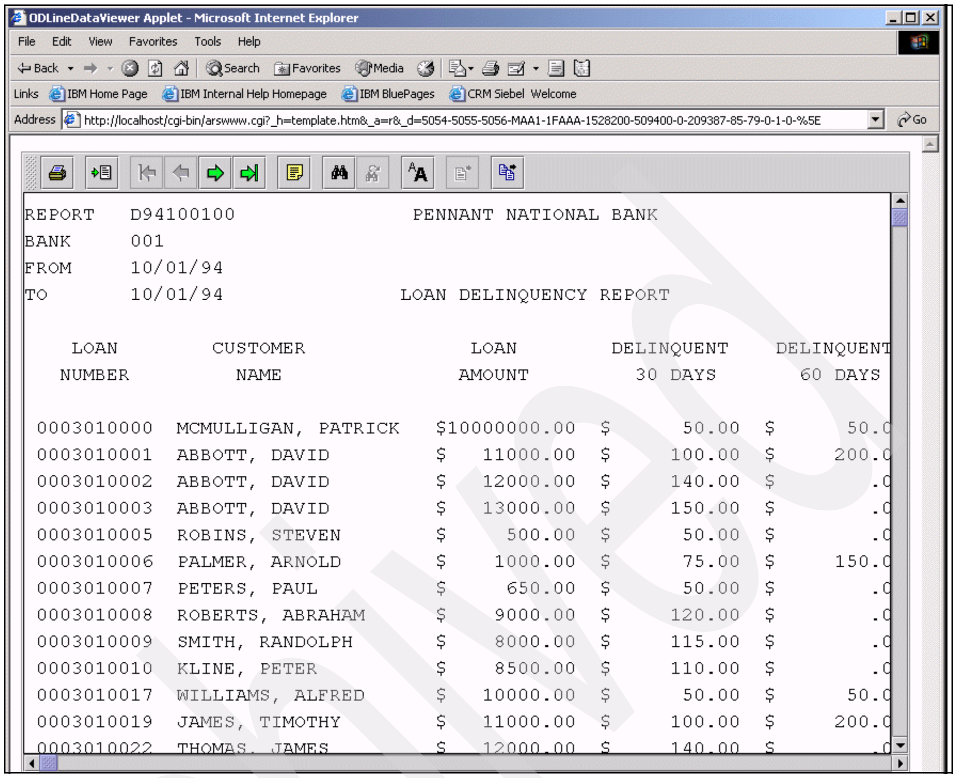
   Otherwise, enter the following URL:

   ```
   http://congo/MyODWEK/logon.htm
   ```

   Substitute congo with the server in your setup.
2. Sign on to the OnDemand server.
3. On the Select a folder screen, select a folder that has some line data.
4. Search for a line data file you want to display,
5. Select one of the line data files from the hit list. ODWEK opens the line data file in the new line data applet. If you do not select a line data document, the line data applet will not launch.

What's new in the data applet Version 2

Figure 12-21 shows the new line data applet. As mentioned earlier, the new applet is built on Java Swing technology, which has a much more user friendly interface and has richer browser user interaction capabilities.


ODLineDataViewer Applet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Home Page IBM Internal Help Homepage IBM BluePages CRM Siebel Welcome
Address http://localhost/cgi-bin/arswww.cgi?_h=template.htm&_a=r&_d=5054-5055-5056-MAA1-1FAAA-1528200-509400-0-209387-85-79-0-1-0-%5E
Go

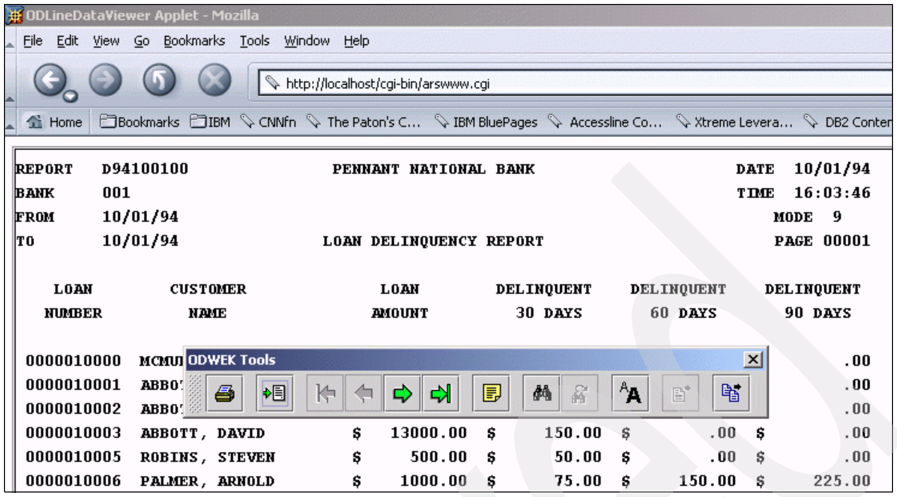
```
REPORT   D94100100                PENNANT NATIONAL BANK
BANK     001
FROM     10/01/94
TO       10/01/94                LOAN DELINQUENCY REPORT

    LOAN         CUSTOMER              LOAN         DELINQUENT    DELINQUENT
   NUMBER          NAME               AMOUNT         30 DAYS       60 DAYS

 0003010000  MCMULLIGAN, PATRICK  $10000000.00  $      50.00  $      50.0
 0003010001  ABBOTT, DAVID        $   11000.00  $     100.00  $     200.0
 0003010002  ABBOTT, DAVID        $   12000.00  $     140.00  $        .0
 0003010003  ABBOTT, DAVID        $   13000.00  $     150.00  $        .0
 0003010005  ROBINS, STEVEN       $     500.00  $      50.00  $        .0
 0003010006  PALMER, ARNOLD       $    1000.00  $      75.00  $     150.0
 0003010007  PETERS, PAUL         $     650.00  $      50.00  $        .0
 0003010008  ROBERTS, ABRAHAM     $    9000.00  $     120.00  $        .0
 0003010009  SMITH, RANDOLPH      $    8000.00  $     115.00  $        .0
 0003010010  KLINE, PETER         $    8500.00  $     110.00  $        .0
 0003010017  WILLIAMS, ALFRED     $   10000.00  $      50.00  $      50.0
 0003010019  JAMES, TIMOTHY       $   11000.00  $     100.00  $     200.0
 0003010022  THOMAS, JAMES        $   12000.00  $     140.00  $        .0
```

Figure 12-21 Line data applet Version 2

By just looking at it, you do not see much difference between the old and new line data applet; but there are a few differences:

- With the Version 2 applet, there are the same 12 icons across the tool bar as with the older applet, but some of the functions are enhanced. For example, the toolbar now can be floated — you can move it wherever you want to on the screen for your convenience (see Figure 12-22).

  You can have the tool bar on the top, right, bottom, left or in the anywhere in the window. To move the tool bar, just click the area next to the print icon, hold down the mouse button, and drag the tool bar to your new location.


ODLineDataViewer Applet - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://localhost/cgi-bin/arswww.cgi
Home Bookmarks IBM CNNfn The Paton's C... IBM BluePages Accessline Co... Xtreme Levera... DB2 Conter
ODWEK Tools

```
REPORT   D94100100            PENNANT NATIONAL BANK                       DATE  10/01/94
BANK     001                                                              TIME  16:03:46
FROM     10/01/94                                                            MODE  9
TO       10/01/94            LOAN DELINQUENCY REPORT                         PAGE 00001

    LOAN        CUSTOMER              LOAN        DELINQUENT   DELINQUENT   DELINQUENT
   NUMBER         NAME               AMOUNT         30 DAYS      60 DAYS      90 DAYS

 0000010000  MCMU                                                                  .00
 0000010001  ABBO                                                                  .00
 0000010002  ABBO                                                                  .00
 0000010003  ABBOTT, DAVID        $   13000.00  $     150.00  $         .00  $      .00
 0000010005  ROBINS, STEVEN       $     500.00  $      50.00  $         .00  $      .00
 0000010006  PALMER, ARNOLD       $    1000.00  $      75.00  $      150.00  $   225.00
```

Figure 12-22 Moving the tool bar

- Version 2's printing menu is different than Version 1's. There are also more features that come with the later version.

  With the new printing menu, you can:

  – Set the page margins.

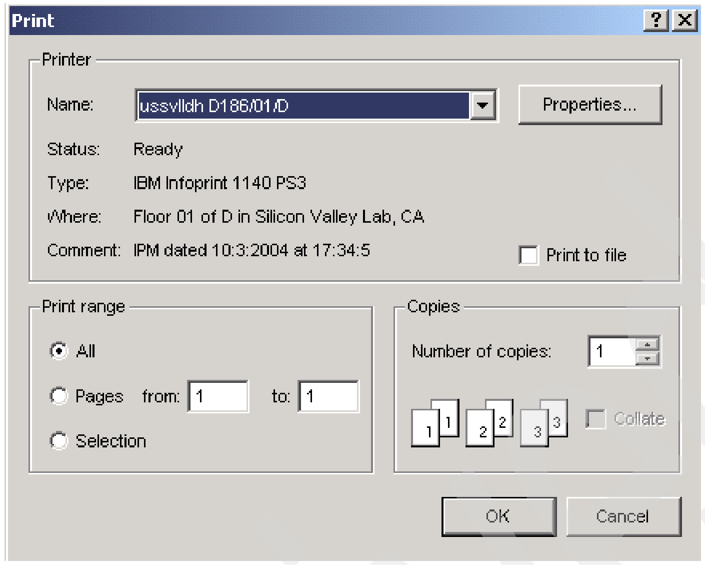
  – Set the page orientation (portrait or landscape).

  – Print the current page.

  – Print preview.

  – Remember the changes you have made when you change the margins or orientation.

  Figure 12-23 shows the printing menu with Version 1 of the line data applet. Figure 12-25 shows the printing menu with Version 2 of the line data applet.



Name: ussvlldh D186/01/D

Status: Ready

Type: IBM Infoprint 1140 PS3

Where: Floor 01 of D in Silicon Valley Lab, CA

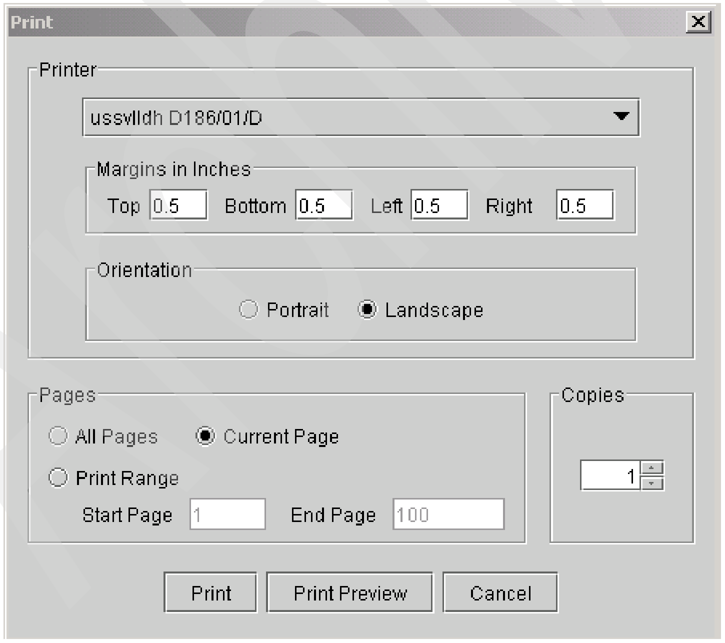
Comment: IPM dated 10:3:2004 at 17:34:5

Print to file [ ]

Print range: [✓ All] Pages from: 1 to: 1 Selection

Number of copies: 1

Figure 12-23 Line data applet Version 1 - Printing menu



Printer: ussvlldh D186/01/D

Margins in Inches: Top 0.5 Bottom 0.5 Left 0.5 Right 0.5

Orientation: Portrait [✓ Landscape]

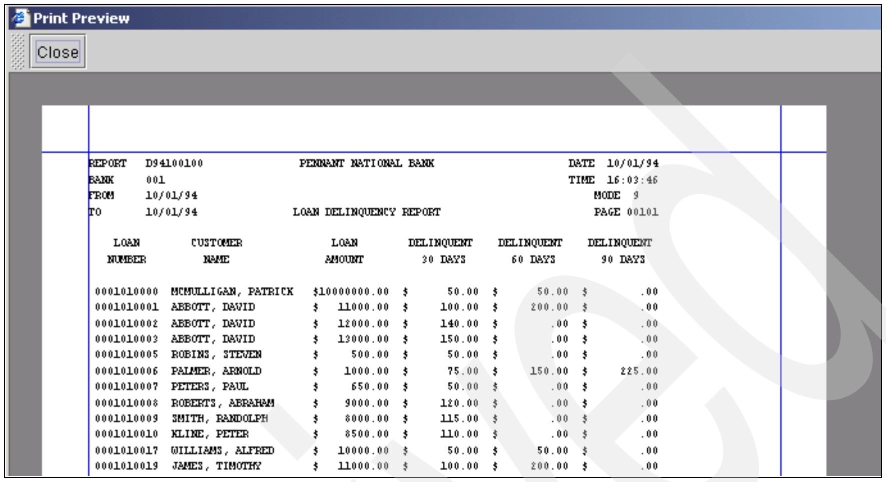
Pages: All Pages [✓ Current Page] Print Range

Start Page 1 End Page 100

Copies: 1

Figure 12-24 Line data applet Version 2 - Printing menu

Change the margins and orientation, and click **Print Preview** to see a preview of what your print-out would look like. Figure 12-25 shows a print preview sample.



```
REPORT   D94100100              PENNANT NATIONAL BANK                    DATE  10/01/94
BANK     001                                                             TIME  16:03:46
FROM     10/01/94                                                          MODE  9
TO       10/01/94             LOAN DELINQUENCY REPORT                      PAGE 00101

   LOAN        CUSTOMER              LOAN       DELINQUENT   DELINQUENT   DELINQUENT
  NUMBER         NAME               AMOUNT        30 DAYS      60 DAYS      90 DAYS

 0001010000 MCMULLIGAN, PATRICK  $10000000.00 $     50.00 $     50.00 $        .00
 0001010001 ABBOTT, DAVID        $   11000.00 $    100.00 $    200.00 $        .00
 0001010002 ABBOTT, DAVID        $   12000.00 $    140.00 $       .00 $        .00
 0001010003 ABBOTT, DAVID        $   13000.00 $    150.00 $       .00 $        .00
 0001010005 ROBINS, STEVEN       $     500.00 $     50.00 $       .00 $        .00
 0001010006 PALMER, ARNOLD       $    1000.00 $     75.00 $    150.00 $     225.00
 0001010007 PETERS, PAUL         $     650.00 $     50.00 $       .00 $        .00
 0001010008 ROBERTS, ABRAHAM     $    9000.00 $    120.00 $       .00 $        .00
 0001010009 SMITH, RANDOLPH      $    8000.00 $    115.00 $       .00 $        .00
 0001010010 KLINE, PETER         $    8500.00 $    110.00 $       .00 $        .00
 0001010017 WILLIAMS, ALFRED     $   10000.00 $     50.00 $     50.00 $        .00
 0001010019 JAMES, TIMOTHY       $   11000.00 $    100.00 $    200.00 $        .00
```

Figure 12-25 Print preview sample screen

12.3 Installing the ODWEK servlet on Windows

In this section, we show you a step-by-step process to install the ODWEK servlet on Windows.

The *ODWEK servlet* is a set of code, with Java Native Interface (JNI) wrapped around it. It is more efficient than the CGI version of ODWEK. This is because for every single request from a user, CGI has to spawn (start) a new process on the server. Over time, this can have performance impact on server activity. Running the ODWEK servlet, on the other hand, re-uses the code and does not spawn a new process for every user request.

The ODWEK servlet requires an HTTP server *and* an application server.

An application server is a program that runs on a mid-tier server and handles all application operations between clients and servers. Application servers provide a platform independent programming interface for developing portable applications in a variety of programming languages.

In our scenario, we use IBM WebSphere Application Server as the required application server for ODWEK servlet.

Here are the software requirements for the ODWEK servlet:

- Windows 2000 Server
- IBM WebSphere Application Server Version 5.0 or higher
- IBM HTTP Server Version 1.3.1 or higher (we installed Version 1.3.28 before WebSphere)
- ODWEK Version 7.1.1.1 or Version 7.0.1.13
- OnDemand Version 7.1.1.1
- Supported browsers:
  - Netscape Navigator 4.06 or higher
  - Microsoft Internet Explorer 4.01 or higher

> **Note:** Although we tested many browsers, ODWEK only supports Netscape and Microsoft Internet Explorer.

These are our assumptions:

- OnDemand is installed on a server called *ODServer1*.
- OnDemand server is called *wishart*.
- A line data report is loaded in OnDemand.
- A valid user ID is created for OnDemand.
- D:\ drive is used for installation.
- The system starts from scratch — there is no ODWEK CGI, HTTP server, or WebSphere Application Server is installed.
- IBM HTTP server and WebSphere Application Server are installed on congo.

Figure 12-26 shows the ODWEK servlet implementation in our scenario. Note that we refer to the specific system names and directories here for easier presentation. Substitute these values with your system specific setup.

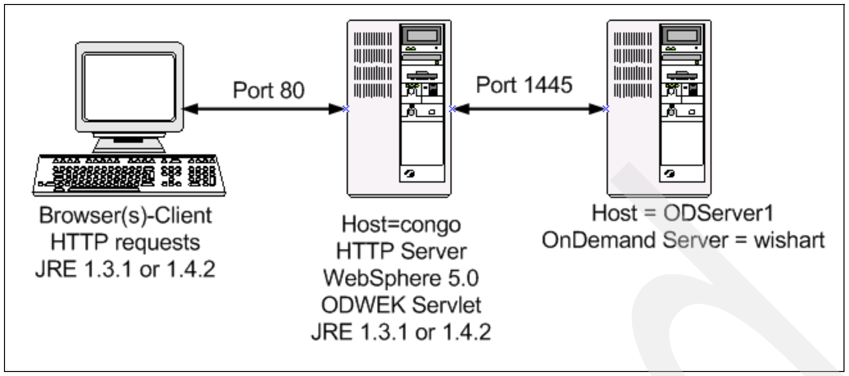


Figure 12-26 ODWEK servlet implementation

Next we describe the steps that are needed to install ODWEK servlet. We first summarize them as follows:

1. Install Java Runtime Environment (JRE).
2. Install WebSphere Application Server.
3. Install ODWEK.
4. Install ODWEK modifications and Program Temporary Fixes (PTF).
5. Copy files.
6. Set up HTTP server.
7. Set up the system environment.
8. Build .ear file in WS using ATT.
9. Deploy the .ear file using WS admin console.
10. Customize the arswww.ini file.
11. Test the servlet.
12. Customize HTTP server.
13. Customize ODWEK.
14. Test ODWEK to OnDemand with one or more browsers.

In the following sections, we describe each step in detail.

12.3.1 Installing Java Runtime Environment (JRE)

If you want to use the ODWEK line data applet Version 2, you must install JRE Version 1.4.1 or above; otherwise, at the minimum you should have JRE Version 1.3.1.

See 12.1.1, “Installing Java Runtime Environment (JRE)” on page 390 for details.

For your convenience, we present a short summary as follows:

1. Download JRE from `http://www.java.sun.com`.
2. Double-click the downloaded executable file.
3. Click **I accept the terms in the license agreement**, and then click **Next**.
4. Select **Typical** and then click **Next**.
5. Click **Finish** when the installation is completed.
6. To verify installation, open a DOS command window and type:

   ```
   java -fullversion.
   ```

   You should see the version of JRE you have just installed.

12.3.2 Installing WebSphere Application Server 5.0 for Windows

In this section, we describe how to install WebSphere Application Server and demonstrate various ways of starting and stopping the server.

Installing WebSphere Application Server 5.0 software

To install:

1. Place the WebSphere Application Server 5.0 in your CD-ROM drive. The LaunchPad program will start to help you install WebSphere. If the Launch Pad does not start, go to the CD-ROM drive, navigate to the NT directory, and double-click **launchpad.bat**.
2. After a short time, you will see the select language screen. Select your language and click **OK**.
3. On the next screen, click **Install the Product**.
4. The select language screen will be shown again. Select the language you want and click **OK**.
5. On the next screen, it asks you to verify that you really want to install WebSphere. Click **Next** to confirm.
6. On the License Agreement screen, click **I accept the terms of the license agreement** and then click **Next**.
7. On Choose the setup type screen, select **Custom**. Click **Next**.
8. On the Custom Installation screen, it asks you what products you want to install with WebSphere Application Server. There are eight items in the products menu. Select the following five products:

- Application Server (WebSphere Application Server 5.0)
- Administration
- IBM HTTP Server
- Web Plug-ins
- Javadocs

Click **Next** to continue.

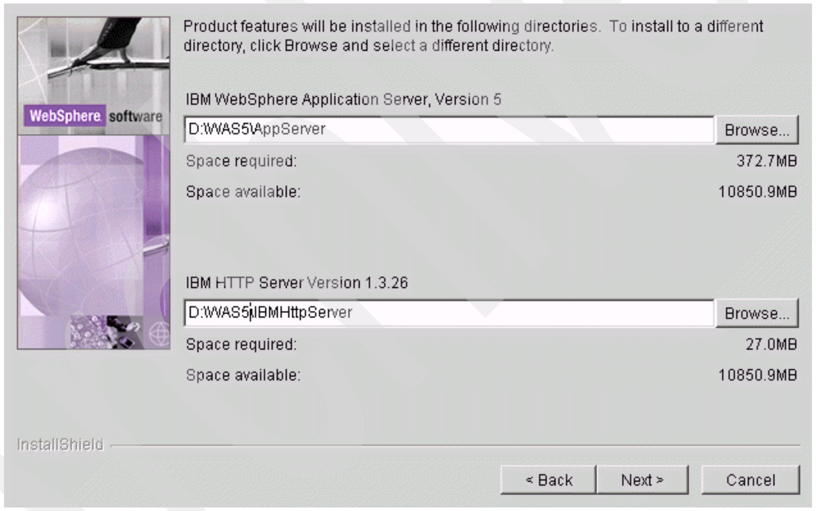
9. On the next screen (Figure 12-27):

   a. Enter `D:\WAS5\AppServer` for IBM WebSphere Application Server field

   b. Enter `D:\WAS5\AppServer\IBMHttpServer` for IBM HTTP Server field.

   c. Click **Next**.

Tip: To make working with WebSphere Application Server easier, use a shorter named directory as stated. You may substitute D:\ drive with C:\ drive.


Product features will be installed in the following directories. To install to a different directory, click Browse and select a different directory.

IBM WebSphere Application Server, Version 5: D:\WAS5\AppServer

Space required: 372.7MB

Space available: 10850.9MB

IBM HTTP Server Version 1.3.26: D:\WAS5\IBMHttpServer

Space required: 27.0MB

Space available: 10850.9MB

Figure 12-27 WebSphere installation - Directory setting screen

10. On the next screen, enter your machine name for both node and host name fields. In our scenario, we use `Alley` for node and `congo` for host name. Click **Next**.

11. On the next screen:

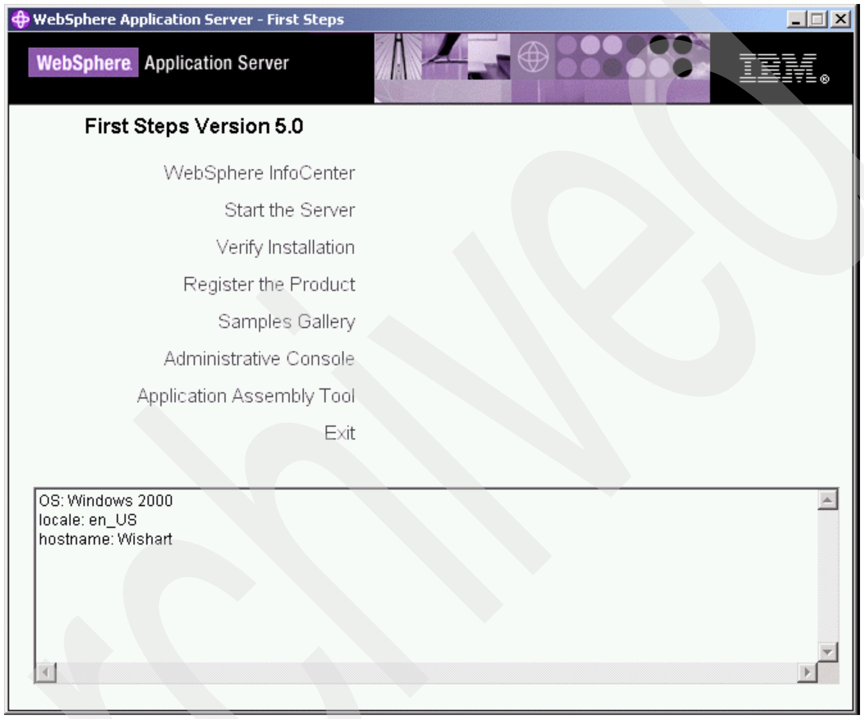
    a. Enter a valid user ID and password. This user must have authority to control WebSphere. Usually this is an ID with administrative authority.

    b. Select **Run WebSphere Application Server as a service** box.

c. Click **Next** to continue.

12. On the next screen, the system asks you to verify your installation selections.Click **Next**. The WebSphere installation will start, which may take a few minutes.

13. On the Registration screen, unclick **Register this product now** and then click **Next**. Soon you will see the First Steps main menu as shown in Figure 12-28.



```
OS: Windows 2000
locale: en_US
hostname: Wishart
```

Figure 12-28 WebSphere - First Steps main menu

Verifying installation

To verify your WebSphere installation:

1. From the First Steps main menu (Figure 12-28), click **Start the Server**. Wait until you see a message in the bottom window that looks similar to this one:

   ```
   Server server1 open for e-business.
   ```

   This means that the default server, server1, started successfully.

2. Click **Verify the Installation** and wait for the following the messages:

   ```
   IVTL0070I: IVT Verification Succeeded
   IVTL0080I: Installation Verification is complete
   ```

If you get these messages, you have a successfully installed WebSphere. Congratulations!

3. Click **Exit** to close the First Steps window.
4. Click **Finish** to complete your installation of WebSphere.
5. **Re-boot** your server where you installed WebSphere.

Starting and stopping WebSphere application servers

There are three ways you can start and stop WebSphere Application Servers:

- Using First Steps
- Using the Startup menu
- Using the command line
- Using System WebSphere System Administration Console

Using First Steps

You can start and stop the WebSphere server as follows:

1. Select **Start** → **Programs** → **IBM WebSphere** → **Application Server v5.0** → **First Steps**.
2. Click Start the Server to start the server. Click Stop the Server to stop the server. These two options are mutually exclusive. You can only see one of the options at a time depending on the state of the WebSphere server.

Using the Startup menu

You can start and stop the WebSphere server as follows:

- To start the server, select **Start** → **Programs** → **IBM WebSphere** →
- To stop the server, select **Start** → **Programs** → **IBM WebSphere** → **Application Server v5.0** → **Stop the Server**.

Using the command line

You can open a DOS command line window, go to <WebSphere_HOME>\AppServer\bin directory, where <WebSphere_HOME> is the main directory where you installed WebSphere. In our scenario, it is D:\WAS5, and do the following steps:

- To check what servers are running:

```
serverstatus -all
```

- To start the default WebSphere server:

```
startserver server1
```

- To stop the default WebSphere server:

```
stopserver server1
```

12.3.3 Installing the ODWEK servlet

The installation of the ODWEK servlet is the same as the installation for the CGI ODWEK. Please see 12.1.3, "Installing ODWEK base code" on page 396 for detailed instructions.

For you convenience, we present a short summary as follows:

1. Double-click **setup.exe** from the installation directory.
2. Click **Next**. Click **I accept**, and then click **Next**.
3. Use `D:\ODWEK` as default directory. This is simpler directory to work with than the given one (C:\Program Files\IBM\OnDemand Web Enablement Kit).
4. Click **Next** to confirm the setup. The system will begin installing ODWEK.
5. Uncheck **view Readme file**, and click **Finish** to complete the installation.
6. On the Maintenance Complete screen, click **No, I will restart my computer later**, and then click **Finish**.

12.3.4 Installing ODWEK servlet fixes

The installation of ODWEK servlet fixes is the same as the installation done for the CGI ODWEK. Please see 12.1.4, "Installing ODWEK releases, modifications, and PTFs" on page 400 for detailed instructions.

For your convenience, we present a short summary as follows:

1. Download the latest fixes from `ftp://service.software.ibm.com/software/ondemand/fixes`.

1. Double-click **odwekwin.exe** file from where you downloaded the fixes.
2. Click **Next**. Click **I agree**, and then click **Next**.
3. Click **Next** to confirm the setting.
4. Click **No** for reviewing Redme.txt file, and then click **Next**.
5. Click **Finish** to complete the installation.
6. Re-boot your server now.

12.3.5 Copying files

You need to copy some files from the ODWEK installed directories to the directories in the HTTP server and to the WebSphere Application Server directories. In addition, you need to create some new directories.

Copying files for WebSphere Application Server

To copy the appropriate files and creating appropriate directories for WebSphere, do the following steps:

1. Copy <ODWEK_HOME>\Servlets\ArsWWWServlet.jar to <WAS_HOME>\AppServer\lib directory

   Where:

   <ODWEK_HOME> is the directory where you installed ODWEK. In our scenario, it is D:\ODWEK.

   <WAS_HOME> is the directory where you installed WebSphere. In our scenario, it is D:\WAS5.

   Note that the ArsWWWServlet.jar contains a list of ODWEK servlet classes. The <WAS_HOME>\AppServer\lib directory is where WebSphere locates its servlet class files.

2. Create the following directory under <WAS_HOME>\AppServer\classes:

   ```
   com\ibm\edms\od
   ```

   You can do this by going to a DOS command, changing to the classes directory, and use **`md com\ibm\edms\od`**.

   Your directory structure should look similar to Figure 12-29.

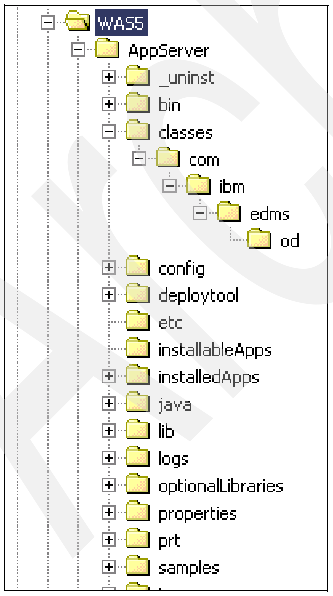


Figure 12-29 Placing the ArsWWWInterface.class file

Note that the formal packaging for ODWEK class files is com.ibm.edms.od. WebSphere builds a classpath to that location in the deployment steps, so you must create a directory structure that corresponds to the packaging and put the class file there.

3. Copy <ODWEK_HOME>\servlets\ArsWWWInterface.class to <WAS_HOME>\AppServer\classes\com\ibm\edms\od

   Where:

   <ODWEK_HOME> is the directory where you installed ODWEK (in our scenario, it is D:\ODWEK).

   <WAS_HOME> is the directory where you installed WebSphere (in our scenario, it is D:\WAS5).

Copying files for IBM HTTP server

You also need to copy the appropriate files IBM HTTP server and create the necessary directories. Follow the steps below:

1. Create the following subdirectories under <HTTP_HOME> directory (in our scenario, it is D:\IBMHTTPServer):

   ```
   servlet
   servlet\applets
   servlet\images
   servlet\templates
   servlet\tmp
   servlet\tmp\cache
   ```

   Figure 12-30 shows the resulting directory structure.

2. Copy the following files

   ```
   arswwwsl.dll
   arsct32.dll
   arsscknt.dll
   arswww.ini
   arsxenos.ini
   afp2html.ini
   afp2pdf.ini
   ```

   From <ODWEK_HOME> to <HTTP_HOME>\servlet

   Where:

   <ODWEK_HOME> is the directory where you installed ODWEK (in our scenario, it is D:\ODWEK).

   <HTTP_HOME> is the directory where you installed IBMHTTP Server (in our scenario, it is D:\IBMHTTPServer).

Note that you do not need the arswww.cgi file, because we are doing servlet, not CGI, installation. Do not copy the arssck32.dll file, because it is not needed for the servlet.

3. Copy <ODWEK_HOME>\Samples\logon.htm to <HTTP_HOME>\servlet.
4. Copy <ODWEK_HOME>\Samples\*.* to <HTTP_HOME>\servlet\templates.
5. Copy <ODWEK_HOME>\images\*.* to <HTTP_HOME>\servlet\images.
6. Copy <ODWEK_HOME>\applets\*.* to <HTTP_HOME>\servlet\applets.

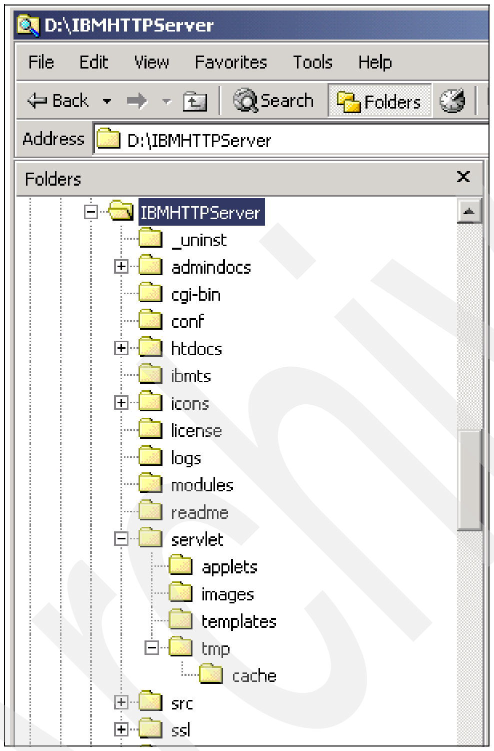


Figure 12-30 Resulting servlet directory structure under IBMHTTPServer directory

12.3.6 Setting up the system environment

After copying the DLL and INI files, you need to let the Windows system and WebSphere know where the files are located. You do this by adding a *path* and *classpath* to the Windows Path Environment:

1. Using Windows 2000 as an example, go to the desktop, and right-click the **My Computer** icon.
2. Select **Properties** from the pull-down context menu.
3. When the System Properties screen appears, click the **Advanced** tab.
4. Click **Environment Variables**.
5. On the Environment Variables screen, there are two windows, the one on top is the *User variables* and the one on the bottom is the *System variables*. You want to work on the System Variables. Scroll down to the System Variables window until you see Path. Select **Path** and click **Edit**.
6. When the Edit window appears, add <HTTP_HOME>\servlet to the end of the path value. Click **OK**.

   Be sure to append a semicolon after the existing old value before you add your new value.

   Substitute <HTTP_HOME> with the directory where you installed IBM HTTP server. In our scenario, it is D:\IBMHTTPServer.

   Adding this path enables the system to read the DLLs you have added.
7. Go back to the System Variables window. Scroll up or down the window until you see *Classpath*. Select **Classpath** and click **Edit**.
8. When the Edit window appears, add <WAS_HOME>\lib to the end of the classpath value. Click **OK**.

   Be sure to append a semicolon (;) after the existing old value before you add your new value.

   Substitute <WAS_HOME> with the directory where you installed WebSphere. In our scenario, it is D:\WAS5\AppServer\lib.
9. When finishing editing the path and classpath system variables, click **OK** and **OK** again.

Diagram of servlet install

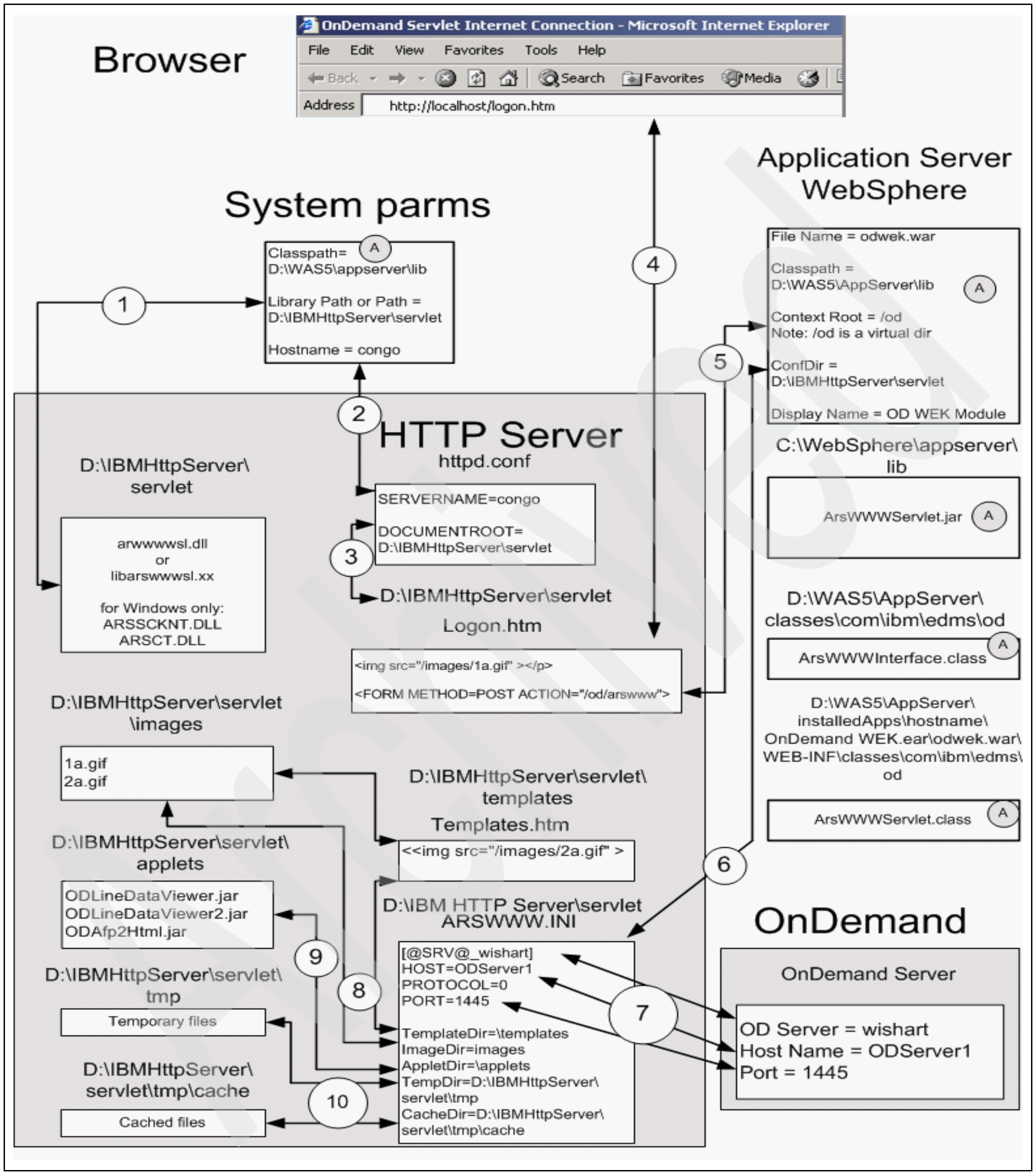
Before we go any further, let us look at the road map to see how these files are working together. Figure 12-31 shows the servlet diagram (which we have labeled with the numbers 1 through 10, as explained in the steps below).

Specifically, it details the following operations:

1. When a system starts up, it reads the system parameters and sees the path for the shared libraries and DLL files and the class path for the ArsWWWServlet.jar file.
2. HTTP server is started.
3. HTTP points to the document root.

A> WebSphere is started and locates its files and resources.

4. The user types the URL of logon.htm which locates the logon.htm in the document root.
5. The logon.htm contains the Context root of the WebSphere directory.
6. WebSphere then points to the ODWEK arswww.ini file.
7. The arswww.ini file points to the OnDemand server and port number.
8. The arswww.ini file points to the templates and images.
9. When a user requests a line data report the applet is invoked.
10. Files are cached for faster retrieval if recalled.



Browser
OnDemand Servlet Internet Connection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost/logon.htm
System parms
Classpath= D:\WAS5\appserver\lib
Library Path or Path = D:\IBMHttpServer\servlet
Hostname = congo
Application Server WebSphere
File Name = odwek.war
Classpath = D:\WAS5\AppServer\lib
Context Root = /od
Note: /od is a virtual dir
ConfDir = D:\IBMHttpServer\servlet
Display Name = OD WEK Module
C:\WebSphere\appserver\lib
ArsWWWServlet.jar
D:\WAS5\AppServer\classes\com\ibm\edms\od
ArsWWWInterface.class
D:\WAS5\AppServer\installedApps\hostname\OnDemand WEK.ear\odwek.war\WEB-INF\classes\com\ibm\edms\od
ArsWWWServlet.class
HTTP Server
httpd.conf
SERVERNAME=congo
DOCUMENTROOT= D:\IBMHttpServer\servlet
D:\IBMHttpServer\servlet
Logon.htm
<img src="/images/1a.gif" ></p>
<FORM METHOD=POST ACTION="/od/arswww">
D:\IBMHttpServer\servlet
arwwwwsl.dll or libarswwwsl.xx
for Windows only: ARSSCKNT.DLL ARSCT.DLL
D:\IBMHttpServer\servlet\images
1a.gif
2a.gif
D:\IBMHttpServer\servlet\templates
Templates.htm
<<img src="/images/2a.gif" >
D:\IBMHttpServer\servlet\applets
ODLineDataViewer.jar
ODLineDataViewer2.jar
ODAfp2Html.jar
D:\IBMHttpServer\servlet\tmp
Temporary files
D:\IBMHttpServer\servlet\tmp\cache
Cached files
D:\IBM HTTP Server\servlet
ARSWWW.INI
[@SRV@_wishart]
HOST=ODServer1
PROTOCOL=0
PORT=1445
TemplateDir=\templates
ImageDir=images
AppletDir=\applets
TempDir=D:\IBMHttpServer\servlet\tmp
CacheDir=D:\IBMHttpServer\servlet\tmp\cache
OnDemand
OnDemand Server
OD Server = wishart
Host Name = ODServer1
Port = 1445
A
1
2
3
4
5
6
7
8
9
10

Figure 12-31 Servlet diagram

12.3.7 Assembling the servlet

Now that our environment and files are in place, we can use WebSphere to assemble the EAR file, and in the next step, deploy the EAR file.

EAR file stands for Enterprise Archive file. It is a place to keep or point to all the necessary files to make a Java application work.

To assemble the EAR file:

1. Launch the Application Assembly Tool (ATT) to assemble the EAR file by select **Start** → **Programs** → **IBM WebSphere** → **Application Server V5.0** → **Application Assembly Tool**.
2. When the Application Assembly screen appears (see Figure 12-32), select **Application** (from the New tab), and click **OK**.

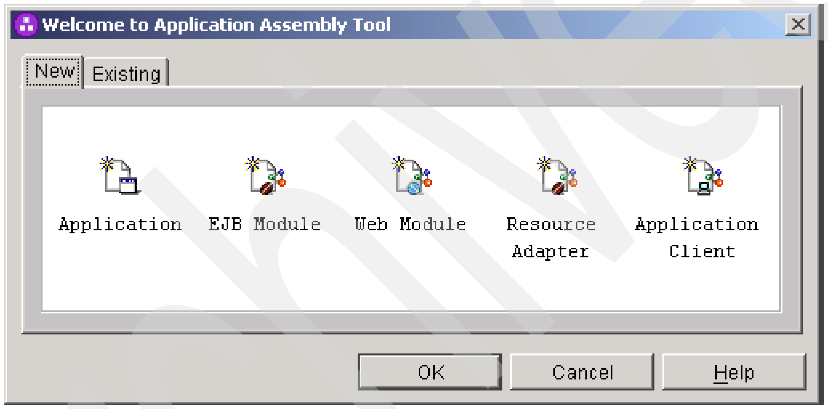
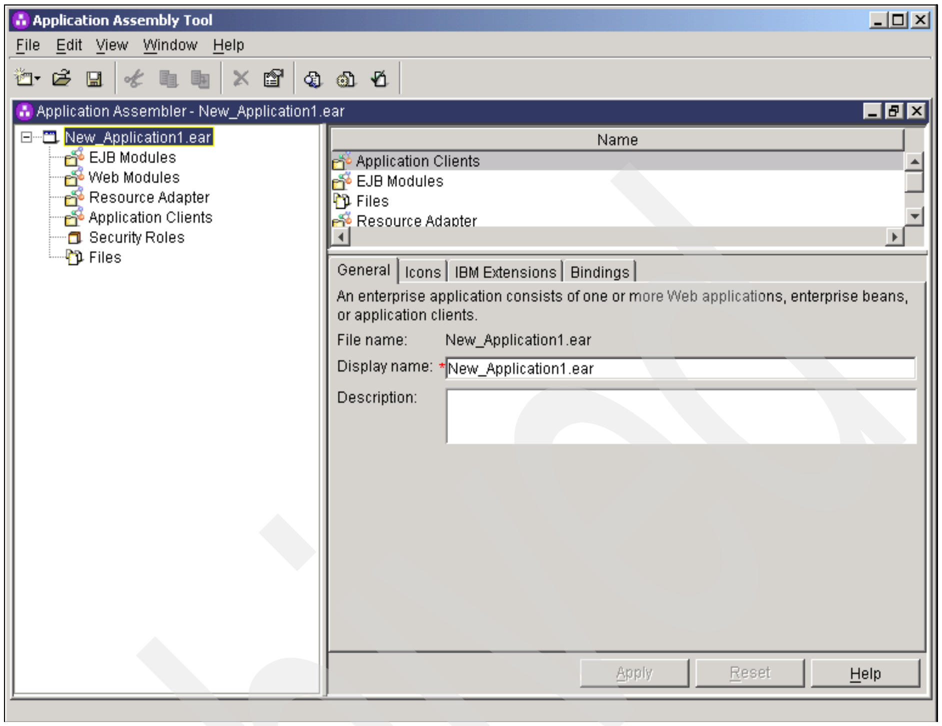


Figure 12-32 Application Assembly Tool - New Application

3. When the next screen appears (see Figure 12-33), enter `ODWEK.ear` in the Display Name field. Click **Apply**.



An enterprise application consists of one or more Web applications, enterprise beans, or application clients.

File name: New_Application1.ear

Display name: * New_Application1.ear

Description:

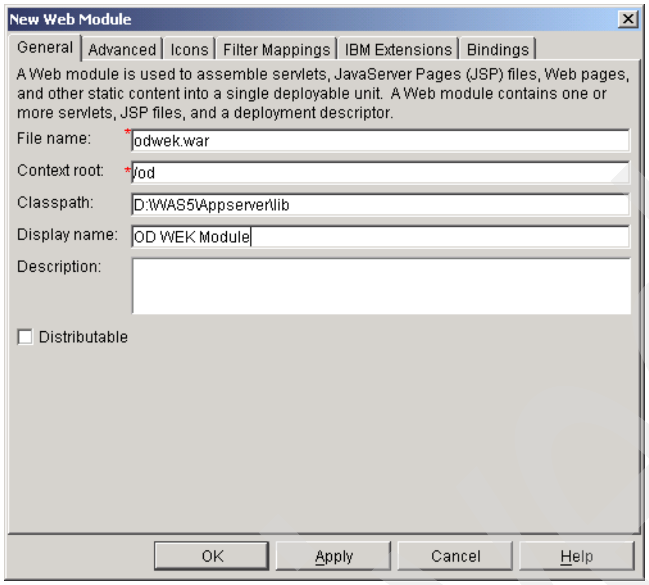
Figure 12-33 Application Assembly Tool: New EAR application

4. Create a new Web module:

   a. In the left panel, right-click **Web Modules** and select **New** from the context menu.

   b. When New Web Module window appears, enter `odwek.war` for File name, `/od` for Context root, `<WAS_HOME>\AppServer\lib` for Classpath, `OD WEK Module` for Display name (see Figure 12-34). Click **OK**.

      Substitute <WAS_HOME> with the directory where WebSphere is installed. In our scenario, we use D:\WAS5.



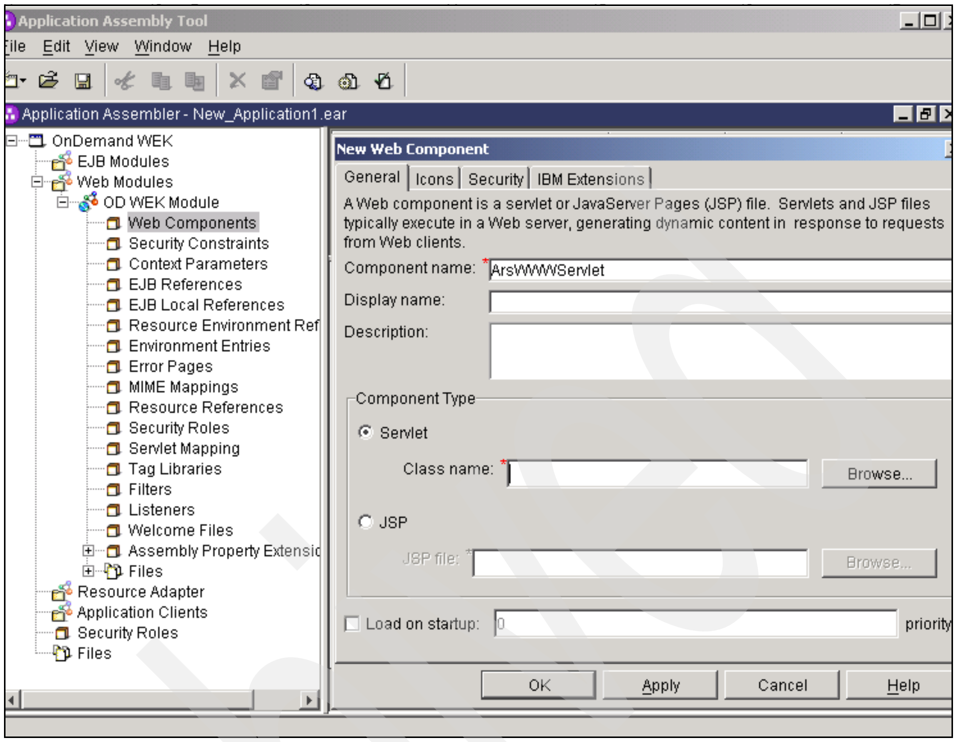
A Web module is used to assemble servlets, JavaServer Pages (JSP) files, Web pages, and other static content into a single deployable unit. A Web module contains one or more servlets, JSP files, and a deployment descriptor.

File name: odwek.war
Context root: /od
Classpath: D:\WAS5\Appserver\lib
Display name: OD WEK Module
Description:
☐ Distributable

Figure 12-34 Application Assembly Tool: New Web Module

Note: The *context root* is not a real directory. It is a virtual directory that WebSphere uses as a root directory for this application. Whenever you use the browser to logon to this application, you address the application by specifying /od as part of the URL to locate this application.

5. Create a new Web component:
   a. In the left panel, expand **Web Modules** → **OD WEK Module**.
   b. Right-click **Web Component**, and select **New** from the context menu.
   c. When the New Web Component window appears (see Figure 12-35), enter `ArsWWWServlet` for Component name, click **Servlet** for Component Type.


Application Assembly Tool

File Edit View Window Help

Application Assembler - New_Application1.ear

- OnDemand WEK
  - EJB Modules
  - Web Modules
    - OD WEK Module
      - Web Components
      - Security Constraints
      - Context Parameters
      - EJB References
      - EJB Local References
      - Resource Environment Ref
      - Environment Entries
      - Error Pages
      - MIME Mappings
      - Resource References
      - Security Roles
      - Servlet Mapping
      - Tag Libraries
      - Filters
      - Listeners
      - Welcome Files
      - Assembly Property Extensic
      - Files
  - Resource Adapter
  - Application Clients
  - Security Roles
  - Files

New Web Component

General | Icons | Security | IBM Extensions

A Web component is a servlet or JavaServer Pages (JSP) file. Servlets and JSP files typically execute in a Web server, generating dynamic content in response to requests from Web clients.

Component name: * ArsWWWServlet

Display name:

Description:

Component Type

(●) Servlet

Class name: *

( ) JSP

JSP file: *

[ ] Load on startup: 0 priority

OK | Apply | Cancel | Help

Figure 12-35 Application Assembly Tool: New Web Component

d. Click **Browse** next to the Class name box.

e. When the Select file for Class name window appears, odwek.war should appear in the Root Directory or Archive field. Click **Browse** again.

f. When the Select a Root Directory window appears, navigate to <WAS_HOME>\AppServer\lib where you placed the ArsWWWServlet.jar file, select the **ArsWWWServlet.jar** file as shown in Figure 12-36 and then click **Select**.

   Substitute <WAS_HOME> with the directory where you installed WebSphere. In our scenario, it is D:\WAS5.

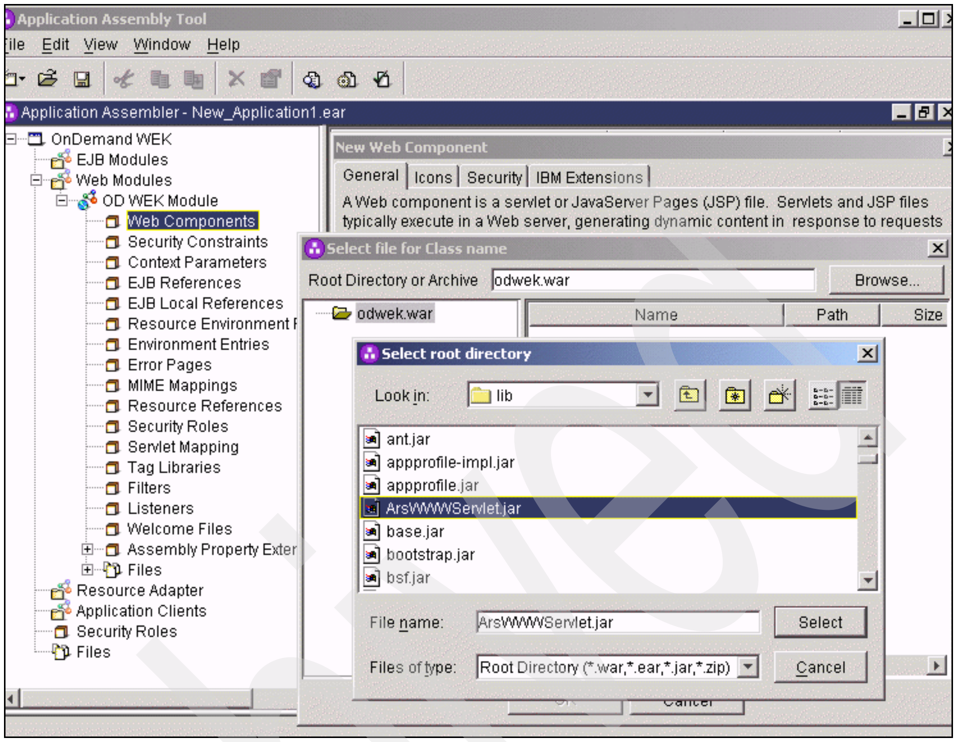
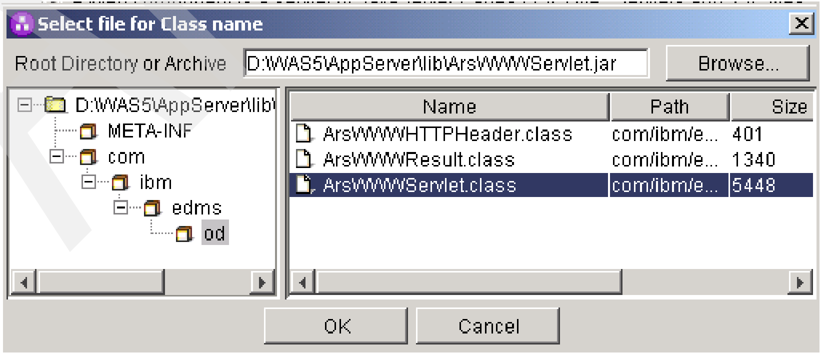


Figure 12-36 Finding and selecting the ArsWWWServlet.jar file

g. When the Select file for Class name window appears, expand **D:\WAS5\Appserver\lib\ArsWWWServlet.jar** → **com** → **ibm** → **edms** → **od**. (You may start with a different directory if you installed WebSphere in a directory other than D:\WAS5.)

h. Select **ArsWWWServlet.class** as in Figure 12-37. Click **OK.**


Select file for Class name
Root Directory or Archive D:\WAS5\AppServer\lib\ArsWWWServlet.jar
Browse...
D:\WAS5\AppServer\lib\
META-INF
com
ibm
edms
od
OK
Cancel

Name	Path	Size
ArsWWWHTTPHeader.class	com/ibm/e...	401
ArsWWWResult.class	com/ibm/e...	1340
ArsWWWServlet.class	com/ibm/e...	5448

Figure 12-37 Application Assembly Tool: Selecting file for class name

   i. After creating the Web component, you are back at the New Web Component window, your Class name should be com.ibm.edms.od.ArsWWWServlet.

6. Set the servlet's initialization parameters as follows:

   a. In the left panel, expand **Web Components** → **ArsWWWServlet.** Right-click **Initialization Parameters** and select **New** from the context menu.

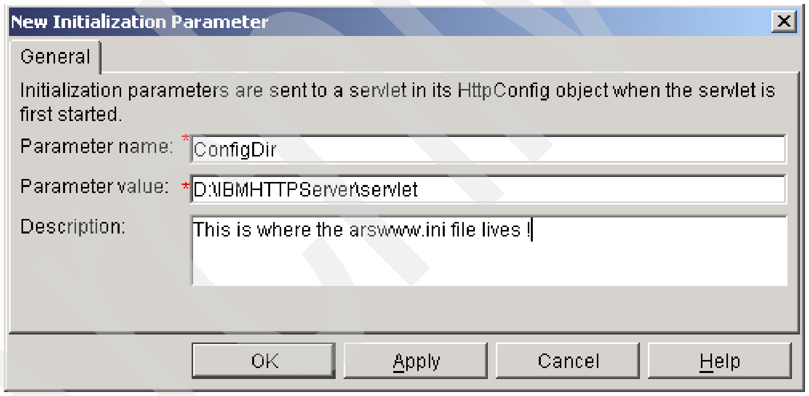
   b. When the New Initialization Parameters window appears, enter `ConfigDir` for Parameter name, `<HTTP_HOME>\servlet` for Parameter value. Optionally, you can enter the following for Description:

   ```
   This is where the arswww.ini file lives !
   ```

   Substitute <HTTP_HOME> with the directory where you installed IBM HTTP server. In our scenario, it is D:\IBMHTTPServer.

   Your Initialization parms should look similar to Figure 12-38. Click **OK**.

   Note: The `ConfigDir` is where the servlet picks up the arswww.ini file.



Initialization parameters are sent to a servlet in its HttpConfig object when the servlet is first started.

Parameter name: * ConfigDir

Parameter value: * D:\IBMHTTPServer\servlet

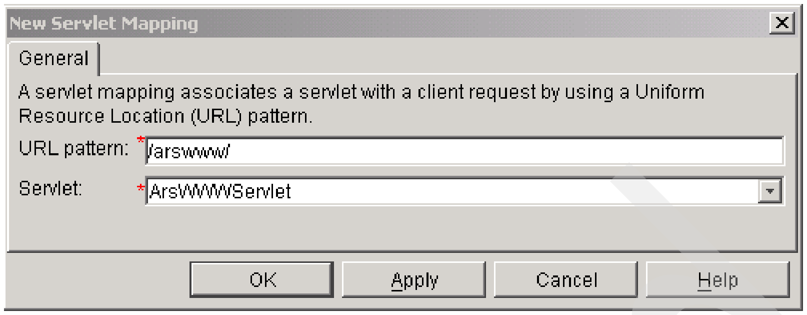
Description: This is where the arswww.ini file lives !

Figure 12-38 Initialization parms

7. Set the servlet mapping as follows:

   a. In the left panel, right-click **Servlet Mappings** and select **New** from the context menu.

   b. When the New Servlet Mapping window appears, ArsWWWServlet should already be in the Servlet box. Enter `/arswww/` for URL pattern. You must use beginning "/" and ending "/" slashes (see Figure 12-39). Click **OK**.



A servlet mapping associates a servlet with a client request by using a Uniform Resource Location (URL) pattern.

URL pattern: */arswww/

Servlet: *ArsWWWServlet

Figure 12-39 New Servlet Mapping screen

8. We need to save the EAR file:
   a. From the Application Assembly Tool main screen, select **File** → **Save As**.
   b. On the Save dialog box, enter odwek.ear, and save it under D:\TEMP directory or any directory where we can find it later during the deploy step.
9. Exit the Application Assembly Tool.

12.3.8 Deploying the ODWEK servlet

Once you assemble the EAR file, it is time to deploy it in WebSphere. Here is a summary of the steps:

1. Install the ODWEK.ear as the new application in WebSphere.
2. Update Web plug-in.
3. Start ODWEK application.
4. Restart WebSphere Application Server.

These are the detailed step-by-step procedures to deploy the ODWEK servlet:

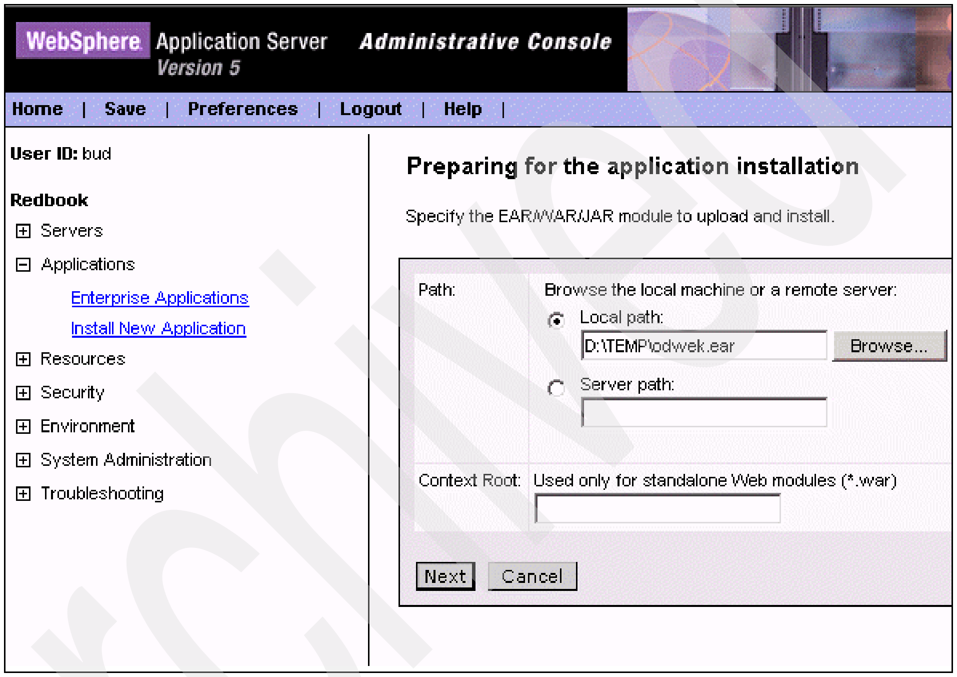
1. Launch the WebSphere Administration Console by select **Start** → **Programs** → **IBM WebSphere** → **Application Server v5.0** → **Administrative Console**.

   Enter a user ID to access the WebSphere Administrative console. This ID is used to track your session while you are working within the WebSphere Administrative console. Any ID will do. Click **OK**.

2. Install the new ODWEK application:

   a. From the main administration window, select **Applications** → **Install New Application**.

   b. When the Preparing for the application installation window appears, select **Local path.** Click **Browse** to locate the odwek.ear file you saved in the previous step. (In our scenario, it is in D:\TEMP directory.) Select odwek.ear, click **Next**. See Figure 12-40.



Specify the EAR/WAR/JAR module to upload and install.

Path: Browse the local machine or a remote server:
[● Local path:] D:\TEMP\odwek.ear
○ Server path:
Context Root: Used only for standalone Web modules (*.war)

Figure 12-40 Locating the .ear file

   c. Accept the defaults on Preparing for the application install, click **Next**.

   d. Click **Next** at Step 1: Provide options to perform the installation.

   e. Click **Next** at Step 2: Install New Application window.

   f. Click **Next** at Step 3: Map modules to application servers.

   g. Click **Finish** on Step 4: Summary.

   h. On the Installing window, click **Save to Master Configuration**. On the next window, click **Save** *again*.

3. Update Web Server plug-in:
   a. From the main window, expand **Environment**.
   b. Click **Update Web Server Plugin**. On the right panel, click **OK**. Wait until you see the following message:

      ```
      The web server plug-in configuration was updated successfully.
      ```
   c. Click **Cancel** to leave the plug-in screen.

> **Important:** Anytime you make changes to the application, you must re-generate the server plug-in!

4. Start the ODWEK application:
   a. From the main System Administrative window, click **Enterprise Applications**.
   b. Select **OnDemand WEK** application.
   c. Click **Start** (located on the top of the Window).

   You should see the red X change to a green arrow when the application is started.
5. Click **Logout** to exit the Administrative console.

   You have not deployed the ODWEK servlet.
6. Restart WebSphere Application Server.
   a. Select **Start** → **Program** → **IBM WebSphere Application Server v5.0** → **First Steps**.
   b. Click **Stop the Server**. Wait until a similar message appears on the bottom of the window:

      ```
      Server server1 stop completed.
      ```
   c. Click **Start the Server**. Wait until a similar message appears on the bottom of the window:

      ```
      Server server1 open for e-business; process id is 3264.
      ```

> **Important:** Make sure the HTTP Server is running. Go to the Windows Services menu and see if IBM HTTP Server is running. If it is not, start it.
>
> **Tip:** There are many ways to stop and restart WebSphere Application Server. Refer to “Starting and stopping WebSphere application servers” on page 427 for some of the methods.

12.3.9 Customizing arswww.ini

We need to customize the arswww.ini file:

1. Go to the <HTTP_HOME>\servlet

   Substitute <HTTP_HOME> with the directory where you installed IBM HTTP Server. In our scenario, it is D:\IBMHTTPServer.
2. Make a backup of the arswww.ini file first.
3. Open the file with any text editor such as Notepad or Wordpad for editing.
4. Make the following changes to arswww.ini file near the top of the file. Change the following items:
   - OnDemand server name from `gunnar` to `wishart` (or your OD server)
   - Host name from `gunnar` to `wishart` (or your host or IP address)
   - Templates from `/home/httpd/docs/templates` to `/templates`.

   Refer to Example 12-4, which shows the original arswww.ini file, and Example 12-5 , which shows the modified arswww.ini file.

Example 12-4 Original arswww.ini file

```
[@SRV@_gunnar]
HOST=gunnar
PROTOCOL=0
PORT=1445
.
.
[configuration]
TemplateDir=/home/httpd/docs/templates
AppletDir=/applets
TempDir=/tmp
CacheDir=/tmp/cache
```

Example 12-5 Modified arswww.ini file sample for the servlet

```
[@SRV@_wishart]
HOST=ODServer1
PROTOCOL=0
PORT=1445
.
.
[configuration]
TemplateDir=/templates
AppletDir=/applets
TempDir=/tmp
CacheDir=/tmp/cache
```

5. Save the modified arswww.ini file.

12.3.10 Testing the ODWEK servlet

To test the servlet:

1. Open a browser.
2. Enter the following URL:

   ```
   http://<host_name>/od/arswww
   ```

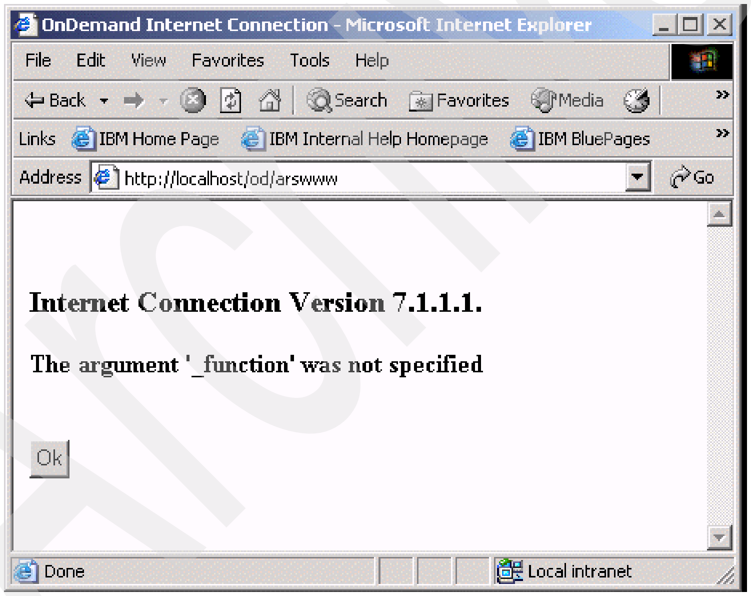
   where <host_name> is the server where you installed ODWEK application. If you launched the browser on the same machine as where you installed ODWEK servlet (as in your scenario), use:

   ```
   http://localhost/od/arswww
   ```

   Otherwise, substitute the <host_name> with the server that you installed the WebSphere Application Server and ODWEK. In our scenario, for example, it is:

   ```
   http://congo/od/arswww
   ```

3. You should see a screen similar to Figure 12-41.


OnDemand Internet Connection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Home Page IBM Internal Help Homepage IBM BluePages
Address http://localhost/od/arswww Go
Ok
Done Local intranet

Internet Connection Version 7.1.1.1.

The argument '_function' was not specified

Figure 12-41 Testing the servlet

If you see the message "Internet Connection Version 7.1.1.1" (or similar), then you can proceed to the steps below to customize HTTP, add the logon and the templates. If not, then go back and retrace the steps above and try again.

12.3.11 Customizing IBM HTTP server

Once the servlet is working, we need to add some Web pages to make it work with OnDemand. The following steps show how to customize the servlet to make it useful:

1. Update httpd.conf file:
   a. Go to <HTTP_HOME>\conf directory.
   b. Make a backup copy for httpd.conf. *This is extremely important!*
   c. Open httpd.conf in a text editor.
   d. Search through the file until you find the `documentroot statement`. It may appear in the file multiple times.
   e. Change it from:

      ```
      DocumentRoot "D:\IBMHttpServer/htdocs/en_US"
      ```

      To the location where you placed the arswww.ini file:

      ```
      DocumentRoot “<HTTP_HOME>/servlet”
      ```

      Be sure to substitute `<HTTP_HOME>` with the directory where you installed the HTTP server. In our scenario, it is D:\IBMHTTPServer.

      <HTTP_HOME>/servlet is what the IBM HTTP server uses as its homebase, or root directory. Every file it looks for is related to this location. This is important because this is where we have placed the arswww.ini and logon.htm files.

      We also added applets, images, templates, tmp, and tmp\cache subdirectories under the <HTTP_HOME>/servlet directory.

Note: We use all forward slashes “/” in the httpd.conf file. You can use back slashes, but you have to be careful when you can and cannot. It is much safer to use all forward slashes “/”.

Important: Anytime you make changes to the httpd.conf file, you must *stop* and *re-start* the HTTP Server or you will not see your changes.

12.3.12 Customizing ODWEK

In this section, we show a simple way of customizing ODWEK's logon screen. The logon.htm file is the first file presented to the user, as they go to the URL site to logon.

To customize the logon screen, make the following changes:

1. Go to the <HTTP_HOME>\servlet directory.
2. Open logon.htm for edit, and change the statements from:

   ```
   <FORM METHOD=POST ACTION="/scripts/arswww.cgi">
   <b>Server Name:</b> <input type=text name=_server value=everest>
   ```

   To:

   ```
   <FORM METHOD=POST ACTION="/od/arswww">
   <b>Server Name:</b> <input type=text name=_server value=<OD_SERVER>
   ```

   Note: The `FORM` line points to the location of the application where we created in WebSphere. Remember you created a Context root of `/od` and a URL of **`/arswww/`** in WebSphere.

   If we have not changed the FORM line, HTTP would have started at base or its root directory <HTTP_HOME>\servlet and it would have gone one subdirectory down looking for a `scripts` directory; it would not have found the `scripts` directory.

   The modification are in bold. Substitute <OD_SERVER> with your OnDemand server name. In our scenario, it is `wishart`.
3. Save the logon.htm file and exit.

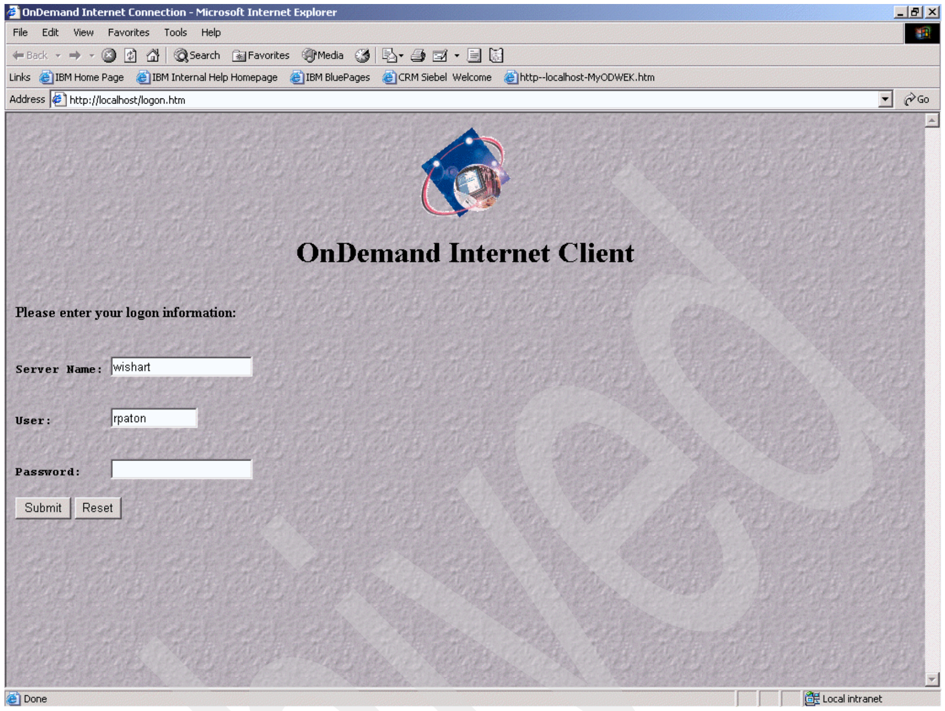
12.3.13 Testing the sample application

To test the ODWEK servlet application, we do the following steps:

1. Open a browser.
2. Enter the following URL:

   ```
   http://<host_name>/logon.htm
   ```

   If you launched the browser on the same machine where your application server is, you can use localhost as <host_name>. Otherwise, substitute <host_name> with the name of your server, such as congo, in our scenario.
3. You should see the logon screen with background image and logo as in Figure 12-42.


OnDemand Internet Connection - Microsoft Internet Explorer

Address http://localhost/logon.htm

OnDemand Internet Client

Please enter your logon information:

Server Name: wishart

User: rpaton

Password:

Figure 12-42 Servlet logon screen

We copy logon.htm under the document root directory, which is <HTTP_HOME>\servlets directory. When the screen appears, it means the system is able to find the logon screen, background and the associated logo images. Now we need to test the servlet, template.htm and connection to your OnDemand server.

4. Enter a Server name, user ID and password.
5. Click **Submit**.
6. You should see the servlet connecting to OnDemand. You know this if you see a list of folders appear on the next screen. You know the template works if you see the background image and logo. The screen should look similar to Figure 12-43. The last thing to test is the line data applet.

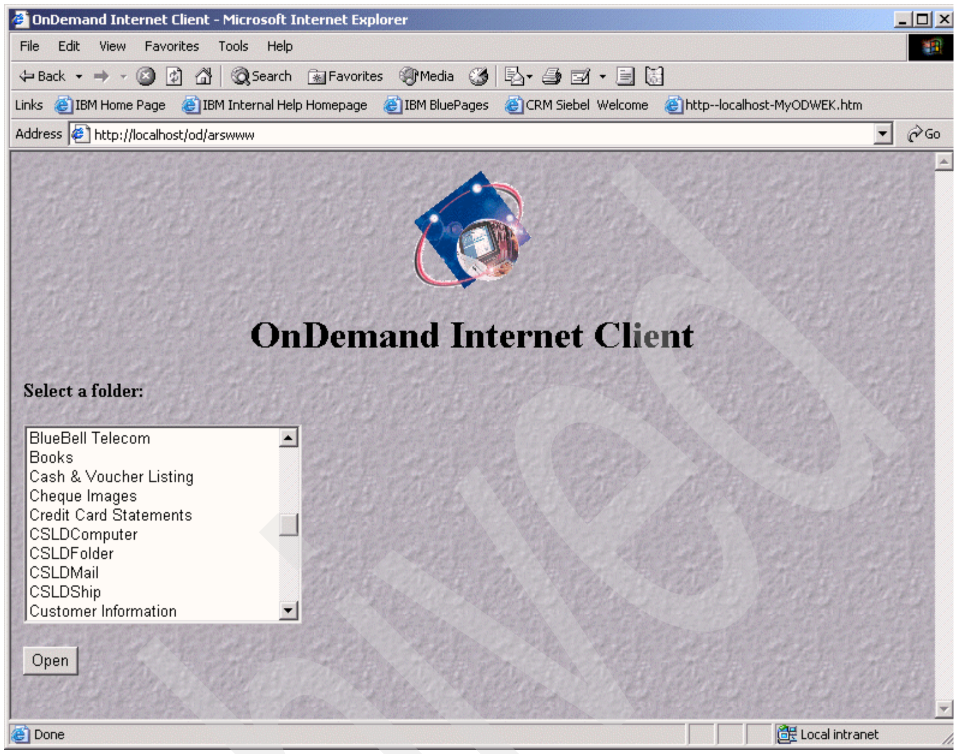

OnDemand Internet Client - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links IBM Home Page IBM Internal Help Homepage IBM BluePages CRM Siebel Welcome http--localhost-MyODWEK.htm
Address http://localhost/od/arswww
Go
OnDemand Internet Client
Select a folder:
BlueBell Telecom
Books
Cash & Voucher Listing
Cheque Images
Credit Card Statements
CSLDComputer
CSLDFolder
CSLDMail
CSLDShip
Customer Information
Open
Done
Local intranet

Figure 12-43 Select a folder

7. Select a folder that you know has a *line data report* in it, and click **Open**.
8. You should see a Search window. Search for a report, and click **Search**.
9. From the Hit List window, select a report.
10. Your report should now appear on a new window with the line data applet. If so, your applet works! If your report does not show, then the applet cannot be found. Refer to 12.2, “Line data applets” on page 416 to see how to configure the system to use the line data applet.

12.4 Conclusion

In this chapter, we showed you how to install the CGI and the servlet version of ODWEK. Without any programming, both of these versions offer the out-of-box Web client that enables users to access and manage OnDemand data.

To customize the client to fulfill your business requirements, you can take several approaches, depending on your business requirements and the resources available to make these changes, which could range from simple modifications to more extensive changes such as rewriting the client piece using ODWEK APIs and/or Information Integrator for Content APIs. See the next chapter for developing an OnDemand Web application using ODWEK APIs.

Building a Web application with ODWEK APIs

In this chapter, we describe how to build a Web application with IBM Content Manager OnDemand Web Enablement Kit (ODWEK).

We cover the following topics:

- ODWEK Java APIs overview
- Set up development environment
- Developing the sample Web application, including:
  - Servlet implementation details
  - Login and logoff implementation
  - Retrieving OnDemand folders implementation
  - Retrieving OnDemand folder reports implementation
- Packaging and deploying Web applications
- Running the sample Web application

13.1 ODWEK Java APIs overview

In this section, we present a brief overview of the IBM Content Manager OnDemand Web Enablement Kit (ODWEK) Java APIs.

To learn more about the APIs, refer to *IBM Content Manager OnDemand for Multiplatforms: Web Enablement Kit Implementation Guide Version 7.1*, SC27-1000.

The ODWEK Java APIs are a set of classes that allow you to access and manipulate data on OnDemand servers. The APIs provide the following capabilities:

- A common object model for data access.
- Search and update across OnDemand servers.
- Client/server implementation for Java application users.

The Java API classes consist of one package, com.ibm.edms.od, which contains the following classes:

ODServer	Represents a connection to an OnDemand server. Use this class to logon, logoff, and change the password. This class also provides several methods to manipulate OnDemand contents.
ODFolder	Represents an OnDemand folder. This object is returned from a successful call to ODServer.openFolder().
ODCriteria	Represents the search criteria for an OnDemand folder. Use this class to retrieve folder's search criteria, criteria operator, and criteria values.
ODHit	Represents an OnDemand document.
ODException	Returns compiled and run-time exceptions that may occur when using the APIs.
ODConstant	Consists of a set of static variables that ODWEK uses and that are also available for you to use in your program.
ODCallback	Used with all methods in which the server operation returns data while processing.
ODNote	Represents an OnDemand annotation.

13.1.1 ODServer class

The *ODServer* class represents a connection to an OnDemand server. It is defined as follows:

```
public class ODServer extends java.lang.Object
implements java.io.Serializable
```

Use an instance of this class to logon, logoff, and update user password on OnDemand servers.

Some of the methods of the ODServer class are:

- ODServer( ) — The class constructor. The following code fragment creates an instance of the ODServer class:

  ```
  ODServer odServer = new ODServer( );
  ```

- void initialize (String configDir, String appName) — Initialize the connection to the OnDemand server. This method must be called after object creation and before the logon method is called. The *configDir* parameter represents the directory where the arswww.ini file exists. The *appName* is the name of your specific program (in our example, the context path concatenated with the RBAppletCallbackOD servlet) or the ODWEK CGI file (arswww.cgi) application name. This is only used when retrieving a document for viewing when one of ODWEK's viewers is invoked. The following code fragment illustrates how to invoke the method:

  ```
  // Set the root directory and get the context path.
  String configDir= "C:\\ODWEK";
  String contextPath = request.getContextPath();
  // Call the initialize method. App name is RBAppletCallbackOD.
  odServer.initialize( configDir, contextPath+"/RBAppletCallbackOD" );
  ```

- void logon (String server, String user, String password) — Logon to the OnDemand server. The *server* parameter represents the name of the OnDemand server. In our example, it is namibia. The *user* and the *password* parameters represent the user name and password.
- void logoff ( ) — Log off from the OnDemand server.
- ODFolder openFolder (String folderName) — Open a specified folder. Note that you must open a folder before querying it.
- Enumeration getFolderNames () — Returns a list of OnDemand folder names accessible to the user.
- byte[ ] viewerPassthru (String queryStr) — Pass ODWEK document viewer requests for data to the ODWEK applet viewer. The *queryStr* parameter, obtained from the request object (for example, request.getQueryString ()), contains the ODWEK document viewer requests for data. The method returns a byte array containing data to be sent back to the ODWEK document viewer and to be displayed to the user. (See 13.3.4, “Retrieving OnDemand folder reports” on page 472 to learn more about this method).

13.1.2 ODFolder class

Use an instance of the *ODFolder* class to manipulate folder criteria information and get a list of folders based on default or specific criteria information.

The ODFolder class represents an OnDemand folder and is defined as follows:

```
public class ODFolder extends ODCallback implements java.io.Serializable
```

The following code fragment creates an instance of the ODServer class and uses the object to logon to OnDemand:

```
// Create an ODServer object
ODServer odServer = new ODServer();
...
...
// Set server name, user ID, and user password.
// Use the logon() method of the ODServer class to connect to OnDemand
String server = “Namibia”;
String userId = “odadmin”;
String userPwd = “odamin”;
odServer.logon(server,userId, userPwd);

// Set the OnDemand folder name
String folderName = “Tax”;
```

Once you have an ODServer object, you can use the openFolder ( ) method of the ODServer object to create an instance of the ODFolder class:

```
// Create an instance of the ODFolder class
ODFolder odFolder = odServer.openFolder(folderName);
```

Some of the methods of the ODFolder class are:

- void close () — Close a folder.
- Object[] getApplGroups () — Get a list of a folder's application groups.
- ODCriteria getCriteria ( ) — Get a list of the search criteria for a folder. The following code fragment illustrates how to use this method:

  ```
  // Create an ODFolder object
  ODFolder odFolder = odServer.openFolder(folderName);
  // Get the criteria associated to the folder that you opened
  Enumeration folderCriteria = odFolder.getCriteria();
  ```

- ODCriteria getCriteria (String name) — Get the criteria object for a specified criteria name.
- String [] getDisplayOrder () — Get the display order for the folder's criteria. Use this method when you want to return to the user the names of the folder’s criteria.

- Vector getHits ( ) — — Get a vector consisting of ODHit objects. Note that an ODHit object represents an OnDemand folder.
- Vector search () — Search the folder using the default search criteria, and returns a vector consisting of ODHit objects.

13.1.3 ODCriteria class

The *ODCriteria* class represents an OnDemand folder criteria and allows you to set the criteria's search operator and search the values of the criteria. It is defined as follows:

```
public class ODCriteria extends java.lang.Object
implements java.io.Serializable
```

The following code fragment creates an ODCriteria object using the getCriteria (String criteriaName) method of an ODFolder object:

```
// The parameter criteriaName is the OnDemand's folder's criteria.
ODCriteria odCriteria = odFolder.getCriteria(criteriaName);
```

Some of the methods of the ODCriteria class are:

- String getName () — Get the name of the criteria. Use this method when you want to display to the user the name of the folder criteria.
- int getOperand () — Get the current search operand. Note that all of the operand types are represented in ODConstant.

13.1.4 ODHit class

The *ODHit* class represents an OnDemand folder's document. It is defined as follows:

```
public class ODHit extends java.lang.Object
implements java.io.Serializable
```

The following statement searches the folder using its default criteria, and then returns a Vector consisting of ODHit objects:

```
Vector odHits = odFolder.search();
```

Some of the methods of the ODHit class are:

- Enumeration getDisplayValues () — Return to the documents display values of the hits that you retrieve from the OnDemand server. Call this method when you are returning to the user a list of OnDemand hits.
- String getDocId () - Get the persistent ID of this document. Once you get the document ID, you can retrieve the document and send it to the user.

- int getDocLocation ( ) — Get the document location.
- byte[ ] retrieve (String viewer) — Returns a byte array containing the document data. However, if your application is using ODWEK applets to view documents, the data returned from this call is an HTML page containing a reference to the ODWEK applet. The *viewer* parameter specifies what kind of ODWEK viewer that you want to use. ODWEK viewers include: APPLET, ASCII, HTML, NATIVE, PDF, PLUGIN, XENOS, or ODServer.viewerPassthru (String queryStr). See 13.3.4, "Retrieving OnDemand folder reports" on page 472 to learn more about this method.

13.2 Set up development environment

Your development environment is based on a successful installation of ODWEK. The very first steps are to install ODWEK and, more importantly, test your installation. The easiest implementation is the CGI. Chapter 12, "ODWEK installation and configuration" on page 387 presents step-by-step instructions on how to install ODWEK. In this section, we summarize the steps so we can present the ODWEK directory structures after the product has been installed.

Install ODWEK and set up the home directory

Install ODWEK and set up the ODWEK home directory as follows:

1. Create the C:\ODWEK directory. Install the software in that directory. Figure 13-1 on page 458 shows the ODWEK directory structure.

   Note that in Chapter 12, "ODWEK installation and configuration" on page 387, all references to the installation are done in D:\ drive. In this chapter, we refer all the installation in C:\ drive.

2. Under the C:\ODWEK directory, create the following subdirectories:

   ```
   templates
   tmp\cache
   ```

3. Copy C:\ODWEK\samples\*.* to C:\ODWEK\templates.
4. Modified C:\ODWEK\arswww.ini and C:\ODWEK\templates\ logon.htm files. See Chapter 12, "ODWEK installation and configuration" on page 387 to learn how to modify these files.
5. Move logon.htm to C:\ODWEK directory.
6. In our example, we use IBM HTTP Server. Modify httpd.conf under <HTTP_HOME>\conf directory, where <HTTP_HOME> is the directory where you installed IBM HTTP server.
7. Test the ODWEK installation.

In our development, we use IBM WebSphere Studio Application Development tool, Version 5.1, exclusively to build our Web application. You can use any development tool of your choice.

To obtain trial version of WebSphere Studio Application Developer:

http://www.ibm.com/software/awdtools/studioappdev

Set up development environment

Set up the development environment as follows:

1. Create the C:\RbOD directory. This is the root directory of the application we are trying to develop.
2. Copy <ODWEK_HOME>\api\ODApi.jar to the directory C:\RbOD\WebContent\WEB-INF\lib.

   <OKWEK_HOME> is where you installed ODWEK. In our scenario, it is C:\ODWEK directory. We use the following command to do the copy:

   ```
   copy C:\ODWEK\api\ODApi.jar C:\RbOD\WebContent\WEB-INF\lib
   ```

   ODApi.jar file contains the ODWEK Java APIs that you need to develop any OnDemand Java application. Copy the file into your application environment.
3. Copy <ODWEK_HOME>\applets\*.* to C:\RbOD\WebContent\applets.

   <ODWEK_HOME> is where you installed ODWEK. In our scenario, execute:

   ```
   copy C:\ODWEK\applets\*.* C:\RbOD\WebContent\applets
   ```
4. Copy <ODWEK_HOME>\images\*.* to C:\RbOD\WebContent\images.
5. Modify the <ODWEK_HOME>\arswww.ini file. You need to modify the file when you write the code to use the ODWEK applet viewer to return line data documents to the user's browser.

   Replace the following lines:

   ```
   ImageDir=/images
   AppletDir=/applets
   ```

   With

   ```
   ImageDir=images
   AppletDir=applets
   ```

Note: Once you modified the lines in arswww.ini file, it works nicely with your developed application, but the ODWEK applet viewer will no longer work in your environment.

If you want to make the changes to arswww.ini *isolated* (which we recommend although we did not do it here), the best way to do this as follows:

1. Move arswww.ini to the Web application class path. In our example, it is C:\RbOD\WebContent\WEB-INF\classes directory. This allows the INI file to be modified without having to interfere with other installations, including other Web applications built with ODWEK.
2. Check the ConfigDir parameter in the web.xml to make sure it is referencing the above directory for the arswww.ini file.

Figure 13-1 shows the directory structure of the Web application that you will develop in this chapter.

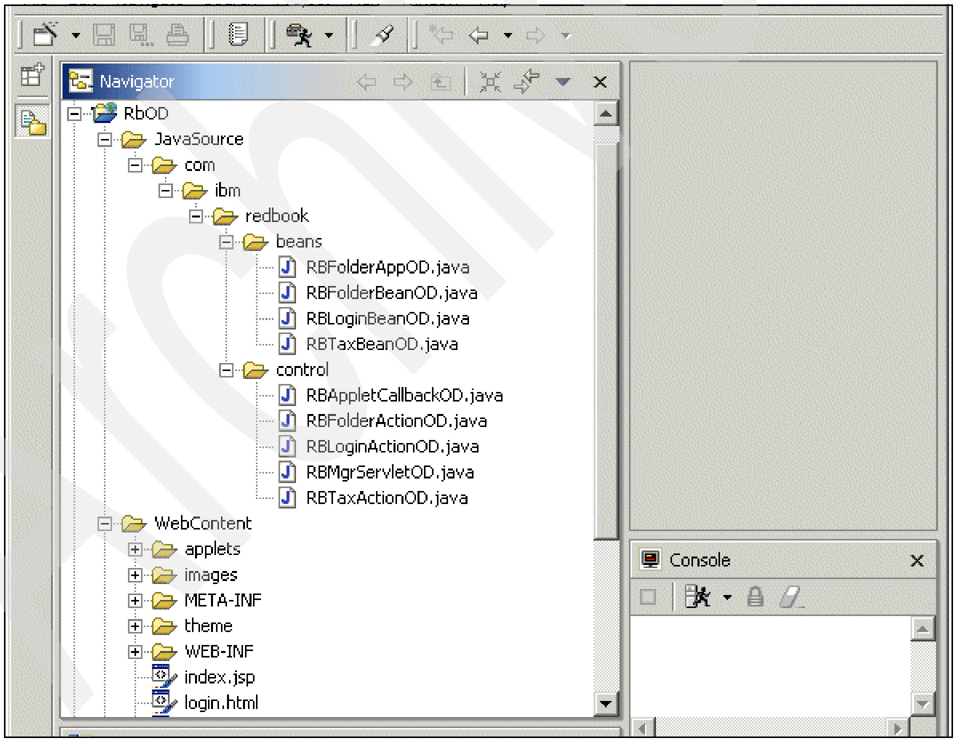


Figure 13-1 Sample OnDemand application directory structure

13.3 Developing the sample Web application

In Chapter 4, “Case study” on page 61, we discuss the case study we use when developing sample Web applications throughout this IBM Redbook. One of the components, *Retrieve Tax & Fee History* sub-system (see 4.4.2, “Web-based application modules” on page 71), provides OnDemand data query and retrieval. In this section, you learn how to develop this module.

The Retrieve Tax and Fee History sub-system enables users to query, retrieve, and view OnDemand reports. It consists of three modules:

- Login — Handle user login. Main servlet: RBLoginServletOD.
- Retrieve Folder Report — Handle retrieving a list of folder reports. Main servlet: RBFolderServletOD.
- Retrieve Tax Discount Report — Handle retrieving a specific report. Main servlet: RBTaxServletOD.

Figure 13-2 shows the architecture for the Retrieve Tax and Fee History sub-system.

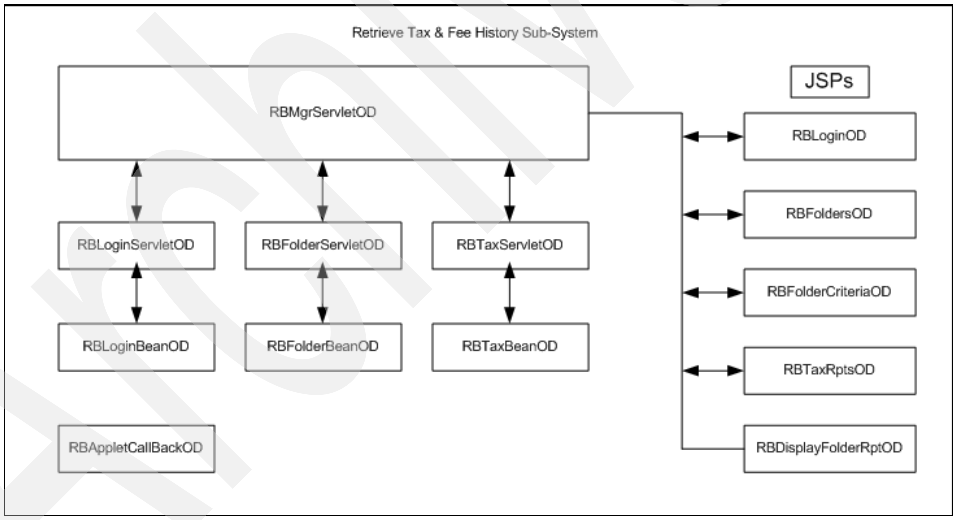


Figure 13-2 Retrieve Tax and Fee History sub-system architecture

Figure 13-3 shows how the Login, Retrieve Folder Report, and Retrieve Tax Discount Report application modules work together using sequence diagrams.

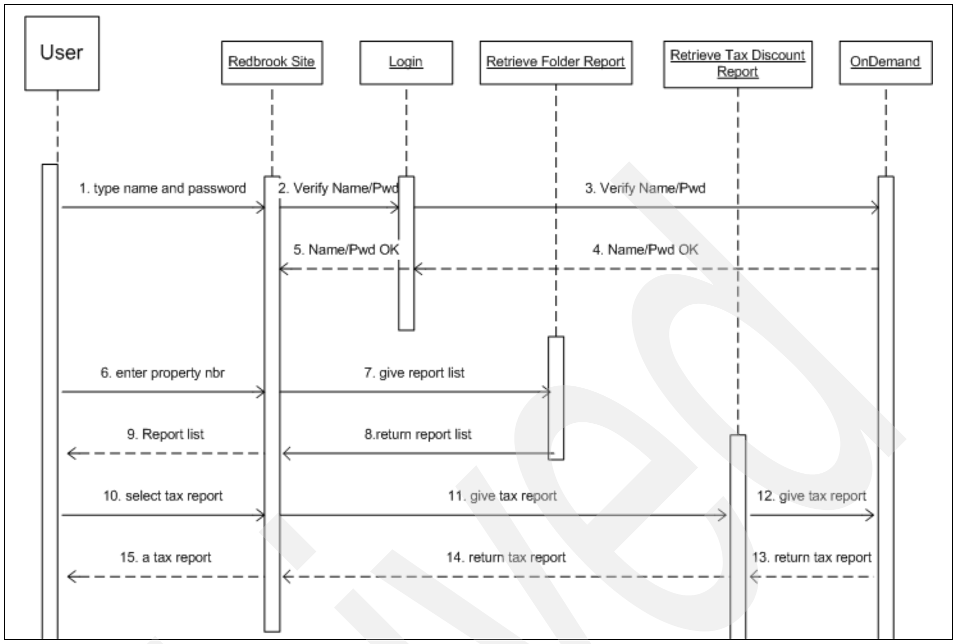


Figure 13-3 Login, Retrieve Folder, and Retrieve Report working together

There are several files (Figure 13-2) involved in implementing the Retrieve Tax & Fee History Sub-System. They are as follows:

- RBMgrServletOD.java — Controller servlet that manages contents on OnDemand servers.
- RBLoginActionOD.java — Java class that manages OnDemand connections.
- RBFolderActionOD.java — Java class that manages folder reports.
- RBTaxActionOD.java — Java class that manages tax discount reports.
- RBAppletCallbackOD.java — Servlet that handles OnDemand data callback from the ODWEK applet. To learn more about this servlet, see 13.3.2, "Login and logoff" on page 463.
- RBFolderAppOD.java — Java bean that manipulates OnDemand folder applications and reports.
- RBLoginBeanOD.java — Java bean that handles OnDemand connections.
- RBFolderBeanOD.java — Java bean that manipulates OnDemand folders.
- RBTaxBeanOD.java — Java bean that handles the Tax Discount reports.
- RBLoginOD.jsp — JavaServer Pages (JSP) that displays the login screen.
- RBFoldersOD.jsp — JSP that displays a list of OnDemand folders.

- RBFolderCriteriaOD — JSP that displays an OnDemand folder's criteria.
- RBTaxRptsOD — JSP that displays a list of Tax reports.
- RBDisplayFolderRptOD — JSP that handles the requests for the ODWEK applet viewer.

While developing the sample Web application, you learn the following topics:

- How to login to OnDemand and logoff from OnDemand. See 13.3.2, "Login and logoff" on page 463.
- How to retrieve a list of OnDemand folders and a list of search criteria for an OnDemand folder. See 13.3.3, "Retrieving OnDemand folders" on page 466.
- How to use a folder search criteria to retrieve OnDemand folder's reports. See 13.3.4, "Retrieving OnDemand folder reports" on page 472.
- How to use ODWEK's applet viewer to retrieve line data reports in a browser. See 13.3.4, "Retrieving OnDemand folder reports" on page 472.

13.3.1 The RBMgrServlet implementation details

Let us take a closer look on the RBMgrServletOD servlet controller. The RBMgrServletOD class performs the following tasks:

- Manages user requests, that is, input from the browser.
- Delegates business processing, that is, calls the appropriate methods using the RBLoginActionOD, RBFolderActionOD, or the RBTaxActionOD Java classes.
- Manages the choice of an appropriate view, that is, calls the appropriate JSP program such as RBFoldersOD, RBFolderCriteriaOD, RBTaxRptsOD, or RBDisplayFolderRptOD.
- Handles errors. In most cases, the controller handles error messages. It does this by calling a JSP error page, which returns the appropriate error message to the user. In our sample application, we elected not to do so. We leave it as an exercise for the reader.

The RBMgrServletOD servlet controller consists of the following methods:

- public void init (ServletConfig config) — Called by the Web container to initialize the servlet instance.
- public synchronized void doPost (HttpServletRequest request, HttpServletResponse response) — Execute the application actions.
- Calls the doGet(request, response) method.
- public synchronized void doGet (HttpServletRequest request, HttpServletResponse response) — Calls doPost() method.

Example 13-1shows the doPost() code snippet of the RBMgrServletOD servlet controller.

Example 13-1 RBMgrServlet.java - Code snippet for doPost() method

```
public synchronized void doPost(HttpServletRequest request, HttpServletResponse 
response) throws ServletException, IOException {
      try {

      String responseJSP = null;
      String action = request.getParameter("cmd");

      if  (action.equals("login")) {
         RBLoginActionOD.doLoginOD(request);
         responseJSP="RBFoldersOD.jsp";
         RequestDispatcher rd = 
   getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
         rd.forward(request, response);

      } else if (action.equals("criteria")){
         String folderName = request.getParameter("OnDemand_Folders");
         RBFolderActionOD.getFolderCriteria(request);
         responseJSP="RBFolderCriteriaOD.jsp";
         RequestDispatcher rd = 
   getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
         rd.forward(request, response);

      } else if (action.equals("odHit")){
         System.out.println(" OdHit!!!");
         RBTaxActionOD.getODApplications(request);
         responseJSP= "RBTaxRptsOD.jsp";
         RequestDispatcher rd = 
   getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
         rd.forward(request, response);

      }else if (action.equals("displayDoc")){
         RBTaxActionOD.getFolderDoc(request);
         responseJSP= "RBDisplayFolderRptOD.jsp";
         RequestDispatcher rd = 
   getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
         rd.forward(request, response);

...
```

The source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

Study the source code. Make sure you understand it before you continue.

Next, we present the implementation details the RBMgrServletOD controller handles. Let us start with the Login module.

13.3.2 Login and logoff

In this section, we show you how to login and out of OnDemand server with the *Login* module. This module consists of the following files:

- RBLoginActionOD.java (Java class)
- RBLoginBeanOD.java (Java bean)
- RBLoginOD.jsp (JSP)

RBLoginActionOD.java (Java class)

The RBLoginActionOD class allows a user to login to OnDemand and log off. It contains one method that the RBMgrServletOD invokes when it receives a login action:

- public static void doLoginOD (HttpServletRequest request) — This method calls two methods of the RBLoginBeanOD Java bean:
  - RBLoginBeanOD.loginOD(...) — Logs in the user to the OnDemand server and returns to the caller an instance of an ODServer class, which is stored in the session object.
  - RBLoginBeanOD.getFolderNames(...) — Returns to the caller a list of OnDemand folders. The

Both the ODServer object and the Enumeration object (consisting of a list of OnDemand folder names) are stored in the session object. The following code fragment illustrates how to store these objects in the session:

```
session.setAttribute("anODServer", odServer);
session.setAttribute("OD_FOLDERS", folderNames);
```

Example 13-2 shows the doLoginOD() code snippet of the RBLoginActionOD Java class.

Example 13-2 RBLoginActionOD.java - Code snippet for doLoginOD method

```
public class RBLoginActionOD implements ODConstant {

   public static void doLoginOD (HttpServletRequest request)
         throws ServletException, IOException, ODException {
         RBLoginBeanOD rbloginbeanOD = new RBLoginBeanOD();
         RBFolderBeanOD rbFolderBeanOD = new RBFolderBeanOD();

         HttpSession session = request.getSession(true);

         String configDir= "C:\\ODWEK";
         String server   = request.getParameter("server");
         String userName = request.getParameter("user");
         String userPwd  = request.getParameter("pwd");

         String contextPath = request.getContextPath();
         ODServer odServer =
       rbloginbeanOD.loginOD(contextPath, configDir, server, userName,
userPwd,"");

         Enumeration folderNames = rbFolderBeanOD.getFolderNames (odServer);
         session.setAttribute("anODServer", odServer);
         session.setAttribute("OD_FOLDERS", folderNames);
   }  // End of manageFolderOD()

} // End of RBLoginActionOD
```

The source code of the file is provided as the additional material for this redbook. See Appendix C, "Additional material" on page 589 for download instructions.

RBLoginBeanOD.java (Java bean)

The steps to login on an OnDemand server are as follows:

1. Create an ODServer object:

   `ODServer odServer = new ODServer();`

2. Initialize the connection:

   `odServer.initialize( ... );`

3. Set the server to OnDemand:

   `odServer.setServer(server);`

4. Set the user name:

   `odServer.setUserId(userId );`

5. Set the user password:

   `odServer.setPassword(userPwd );`

6. Call the logon(...) method to connect to the server:

   `odServer.logon(...)`

You learn the steps to login to OnDemand while implementing the RBLoginBeanOD class.

The RBLoginBeanOD class contains the following methods:

- public ODServer loginOD (String contextPath, String configDir, String server, String userId, String userPwd, String localDir) — Login to OnDemand.

In this method, we call the initialize ( ) method of the ODServer to initialize the OnDemand connection. Note that the contents of the configDir and contextPath parameters are set in the RBLoginActionOD class. These parameters have to be properly set if you want to invoke any of the ODWEK viewers, specifically, when calling the ODWEK applet viewer.

The code snippet of the loginOD() method is as follows:

```
odServer = new ODServer( );
odServer.initialize( configDir, contextPath+"/RBAppletCallbackOD" );

odServer.setServer(server);
odServer.setUserId(userId);
odServer.setPassword(userPwd);

// Log on to OD server
odServer.logon(server, userId, userPwd);
```

In our example, our application name is RBAppletCallbackOD. Note that this is only used when retrieving a document for viewing and invoking one of the ODWEK's applet viewers.

- public void logoutOD (ODServer odServer) — Logoff from OnDemand using provided ODWEK APIs.

  The code snippet of the method is as follows:

```
odServer.logoff( );
odServer.terminate( );
```

The source code of the file is provided as the additional material for this redbook. See Appendix C, "Additional material" on page 589 for download instructions.

RBLoginOD.jsp (JSP)

RBLoginOD.jsp is the first program that the user invokes to run the application. From a browser, the user accesses the application by entering the following URL address:

```
http://localhost:9080/RbOD/RBLoginOD.jsp
```

The RBLoginOD displays a screen that allows a user to enter the OnDemand server name, user name, and password. Figure 13-4 on page 481 shows the login screen.

Example 13-3 shows a code snippet of the RBLoginOD.jsp file.

Example 13-3 RBLoginOD.jsp - Code snippet

```
<form name="input" action="RBMgrServletOD" method="get">
<H1>Please enter your logon information:</H1>
<TABLE border="0">
```

```
   <TBODY>
      <TR>
         <TD>ODServer:</TD>
         <TD><INPUT type="text" name="server"></TD>
      </TR>
      <TR>
         <TD>Username:</TD>
         <TD><INPUT type="text" name="user"></TD>
      </TR>
      <TR>
         <TD>Password: </TD>
         <TD><input type="password" name="pwd"></TD>
      </TR>
      <TR>
         <TD colspan="2" align="right"><input type="hidden" name="cmd"
value="login">
<input type="submit" value="Login"></TD>

      </TR>

   </TBODY>
</TABLE>
<P>
</form>
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, "Additional material" on page 589 for download instructions.

13.3.3 Retrieving OnDemand folders

In this section, we show you how to manipulate OnDemand folders with the *Retrieve Folder Report* module. This module consists of the following class, bean, and JSP files:

- RBFolderActionOD.java (Java class)
- RBFolderBeanOD.java (Java bean)
- RBFolderCriteriaOD.jsp (JSP)
- RBFoldersOD.jsp (JSP)

RBFolderActionOD.java

The RBFolderActionOD class manages user requests to access OnDemand folders and their associated default criteria as defined in the OnDemand folder. The methods of the RBFolderActionOD class are invoked by the RBMgrServletOD servlet controller when it receives a criteria action.

Figure 13-5 on page 481 and Figure 13-6 on page 482 show what the system returns to the user when RBMgrServletOD servlet controller invokes one of the method of the RBFolderActionOD class.

This class uses the RBFolderBeanOD Java bean to retrieve folders accessible to the user as well as folder's criteria information. The RBFolderActionOD consists of the following methods:

- public static void getFoldersOD (HttpServletRequest request) — Calls the RbFolderBeanOD.getFolderNames ( ) method to get a list of OnDemand folders and stores that list in a session object. The code snippet of the method is as follows:

  ```
  HttpSession session = request.getSession(true);
  ODServer odServer = (ODServer)session.getAttribute("anODServer");

  RBFolderBeanOD rbFolderBeanOD = new RBFolderBeanOD();
  Enumeration folderNames = rbFolderBeanOD.getFolderNames(odServer);

  session.setAttribute("OD_FOLDERS",folderNames);
  ```

- public static void getFolderCriteria (HttpServletRequest request) — Retrieves the criteria information for a specific folder and stores the ODCriteria object in a session object. Example 13-4 shows a code snippet of the getFolderCriteria method.

Example 13-4 RBFolderActionOD.java - Code snippet for getGolderCriteria method

```
public class RBFolderServletOD implements ODConstant {

...

   public static void getFolderCriteria (HttpServletRequest request)
      throws ServletException, IOException, ODException {
      try {
         HttpSession session = request.getSession(true);

         ODServer odServer = (ODServer)session.getAttribute("anODServer");

         RBFolderBeanOD rbFolderBeanOD = new RBFolderBeanOD();

         // Get the name of the folder the user has selected.
         String folderName = request.getParameter("OnDemand_Folders");

         // Call the RBFolderBeanOD.setFolderOD () to open the folder
         rbFolderBeanOD.setFolderOD(odServer,folderName);

         session.setAttribute("OD_FOLDER",
rbFolderBeanOD.getFolderOD(odServer));
```

```
         Object[] appGrps =
(rbFolderBeanOD.getFolderOD(odServer)).getApplGroups();

         for (int i = 0; i < appGrps.length; i++){
            System.out.println("Application Group = "
                       +appGrps[i].toString());
         }


         System.out.println("I am inside RBFolderServletOD getFolderCriteria
!!!");

         // Save folder name
         session.setAttribute("OD_FOLDER_NAME",folderName);

         Enumeration folderCriteria =
                     rbFolderBeanOD.getFolderCriteria(odServer, folderName);


         // Get criteria objects
         ArrayList odCriteria = new ArrayList();
         for ( Enumeration e = folderCriteria;
                folderCriteria.hasMoreElements();)
         {
            odCriteria.add((ODCriteria)e.nextElement());
         }  // End of for loop

         odCriteria.trimToSize();

         // Save OD folder criteria
         session.setAttribute("OD_FOLDER_CRITERIA",odCriteria);

         System.out.println("# of criteria objects = " +odCriteria.size());

...
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBFolderBeanOD.java (Java bean)

Use the RBFolderBeanOD class to manipulate OnDemand Folder contents. This class returns the following information to the caller:

- A list of OnDemand folders that the user is authorized to access
- A list of ODCriteria objects associated with a specific folder
- The name of folder criteria

The RBFolderBeanOD class consists of the following methods:

- public Enumeration getFolderNames (ODServer odServer) — Returns to the caller an Enumeration object consisting of OnDemand folder names. It prepares the information for the RBFoldersOD.jsp file. The main code is as follows:

```
Enumeration folderNames = null;
folderNames = odServer.getFolderNames();
return folderNames;
```

- public ODFolder getFolderOD (ODServer odServer) — Returns to the caller an ODFolder object. The main code snippet:

```
return odFolder;
```

- public void setFolderOD (ODServer odServer, String folderName) — Creates an ODFolder object. The main code snippet:

```
odFolder = odServer.openFolder(folderName);
```

- public Enumeration getFolderCriteria (ODServer odServer, String folderName) — Returns to the caller an Enumeration object consisting of ODCriteria objects. The main code snippet:

```
if  (odFolder == null)
   setFolderOD(odServer,folderName);
folderCriteria = odFolder.getCriteria();
return folderCriteria;
```

- public static String getOperatorName (ODCriteria odCriteria) — Converts the operand of a ODCriteria object into a user friendly output. The main code snippet:

```
oper = odCriteria.getOperand();
switch( oper ) {
   case ODConstant.OPEqual:
      operatorName = "Equal To";
      break;
   ...
   case ODConstant.OPNotBetween:
      operatorName = "Not Between";
      break;
   default:
      operatorName = "*** Unknown operator";
      break;
}
return operatorName;
```

- public void closeFolder () — Closes a previously opened OnDemand folder. The main code snippet:

```
if  (odFolder != null)
   odFolder.close();
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBFolderOD.jsp (JSP)

This JSP file retrieves from the session object an Enumeration object consisting of OnDemand folder names and returns the list of names to the user.

Example 13-5 shows a code snippet of the RBFoldersOD.jsp.

Example 13-5 RBFoldersOD.jsp - Code snippet

```
<%@ page import="com.ibm.redbook.beans.*,
      com.ibm.edms.od.*, java.util.*" %>

<jsp:useBean id="folders"
   scope="session" class="com.ibm.redbook.beans.RBFolderBeanOD" />


<%
   Enumeration folderNames = (Enumeration)session.getAttribute("OD_FOLDERS");
   System.out.println("I am inside RBFoldersOD !!!");

   if  (folderNames == null) System.out.println(" null object !!!");
 %>
  <FORM name="input" action="RBMgrServletOD" method="get">
  <h4>Select a folder:</h4>
  <SELECT NAME="OnDemand_Folders" size=5>
  <%  for (Enumeration e = folderNames;folderNames.hasMoreElements(); )
       { String name = (String)e.nextElement(); %>
         <OPTION VALUE="<%=name%>"> <%=name %>
     <% } %>
  </SELECT>
  <input type="hidden" name="cmd" value="criteria">
  <p>
  <input type="submit" value="open">
 </FORM>

...
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBFolderCriteriaOD.jsp (JSP)

This JSP file retrieves from the session object an *ArrayList* object consisting of ODCriteria objects and returns the criteria to the browser.

Example 13-6 shows a code snippet of the RBFolderCriteriaOD.jsp file.

Example 13-6 RBFolderCriteriaOD.jsp - Code snippet

```
<%@ page import="com.ibm.redbook.beans.*,
      com.ibm.edms.od.*, java.util.*" %>

<jsp:useBean id="folders"
   scope="session" class="com.ibm.redbook.beans.RBFolderBeanOD" />

<%
   ArrayList folderCriteria =
(ArrayList)session.getAttribute("OD_FOLDER_CRITERIA");
   System.out.println("I am inside RBFolderCriteriaOD !!!");

   if  (folderCriteria == null) System.out.println(" folderCriteria is null
!!!");
 %>

<center><h3><%=(String)session.getAttribute("OD_FOLDER_NAME") %></h3></center>
<FORM name="input" action="RBMgrServletOD" method="get">
<TABLE BORDER=1 CELLSPACING=3 CELLPADDING=3>

  <% for (int i = 0; i < folderCriteria.size(); i++)
     { ODCriteria aCriteria = (ODCriteria)folderCriteria.get(i);
       String criteriaName =  aCriteria.getName();
       int operator = aCriteria.getOperand();
       String operatorCriteria =  RBFolderBeanOD.getOperatorName(aCriteria);
        %>
       <TR>
       <TD NOWRAP><FONT SIZE=3><b><%=criteriaName %></b></FONT></td>
       <TD NOWRAP><FONT SIZE=2><i><%=operatorCriteria %></i></FONT></td>
       <TD NOWRAP>

      <%
       Vector currentValues = null;
       if  (aCriteria.getOperand() == ODConstant.OPBetween) {
          currentValues = aCriteria.getValues();
          %>
          <TABLE BORDER=1 CELLSPACING=0 CELLPADDING=0>
             <TR>
             <TD NOWRAP>
               <input type=text name="fromDate"
                           value="<%=currentValues.elementAt(0) %>"
maxlength=254>
             </td>
             <TD><i>and</i>
             </TD>
             <TD>
```

```
                <input type=text name="toDate"
                            value="<%=currentValues.elementAt(1) %>"
maxlength=254>

              </TD>
              </TR>
           </TABLE>
       <% } else { %>

                <input type=text name="<%=criteriaName %>"
                        maxlength=254>

       <% } %>


        </TD>
        </TR>
   <% } %>
</TABLE><br>
<input type=submit value="Search">
<input type=reset value="Reset">
<input type="hidden" name="cmd" value="odHit">
</FORM>
...
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

13.3.4 Retrieving OnDemand folder reports

In this section, you will learn how to retrieve OnDemand line data documents using the ODWEK applet viewer. The *Retrieve Tax Discount* module consists of the following class, Java bean, and JSP files:

- RBTaxActionOD.java (Java class)
- RBTaxBeanOD.java (Java bean)
- RBTaxRptsOD.jsp (JSP)
- RBAppletCallbackOD.java (servlet)

RBTaxActionOD.java (Java class)

The RBTaxActionOD program is a class that manages OnDemand folder documents. In our example, we are using it to retrieve documents from the Tax Discount folder. But the RBTaxActionOD class can be used to retrieve any line data documents. The RBMgrServletOD invokes the RBTaxActionOD class when it receives an odHit or displayDoc action.

The RBTaxActionOD class consists of the following methods:

- public static void getODApplications (HttpServletRequest request) — Get a list of ODHit objects. A ODHit object represents a folder document.
- public static void getFolderDoc (HttpServletRequest request) — Retrieve a specific ODHit based on the document ID.

Example 13-7 shows a code snippet of the RBTaxActionOD.java file.

Example 13-7 RBTaxActionOD.java - Code snippet

```
public class RBTaxActionOD implements ODConstant {

   public static void getODApplications (HttpServletRequest request)
       throws ServletException, IOException, ODException
   {
      Enumeration inputParameters = request.getParameterNames();

        try
        {
          // Create an instance of RBTaxBeanOD JavaBean
          RBTaxBeanOD rbTaxBeanOD = new RBTaxBeanOD();

          HttpSession session = request.getSession(true);

          // Get an ODServer object
          ODServer odServer = (ODServer)session.getAttribute("anODServer");

          // Get the OD folder name
          String folderName = (String)session.getAttribute("OD_FOLDER_NAME");

          // Get ODHit objects that belong to folderName
          // Using the default criteria
          Vector odHits = rbTaxBeanOD.getHitList(odServer, folderName);

   /*  Start Testing (ODHit)odHits.elementAt(i)).getDisplayValues()
          for (int i = 0; i < odHits.size(); i++)
          {

             for (Enumeration e =
((ODHit)odHits.elementAt(i)).getDisplayValues() ; e.hasMoreElements();)
             {

                String parm = (String)e.nextElement();
                System.out.println("   Display Parameter = " +parm);

             } // End of inner for loop

          }  // End of outer for loop
```

```
  End Testing (ODHit)odHits.elementAt(i)).getDisplayValues() */

         // Save the ODHit objects
         session.setAttribute("OD_APPLICATIONS",odHits);

       } catch (Exception ex) {
      ex.printStackTrace();
       }

   }  // End of getODApplications()

   public static void getFolderDoc (HttpServletRequest request)
         throws ServletException, IOException, ODException
   {
      // Create an instance of RBTaxBeanOD JavaBean
      RBTaxBeanOD rbTaxBeanOD = new RBTaxBeanOD();
      try
      {
         HttpSession session = request.getSession(true);

          // Get the hit list from the session object
         Vector odHits = (Vector)session.getAttribute("OD_APPLICATIONS");

         // Get the docID from the request object
//  Find the ODHit object corresponding to the docId
         for (int i = 0; i < odHits.size(); i++)
         {
           ODHit odHit = ((ODHit)odHits.elementAt(i));

           if  ((odHit.getDocId()).equals(docId)){

               session.setAttribute("FOLDER_RPT",
rbTaxBeanOD.getfolderReport(odHit));
break;

           }  // End if

         } // End for loop

      } catch (Exception ex) {
          ex.printStackTrace();
       }

   } // End of getFolderDoc()
} // End of RBTaxActionOD
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBTaxBeanOD.java (Java bean)

The purpose of the RBTaxBeanOD class is to deliver to the caller a list of OnDemand documents using a folder’s default criteria or user’s criteria. The class also allows you to retrieve a specific folder’s document.

The RBTaxBeanOD class contains the following methods:

- public Vector getHitList (ODServer odServer, String folderName) — Get a vector object consisting of ODHit objects for a specific folder using the default criteria of that folder. A ODHit object represents an OnDemand folder’s document. The main code snippet is as follows:

```
// Open the folder
odFolder = odServer.openFolder(folderName);

// Search the folder using default criteria
odHits = odFolder.search();

...
// Return the hit list.
return odHits;
```

- public Vector getHitList (ODServer odServer, String folderName, String criteriaName, String criteriaValue1, String criteriaValue2) — Get a vector object consisting of ODHit objects. The difference between this method and the previous method is that this method uses a specific search criteria and values. The main code snippet is as follows:

```
// Open the folder
odFolder = odServer.openFolder(folderName);

// Get an ODCriteria object
odCriteria = odFolder.getCriteria(criteriaName);

// Set the search values using an instance of ODCriteria
if  (criteriaValue2 == null)
   odCriteria.setSearchValue(criteriaValue1);
else
   odCriteria.setSearchValues(criteriaValue1, criteriaValue2);

// Search the folder using default criteria
odHits = odFolder.search();
...
return odHits;
```

- public static String getDocId (ODHit odHit, String criteriaName) — Get the document ID for a specific ODHit object. The main code snippet is as follows:

  ```
  docId = odHit.getDocId();
  return docId;
  ```

- public static String getParameterValues(ODHit odHit, String criteriaName) — Get the display values for a specific document. The main code snippet is as follows:

  ```
  if  (criteriaName.equals("cmd"))
      return null;
  displayValue = odHit.getDisplayValue(criteriaName);

  ...
  return displayValue;
  ```

- public byte [ ] getfolderReport (ODHit odHit) — Invoke the retrieve (java.lang.String viewer) method of the ODHit class. The main code snippet is as follows:

  ```
  folderRpt = odHit.retrieve(ODConstant.APPLET);
  return folderRpt;
  ```

  The OnDemand viewer can be of several types APPLET, ASCII, HTML, NATIVE, PDF, PLUGIN, and XENOS. In our sample, we use APPLET, which instructs ODWEK that we want to use the ODWEK applet viewer. When we use APPLET, the data returned from this call is an HTML page containing a reference to the ODWEK applet and not the document itself. To obtain the actual document, you must invoke the provide a callback mechanism to the ODWEK applet viewer. You do so by writing a program that invokes the viewerPassthru (java.lang.String queryStr) method of the ODServer class where the queryStr parameter contains the query string that the ODWEK applet viewer had sent. In our sample, our applet callback code is the RBAppletCallbackOD servlet.

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBTaxRptsOD.jsp (JSP)

The RBTaxRptsOD.jsp file retrieves from the session object an vector object consisting of ODHit objects and returns the documents’ display values to the browser. This class also invokes the getDisplayOrder( ) method of the ODFolder class to retrieve the display order for the folder's criteria.

Example 13-8 shows a code snippet of the RBTaxRptsOD.jsp file.

Example 13-8 RBTaxRptsOD.jsp - Code snippet

```
<%@ page import="com.ibm.redbook.beans.*,
      com.ibm.edms.od.*, java.util.*" %>

<% Vector odHits = (Vector)session.getAttribute("OD_APPLICATIONS");

   System.out.println("I am inside RBTaxRptsOD !!!");

   if  (odHits == null) System.out.println(" odHits is null !!!");

   String[] displayOrder =
((ODFolder)session.getAttribute("OD_FOLDER")).getDisplayOrder();
%>
<center><h3><%=(String)session.getAttribute("OD_FOLDER_NAME") %></h3></center>
<FORM name="input" action="RBMgrServletOD" method="get">
<center><table border="2" cellspacing="3" cellpadding="3">
<tr>
<% for (int i = 0; i < displayOrder.length; i++)
   { %>
      <th align="center"><%=displayOrder[i] %></th>
   <% } %>
</tr>
  <% for (int i = 0; i < odHits.size(); i++)
     { String docID = ((ODHit)odHits.elementAt(i)).getDocId(); %>
   <TR>

       <% for (Enumeration e = ((ODHit)odHits.elementAt(i)).getDisplayValues()
;
                  e.hasMoreElements();)
          { String parm = (String)e.nextElement(); %>
     <TD>
       <A
href="RBMgrServletOD?cmd=displayDoc&docID=<%=java.net.URLEncoder.encode(docID)%
>"><%=parm%></A>
          </TD>
          <% } %>
        </TR>
  <% } %>
</TABLE><br>
</form>

...
...
...
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBDisplayFolderRptOD.jsp (JSP)

The RBDisplayFolderRptOD.jsp file retrieves from the session object an HTML page containing a reference to the ODWEK applet and display it.

Example 13-9 shows a code snippet of the RBDisplayFolderRptOD.jsp file.

Example 13-9 RBDisplayFolderRptOD.jsp - Code

```
<%@ page import="com.ibm.redbook.beans.*,
      com.ibm.edms.od.*, java.util.*" %>

<!--   Get the OnDemand folder report  -->
<%
   byte[] theDoc = (byte[])session.getAttribute("FOLDER_RPT");
   out.print(new String(theDoc));
%>
...
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

RBAppletCallbackOD.java (servlet)

The RBAppletCallbackOD servlet controller handles OnDemand data callback from the applet. This servlet is needed by the ODWEK applet viewer. RBAppletCallbackOD retrieves the ODServer object from the session, and calls the viewPassthru ( ) method of the ODServer class, with the request's query string as a parameter. The viewerPassthru ( ) method is used to pass ODWEK document viewer data requests through to the ODWEK code. The returned byte array content from the call is then streamed back to the applet, the actual program that uses the RBAppletCallbackOD servlet controller.

The RBAppletCallbackOD servlet consists of the following methods:

- public void init (ServletConfig config) — Invoked by the Web container to initialize the servlet.
- public synchronized void doPost (httpServletRequest request, HttpServletResponse response) — Calls the doGet (request, response) method.

- public synchronized void doGet (HttpServletRequest request, HttpServletResponse response) — Calls the viewerPassthru ( ) method of the ODServer class. The queryStr parameter, obtained from the request object (for example, request.getQueryString ()), contains the ODWEK document viewer data requests. The method returns a byte array containing data to be sent back to the ODWEK document viewer and to be displayed to the user.

Example 13-10 shows the doGet() code snippet of the RBAppletCallbackOD servlet controller.

Example 13-10 RBAppletCallbackOD.java - Code snippet for doGet method

```
public class RBAppletCallbackOD extends HttpServlet implements ODConstant {

...

   public synchronized void doGet(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

      ODServer odServer = null;
      byte[] results = null;

      try {
         odServer = (ODServer)

         request.getSession().getAttribute("anODServer");
         results = odServer.viewerPassthru(request.getQueryString());
         OutputStream outputStream = response.getOutputStream();
         outputStream.write(results);
      catch (Exception ode) {
         //handle exception here...
      }
   }
} // End of RBAppletCallbackOD
```

The complete source code of the file is provided as the additional material for this redbook. See Appendix C, “Additional material” on page 589 for download instructions.

13.4 Packaging and deploying Web applications

You cannot deploy an ODWEK application in a non-ODWEK environment. You need to install and test ODWEK before you deploy your ODWEK Web application.

The steps to package and deploy your Web application are as follows:

1. Modify web.xml file by adding the following:

```
<servlet-class>com.ibm.redbook.control.RBMgrServletOD</servlet-class>
<!-- After the above, add the following entries to the web.xml file
and before the end of the servlet tag -->

<init-param>
  <param-name>ConfigDir</param-name>
  <param-value>c:\ODWEK</param-value>
  </init-param>
<!-- Before the servlet end tag
  </servlet>
```

2. Create a WAR file for your application.
3. Create a EAR file for your application.
4. Use the EAR file to deploy your application into your Application Server.

We have created the RbOD.ear file for you and is ready for you to use. Once you deploy the application in your environment, you need to modify the <init-para> to reflect your application server's environment.

13.5 Running the sample Web application

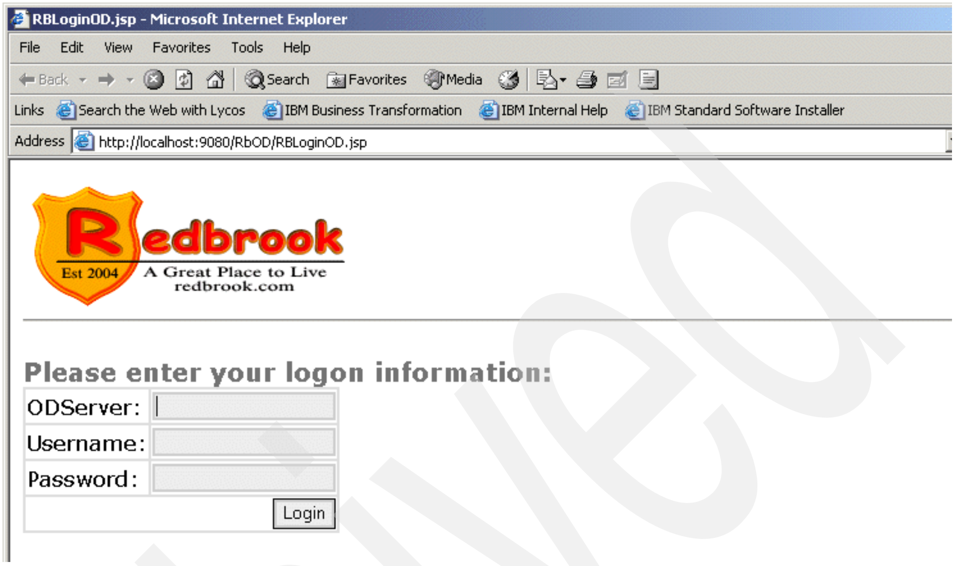
In the previous sections, you learned how to implement, package, and deploy the Web application. In this section, we show you how to run the application. The steps to run the application is as follows:

1. In a browser, type the following URL address:

   `http://localhost:9080/RbOD/RBLoginOD.jsp`

2. Figure 13-4 shows the login screen. Enter the server name, user name, and password, and then click **Login**.
3. A list of OnDemand folders appears as shown in Figure 13-5. Scroll down the list and select **Tax Discount** and click **Open**.
4. The Tax Discount folder appears as shown in Figure 13-6. Accept the default search criteria and click **Search**.
5. A list of reports that belongs to the Tax Discount folder appears as shown in Figure 13-7. Note that the list of reports is based on the default search criteria shown in Figure 13-6. Select a report by clicking a link associated with the one you want to see.
6. The system security warning window appears (Figure 13-8). Click **Yes** to continue.

7. A line data applet viewer appears as shown Figure 13-9.
8. Close the report window when you are done.


RBLoginOD.jsp - Microsoft Internet Explorer

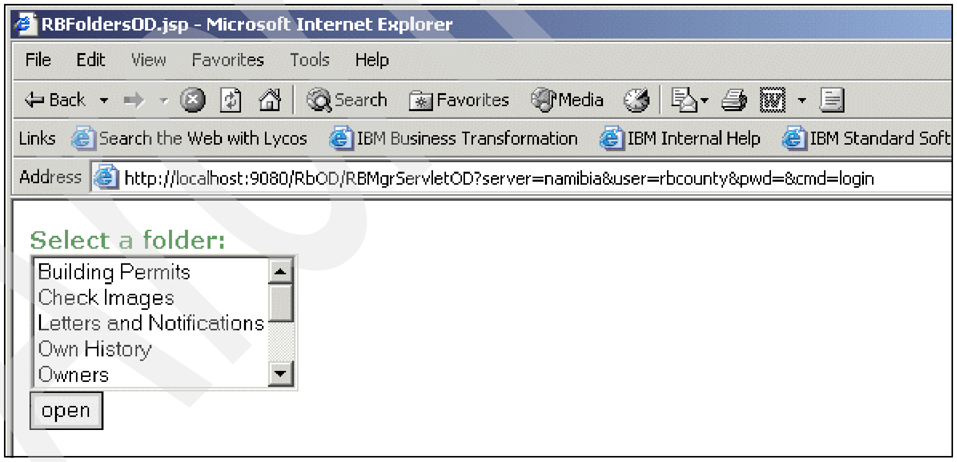
Address http://localhost:9080/RbOD/RBLoginOD.jsp

File Edit View Favorites Tools Help
Back Search Favorites Media
Links Search the Web with Lycos IBM Business Transformation IBM Internal Help IBM Standard Software Installer
Redbrook
Est 2004
A Great Place to Live
redbrook.com

Please enter your logon information:

ODServer:
Username:
Password:
Login

Figure 13-4 Sample Web application using ODWEK - Login screen


RBFoldersOD.jsp - Microsoft Internet Explorer

Address http://localhost:9080/RbOD/RBMgrServletOD?server=namibia&user=rbcounty&pwd=&cmd=login

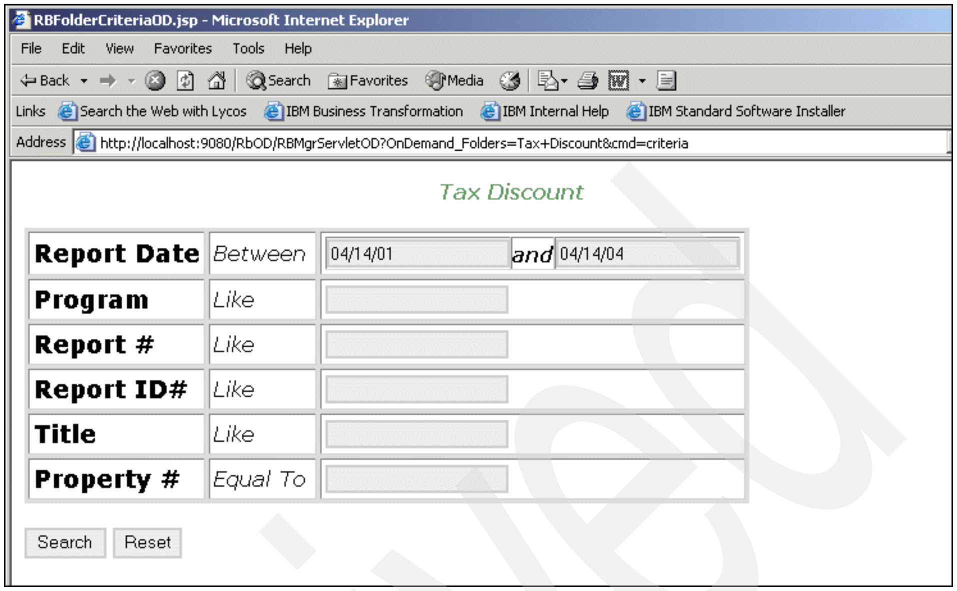
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Search the Web with Lycos IBM Business Transformation IBM Internal Help IBM Standard Soft

Select a folder:

Building Permits
Check Images
Letters and Notifications
Own History
Owners

open

Figure 13-5 Sample Web application using ODWEK - Folder list

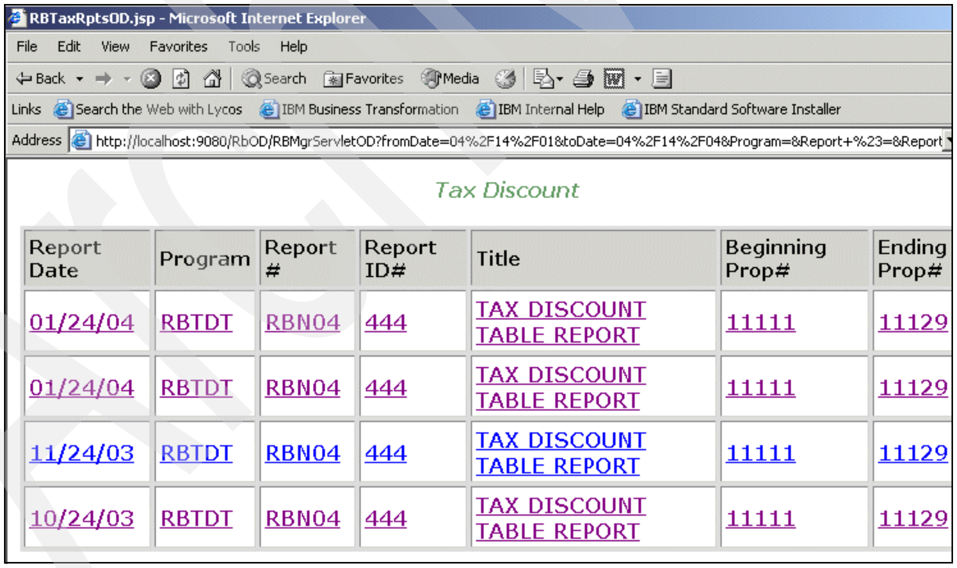

RBFolderCriteriaOD.jsp - Microsoft Internet Explorer

Address: http://localhost:9080/RbOD/RBMgrServletOD?OnDemand_Folders=Tax+Discount&cmd=criteria

Tax Discount

Report Date	*Between*	04/14/01 *and* 04/14/04
Program	*Like*	
Report #	*Like*	
Report ID#	*Like*	
Title	*Like*	
Property #	*Equal To*	

Figure 13-6 Sample Web application using ODWEK - Tax Discount folder search

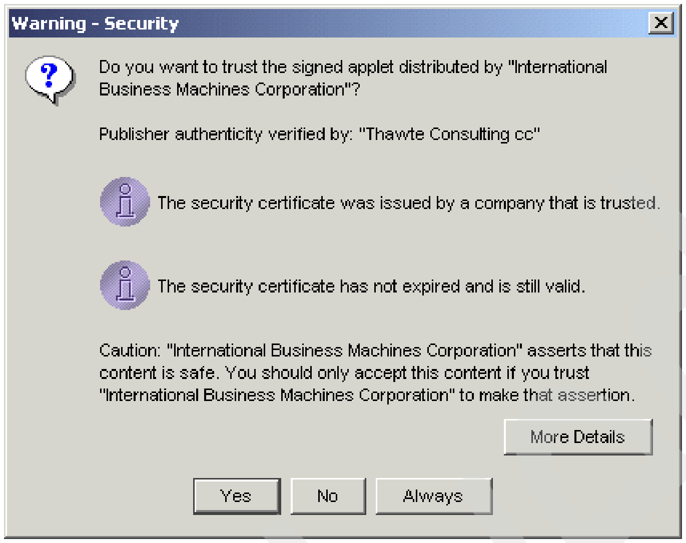

RBTaxRptsOD.jsp - Microsoft Internet Explorer

Address: http://localhost:9080/RbOD/RBMgrServletOD?fromDate=04%2F14%2F01&toDate=04%2F14%2F04&Program=&Report+%23=&Report

Tax Discount

Report Date	Program	Report #	Report ID#	Title	Beginning Prop#	Ending Prop#
01/24/04	RBTDT	RBN04	444	TAX DISCOUNT TABLE REPORT	11111	11129
01/24/04	RBTDT	RBN04	444	TAX DISCOUNT TABLE REPORT	11111	11129
11/24/03	RBTDT	RBN04	444	TAX DISCOUNT TABLE REPORT	11111	11129
10/24/03	RBTDT	RBN04	444	TAX DISCOUNT TABLE REPORT	11111	11129

Figure 13-7 Sample Web app using ODWEK - List of reports in Tax Discount folder



Do you want to trust the signed applet distributed by "International Business Machines Corporation"?

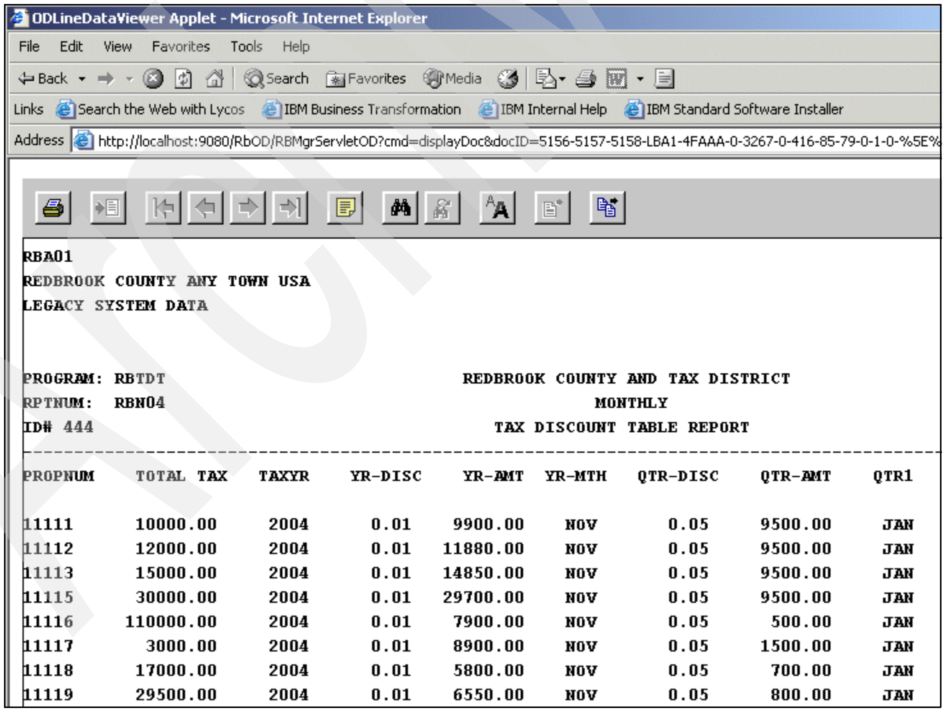
Publisher authenticity verified by: "Thawte Consulting cc"

The security certificate was issued by a company that is trusted.

The security certificate has not expired and is still valid.

Caution: "International Business Machines Corporation" asserts that this content is safe. You should only accept this content if you trust "International Business Machines Corporation" to make that assertion.

Figure 13-8 Sample Web app using ODWEK - Security warning


ODLineDataViewer Applet - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Search the Web with Lycos IBM Business Transformation IBM Internal Help IBM Standard Software Installer
Address

http://localhost:9080/RbOD/RBMgrServletOD?cmd=displayDoc&docID=5156-5157-5158-LBA1-4FAAA-0-3267-0-416-85-79-0-1-0-%5E%

```
RBA01
REDBROOK COUNTY ANY TOWN USA
LEGACY SYSTEM DATA


PROGRAM: RBTDT                          REDBROOK COUNTY AND TAX DISTRICT
RPTNUM:  RBN04                                      MONTHLY
ID# 444                                    TAX DISCOUNT TABLE REPORT
-------------------------------------------------------------------------------------
PROPNUM    TOTAL TAX   TAXYR   YR-DISC   YR-AMT   YR-MTH   QTR-DISC    QTR-AMT   QTR1

11111      10000.00    2004     0.01    9900.00    NOV       0.05     9500.00    JAN
11112      12000.00    2004     0.01   11880.00    NOV       0.05     9500.00    JAN
11113      15000.00    2004     0.01   14850.00    NOV       0.05     9500.00    JAN
11115      30000.00    2004     0.01   29700.00    NOV       0.05     9500.00    JAN
11116     110000.00    2004     0.01    7900.00    NOV       0.05      500.00    JAN
11117       3000.00    2004     0.01    8900.00    NOV       0.05     1500.00    JAN
11118      17000.00    2004     0.01    5800.00    NOV       0.05      700.00    JAN
11119      29500.00    2004     0.01    6550.00    NOV       0.05      800.00    JAN
```

Figure 13-9 Sample Web app using ODWEK - Line data applet view

13.6 Conclusion

In this chapter, we provided ODWEK API overviews, with classes, and some of the methods these classes have, and explained how we used them to build our sample application.

The sample application provided a generic OnDemand Web application. We demonstrated how you can log in to an OnDemand server, get a list of folders, search for reports using existing folder criteria, retrieve reports, and view specific line data report using the provided ODWEK applet viewer.

Building from this working sample application, we hope you can develop more specific applications that fulfill your business requirements.

For a complete Java bean reference, refer to the on-line OnDemand Web Enablement Kit Information Center.

Part 4

Appendixes

In this part of the book, we provide the following reference material:

A

Setting up case study infrastructure

This appendix describes the steps needed to set up the EAR Sample application that contains the whole modules you have been developing in the preceding chapters. In order to have this sample code working, you need to complete the following steps:

- Set up the RedBrook county legacy database system.
- Set up IBM Content Manager.
- Set up IBM Content Manager OnDemand.
- Set up WebSphere Studio Application Developer.
- Import the EAR file and configure your library definition environment.
- Configure the application parameters.

A.1 Set up RedBrook County legacy database system

Use the additional material provided with this IBM Redbook to find the db_setup directory. The script provided creates the database and loads the schema and data.

Open a DB2 Command Window and change your directory to wherever the unzipped setup files are located. The following files are required:

- import.bat
- import.txt
- redbrook.ddl
- redbrook_data <directory>

Make sure the user has the necessary privileges to drop and create a database.

From the command line, run the batch file **import.bat** and verify the import.log files for errors.

This script creates the REDBDB database.

For the complete Redbrook County legacy system database schema, refer to A.7, “Redbrook County legacy system database schema” on page 525.

A.2 Set up Content Manager system

The setup is done in two major steps:

- Use the configuration scripts for attributes and item type definitions.
- Use Content Manager System Administrator to complete the system sample environment.

A.2.1 Configuration scripts

The setup is divided into the following steps:

- **Attribute script setup:** This script creates the attribute definitions for the RedBrook data model. Refer to 6.3.2, “Working with attributes” on page 120 for further details.
- **Item type script setup:** This script creates the item type definitions for the RedBrook data model. Refer to 6.3.3, “Working with item types” on page 131 for further details.

- **Default Resource Manager Setup:** After creation of attributes and items, make sure the item type default Resource Manager definition is done. Follow the instructions in “Setting the default Resource Manager for item types” on page 138.

> **Important:** Every script includes a test section; make sure your definitions have been set up correctly.

For the Content Manager system definition that is created from the above scripts, refer to A.8, “Content Manager system definition” on page 529. Note, instead of running the scripts to create the definition, you can set up the Content Manager system definition based on the provided information in that section.

A.2.2 Content Manager System Administration UI

Before using the CM System Administration UI, create the following OS users in your server:

- **rbclerk:** RedBrook county clerk
- **rbsupervisor:** RedBrook county supervisor

These users do not need any special privileges, but have been defined at the operating system layer.

Next use the CM System Administration UI interface to set up:

- Privilege Sets
- Users and user groups
- Access control lists
- Work nodes, approval process, and worklists
- Auto-linking properties within your data model

Start the CM System Administrator Client:

Start → **Programs** → **IBM Content Manager for Multiplatforms** → **System Administration**

Privilege Sets definitions

Follow these steps:

1. Open the **Library Server folder** → **Authorization** → **Privilege Sets**
2. The default content of the Privilege Sets has a set called *ClientUserAllPrivs,* right-click on it and select **Copy** (Figure A-1).

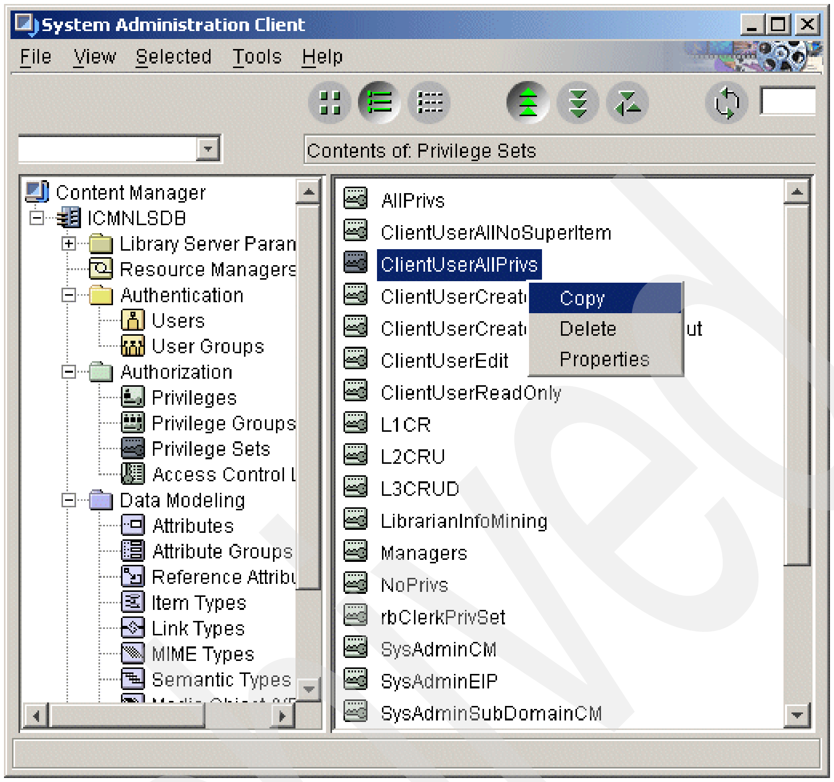
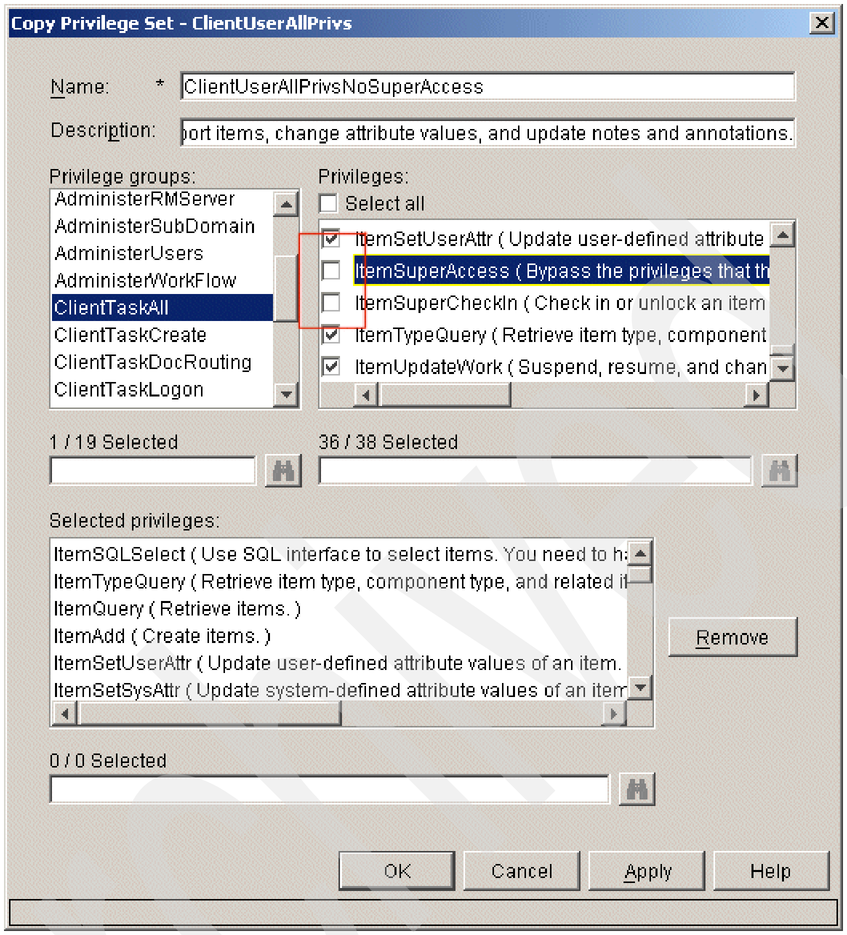


Figure A-1 Create Privilege set by copy

3. Set name to: *ClientUserAllPrivsNoSuperAccess*.
4. Select within Privilege groups section: *ClientTaskAll.*
5. Within Privileges section, deselect *ItemSuperAccess* and *ItemSuperCheckIn Privileges.*
6. Click **OK** (Figure A-2).



Name: * ClientUserAllPrivsNoSuperAccess

Description: port items, change attribute values, and update notes and annotations.

Privilege groups:
AdministerRMServer
AdministerSubDomain
AdministerUsers
AdministerWorkFlow
ClientTaskAll
ClientTaskCreate
ClientTaskDocRouting
ClientTaskLogon

Privileges:
- [✓] ItemSetUserAttr (Update user-defined attribute
- [ ] ItemSuperAccess (Bypass the privileges that th
- [ ] ItemSuperCheckIn (Check in or unlock an item
- [✓] ItemTypeQuery (Retrieve item type, component
- [✓] ItemUpdateWork (Suspend, resume, and chan

1 / 19 Selected

36 / 38 Selected

Selected privileges:
ItemSQLSelect (Use SQL interface to select items. You need to h
ItemTypeQuery (Retrieve item type, component type, and related it
ItemQuery (Retrieve items.)
ItemAdd (Create items.)
ItemSetUserAttr (Update user-defined attribute values of an item.
ItemSetSysAttr (Update system-defined attribute values of an item

0 / 0 Selected

Figure A-2 Privilege Set definition

Users and user groups definitions

Open the Authentication folder and create users with the following specifications:

User name: rbclerk	User description: RedBrook County system clerk Select *Use system password* Privilege set: *ClientUserAllPrivsNoSuperAccess* Grant privilege set: *ClientUserAllPrivsNoSuperAccess*

User name: rbsupervisor	User description: RedBrook County system supervisor Select *Use system password* Privilege set: *ClientUserAllPrivs* Grant privilege set: *ClientUserAllPrivs*

Create the following user groups and add the users as described below:

Name: RBCLERKS	Description: RedBrook County Clerks group Selected users: RBCLERK RBSUPERVISOR
Name: RBSUPERVISORS	Description: RedBrook County Supervisors group Selected users: RBSUPERVISOR

Access control list definitions

Open **Authentication** → **Access Control Lists**

Create the following ACLs as described:

Name: RBClerksWL	Description: RedBrook county clerks ACL Select *Group*: RBCLERKS Select Privilege Sets: ClientUserAllPrivsNoSuperAccess Click Add Click OK
Name: RBSupervisorWL	Description: RedBrook county supervisor ACL Select *Group*: RBSUPERVISORS Select Privilege Sets: ClientUserAllPrivs Click Add Click OK

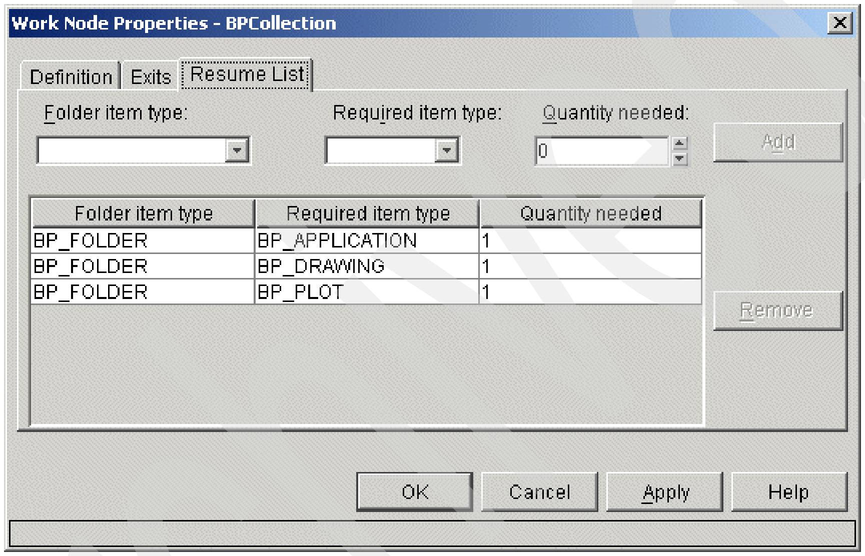
Work nodes definition

Open Document Routing folder and create 1 Collection Point and 3 Workbaskets using the following description:

Attention: Use the exact same name definition in order to avoid problems with the sample code.

1. Definition: **Collection point** (see Figure A-3)
   Name: *BPCollection*
   Description: Waiting for Doc submission
   Access control list: *RBClerksWL*

Resume List	Folder Item Type: BP_FOLDER Required item type BP_APPLICATION -> 1 BP_DRAWING -> 1 BP_PLOT -> 1

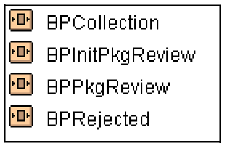


Folder item type	Required item type	Quantity needed
BP_FOLDER	BP_APPLICATION	1
BP_FOLDER	BP_DRAWING	1
BP_FOLDER	BP_PLOT	1

Figure A-3 BPCollection collection point definition

2. Definition: Workbasket
   Name: *BPInitPkgReview*
   Description: Initial package review
   Access control list: *RBClerksWL*
3. Definition: Workbasket
   Name: *BPPkgReview*
   Description: County Supervisor Review
   Access control list: *RBSupervisorWL*
4. Definition: Workbasket
   Name: *BPRejected*
   Description: Rejected applications packages
   Access control list: *RBClerksWL*

Your final work node definition list should look like this (Figure A-4):



BPCollection
BPInitPkgReview
BPPkgReview
BPRejected

Figure A-4 Work nodes definition

Process definition

Figure A-5 shows the process to be defined.

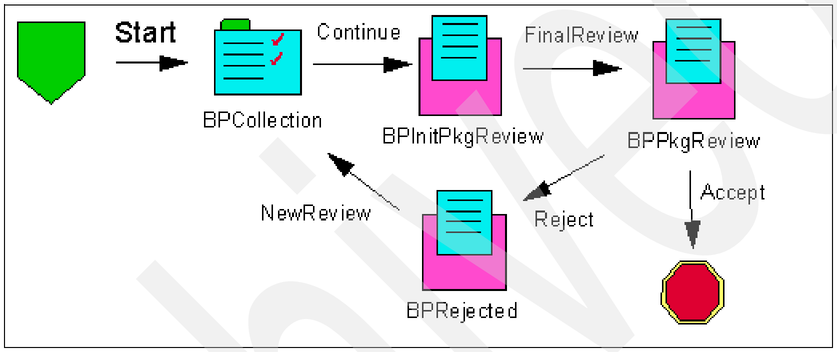


Figure A-5 RBBPApprove process definition

Create a new process using the values listed in Table A-1.

Name: *RBBPApprove*
Description: RedBrook building permit process
Access control list: *PublicReadACL*

Table A-1 RBBPApprove process implementation

From node	Selection	To node
START	Continue	BPCollection
BPCollection	Continue	BPInitPkgReview
BPInitPkgReview	FinalReview	BPPkgReview
BPPkgReview	Reject	BPRejected
BPRejected	NewReview	BPCollection
BPPkgReview	Accept	END

Worklists definition

Open the Worklists folders content and create four worklists according to the following specifications:

1. Name: *BPCollection*
   Description: Collection Point - Waiting for document submission
   Access control list: *RBClerksWL*

   Select the Nodes section and add *BPCollection* collection point

   Click OK
2. Name: *BPInitPkgReview*
   Description: Building permit initial package review
   Access control list: *RBClerksWL*
   Nodes: *BPInitPkgReview*
3. Name: *BPPkgReview*
   Description: Supervisor final application package review
   Access control list: *RBSupervisorWL*
   Nodes: *BPPkgReview*
4. Name: *BPRejected*
   Description: Rejected Application Folders
   Access control list: *RBClerksWL*
   Nodes: *BPRejected*

Review your configuration and make sure you follow the name conventions.

Set up auto-linking properties

As final step, modify your item-types definition in order to activate the auto-linking capability. Open **Data Modeling** → **Item Types**.

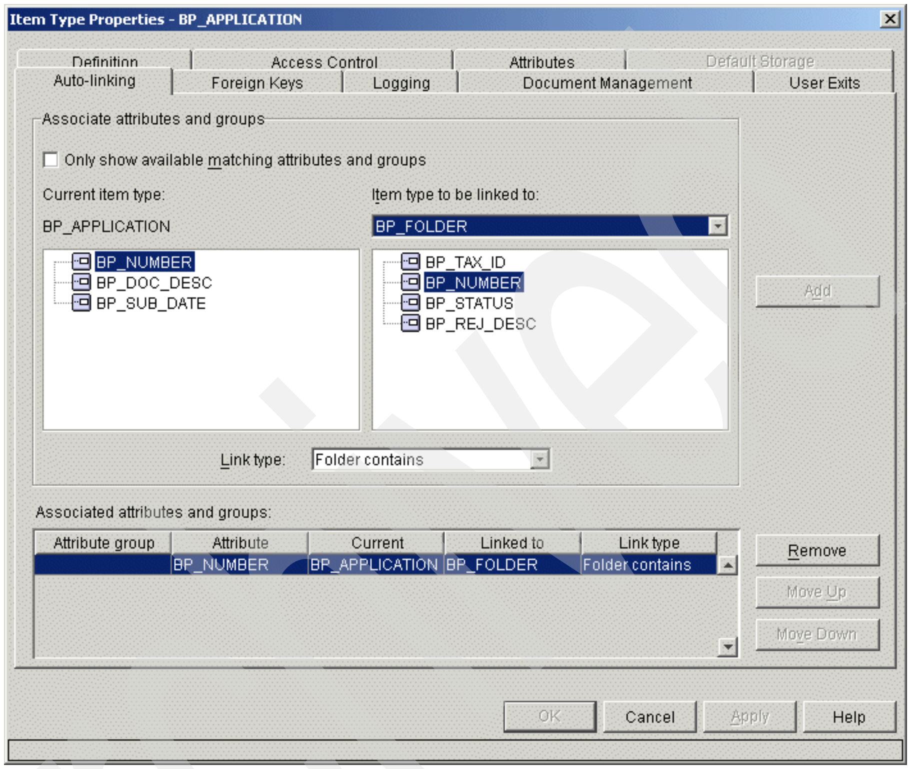
1. Right-click over **BP_APPLICATION** item type and select **Properties**.
2. Select the **Auto-linking** section.
3. From the BP_APPLICATION area, select BP_NUMBER attribute.
4. Select the item type to be linked to: BP_FOLDER.

   Note: If the BP_FOLDER item type was created successfully but is not displayed on the list, review every document part for both items and verify that the default Resource Manager and Collection has been set correctly.

5. The BP_NUMBER attribute should be selected automatically; if not, select it.
6. Select **Folder contains** as the Link type.
7. Click **Add**.

8. Click **OK**.

Repeat the procedure for **BP_DRAWING** and **BP_PLOT** items types (Figure A-6).



Item Type Properties - BP_APPLICATION

Associate attributes and groups

☐ Only show available matching attributes and groups

Current item type: BP_APPLICATION

- BP_NUMBER
- BP_DOC_DESC
- BP_SUB_DATE

Item type to be linked to: BP_FOLDER

- BP_TAX_ID
- BP_NUMBER
- BP_STATUS
- BP_REJ_DESC

Link type: Folder contains

Associated attributes and groups:

Attribute group	Attribute	Current	Linked to	Link type
	BP_NUMBER	BP_APPLICATION	BP_FOLDER	Folder contains

Figure A-6 Item auto-linking setup

Congratulations! Your Content Manager setup is ready to be used by the RedBrook County sample system.

Tip: You can test your data model and process definition using the IBM Content Manager Windows Client. After finishing your tests, delete the whole content in order to keep a clean setup in order to deploy the sample application.

A.3 Set up OnDemand system

The RedBrook County system has a section illustrating the use of II for Content's APIs to connect, query, retrieve, and view data from an OnDemand repository. The OnDemand sample data setup is not required to run the RedBrook county sample system; without this setup, only the application sections related to OnDemand will be unavailable.

For the OnDemand system definition that will be created in this section, refer to A.9, "OnDemand system definition" on page 532.

We have "exported" the Redbrook County OnDemand Folders, Applications, Application Groups and Users (RBuser01 and RBcounty) for you to use. Also included is all the dummy data we have used to load into OnDemand. These steps will help you set up your OnDemand system to include the Redbrook County OnDemand applications.

Where possible, we have given you a bat file to help load the data quickly. These instructions are for Windows systems.

1. Download the ODRedbrook.zip from the additional material provided for this redbook to a temporary directory on a Windows machine that has the OnDemand Administrator Client.
2. Go to that directory and **unzip** the **ODRedbrook.zip** to the temporary directory.
3. Find the **Redbrook1.bat** and execute it. This bat file will:
   a. Create a new directory called RBexport.
   b. Copy the exported files to RBexport directory.
   c. Create a directory called RBfiles.
   d. Copy the dummy data to RBfiles.

Important: Do not execute Redbrook2.bat at this time.

4. Log on to *your* OnDemand Administration Client with an ID that has administrative permissions.
5. Create a new OnDemand Server on *your* OnDemand system. See Figure A-7.

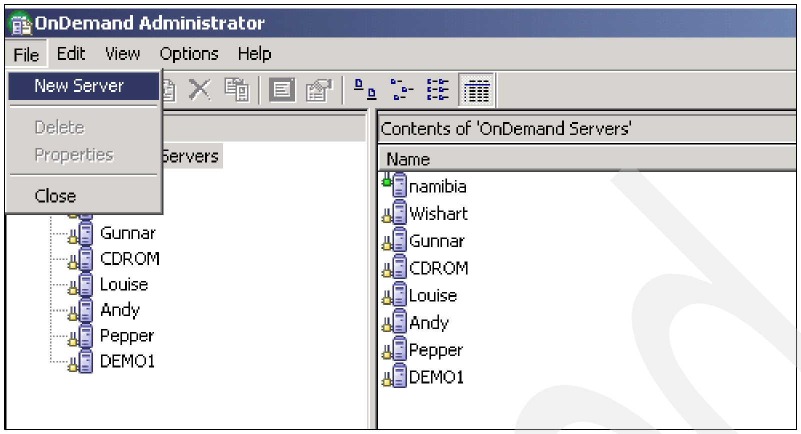
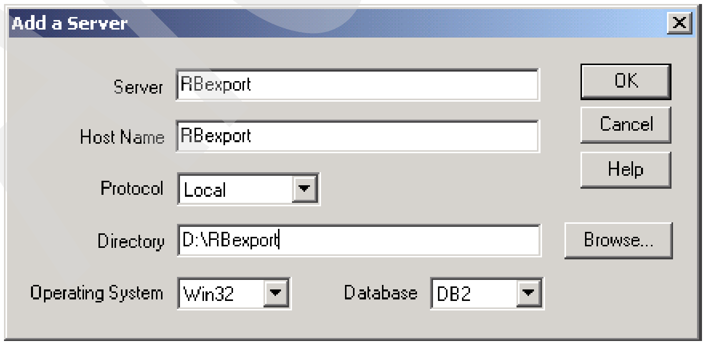


Figure A-7 New OD server

6. Create a new *local* OnDemand Server called RBexport. This is where we have placed the OD Redbrook OnDemand applications, so they can be exported to your OnDemand system. Enter the fields as shown in Figure A-8.
   - Server = `RBexport`
   - Hostname = `RBexport`
   - Protocol = `Local`
   - Directory = x:\RBexport (`x` = the directory you typed in the ODRedbrook1.bat during execution).
   - Operation System = `Win32`
   - Database= `DB2` or `MS SQL`
   - Click **OK**.



Server: RBexport
Host Name: RBexport
Protocol: Local
Directory: D:\RBexport
Operating System: Win32
Database: DB2

Figure A-8 Adding local server RBexport

7. *Right-click* on the newly created RBexport server and click **Logon**. Logon to the new local OD server as `admin` and the password is `ondemand`.
8. When you logon, you will see:
   – The OD Users:
     - ADMIN - password ondemand
     - RBCOUNTY no password
     - RBUSER01 no password
   – Ten (10) OD Applications:
     - RBAssessment
     - RBBuilding Permits
     - RBCheque Images
     - RBletters
     - RBOwn History
     - RBOwners
     - RBProperty Tax
     - RBSales
     - RBTax
     - RBTax Discount
   – Ten (10) OD Application Groups:
     - RBAssessment
     - RBBuilding Permits
     - RBCheque Images
     - RBletters
     - RBOwn History
     - RBOwners
     - RBProperty Tax
     - RBSales
     - RBTax
     - RBTax Discount
   – Ten (10) Folders
     - Assessment
     - Building Permits
     - Cheque Images
     - Letters and Notifications
     - Owner History
     - Owners
     - Property Tax
     - Sales
     - Tax
     - Tax Discount

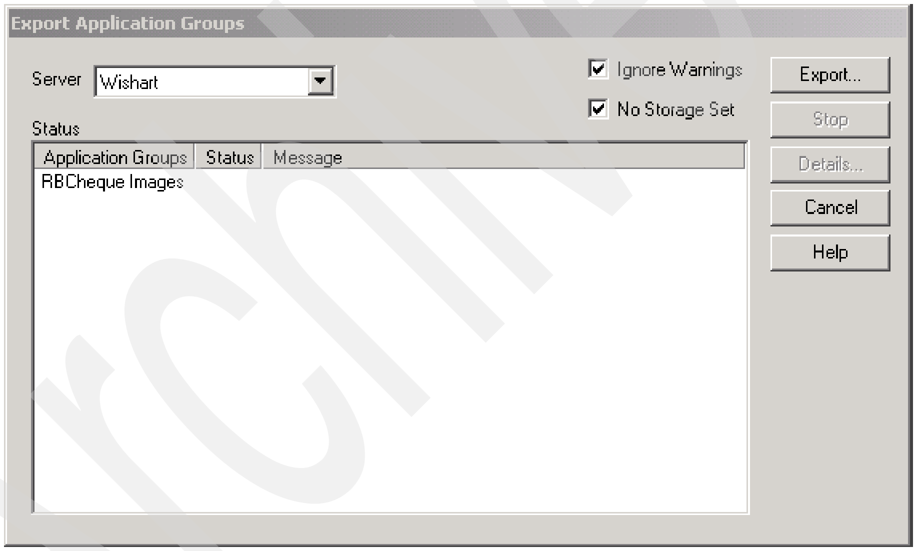
9. ***Before*** you run the second bat file (Redbrook2.bat), *export* all the Users, Application Groups, and Folders (*in that order*) to *your* OnDemand Server:

   a. Users
   b. Application Groups
   c. Folders

10. To export a User, Application Group, or Folder, you highlight the item you are going to export, right-click, export.

> **Tip:** You can highlight multiple items to be exported. This really speeds up the process!

11. When the Export menu appears, select *your* OnDemand server to export to, click **Ignore Warnings** and click **No Storage Set** (see Figure A-9), then click **Export**. Later (in Step12) you will add a storage set.



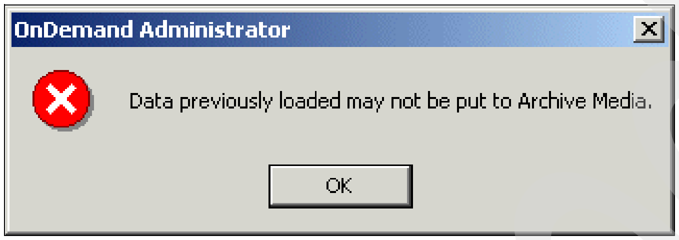
Server: Wishart
[✓] Ignore Warnings
[✓] No Storage Set
Status

Application Groups	Status	Message
RBCheque Images		

Figure A-9 Exporting

12. ***Before*** you run the Redbrook2.bat, you need to define Storage Sets in all of the Application Groups you exported. See Figure A-11.
    – Open the Application Groups icon on your OnDemand server where you exported to.
    – Right-click on the first of 10 Application Groups (RBAssessment).
    – Click **Update**.
    – Click the **Storage Management** tab.

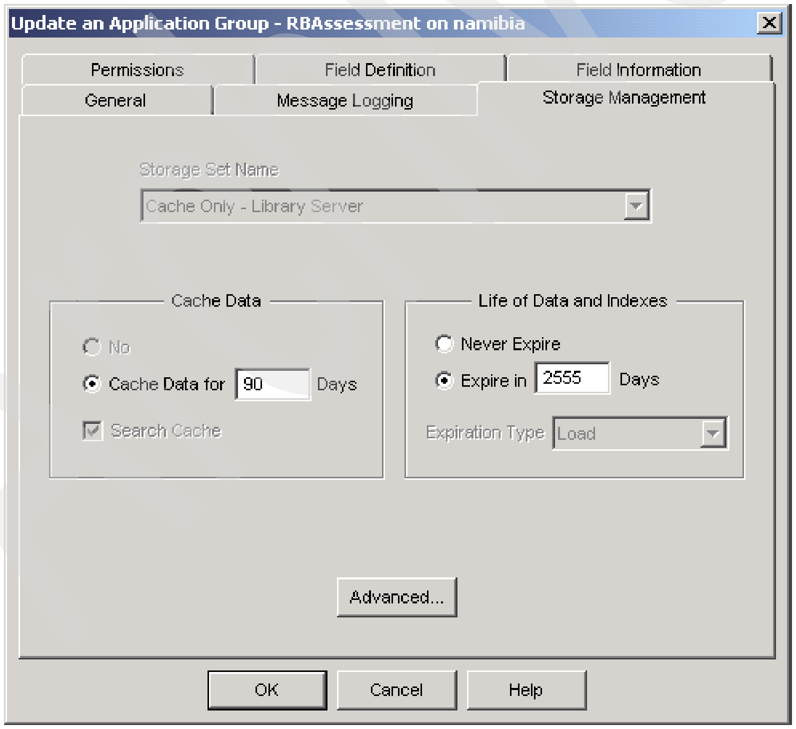
– On the Storage Management tab, click the **Storage Set Name** pull-down tab.
– Select a Storage Set (usually Cache Only - Library Server).
– Click **OK** and you are done with the first Application Group.
– You will get the message in Figure A-10. Click **OK**.


OnDemand Administrator

Data previously loaded may not be put to Archive Media.

Figure A-10 Click OK on this message

– Now repeat the steps above for the other 9 Application Groups. See Figure A-11.


Update an Application Group - RBAssessment on namibia

Storage Set Name: Cache Only - Library Server

Cache Data
- No
- [✓ Cache Data for] 90 Days
- [✓ Search Cache]

Life of Data and Indexes
- Never Expire
- [✓ Expire in] 2555 Days

Expiration Type: Load

Figure A-11 Choosing your Storage Set

13. ***When*** you have *exported all* the items, *choose the Storage Sets,* then you are ready for the loading of the data, which is done by the second bat file. Redbrook2.bat.
14. The Redbrook2.bat will ask you:
    – Where you have installed OnDemand. This is needed to make a path to the executable files in OnDemand's \bin directory.
    – The name of your OnDemand Server (name or IP address).
    – The user ID for OnDemand.
    – The password, so the bat file can load data to your OnDemand system where you have exported the Redbrook application to.
15. Go to the *RBfiles* directory created by the 1st Redbrook1.bat file.
16. Find the second bat file called Redbrook2.bat and execute it.
17. This program will load the data into *your* OnDemand server.
18. You should have the Redbrook OnDemand system set up on your machine!

A.4 Set up WebSphere Studio Application Developer

In order to have a better project organization, we recommend that you set up a specific workplace to deploy the RedBrook County System within your development tool. This can be done by creating a dedicate file system directory like this:

```
c:\wsad_sg246338
```

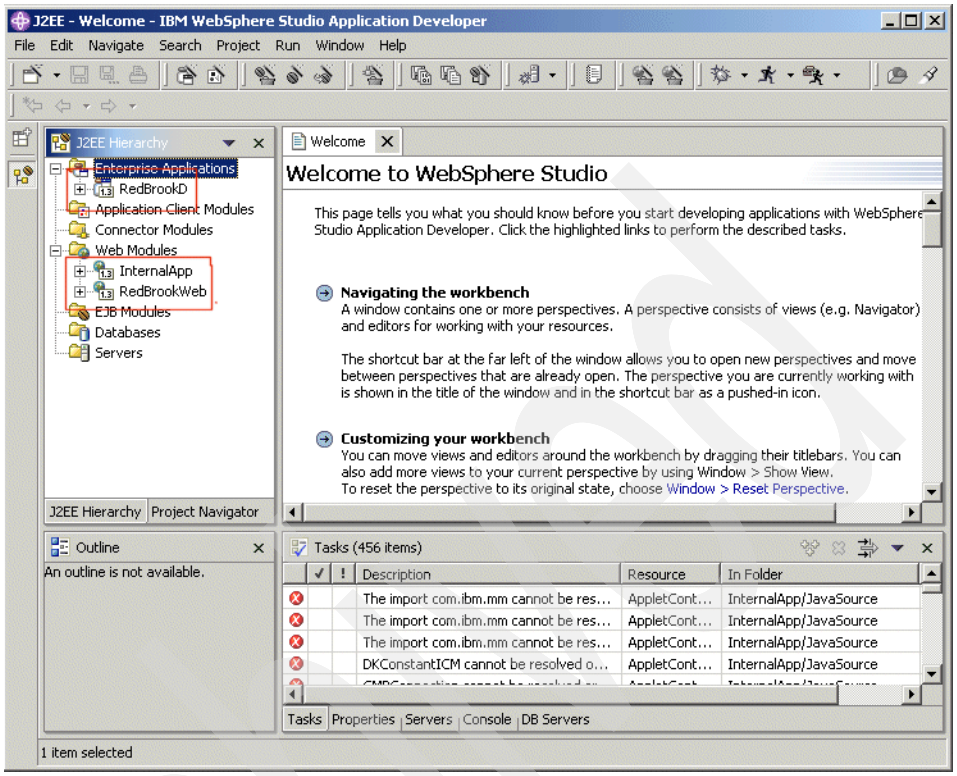
Set up your WebSphere Studio Application Developer launch shortcut to start with the following command:

```
"ROUTE_TO_WSAD_HOME\wsappdev.exe" -data c:\wsad_sg246338
```

Start WebSphere Studio application Server. If you made the configuration above, your workplace should be empty.

1. Select **File** → **Import**. This action launches the import wizard menu; select the **EAR file** wizard.
2. Select **RedBrookD.ear** from the RedBook additional materials directory.
3. Use the default values and click **Finish**.

WebSphere Studio Application Developer imports your file and displays many compilation problems, but don't worry about them right now. See Figure A-12.



Welcome to WebSphere Studio

This page tells you what you should know before you start developing applications with WebSphere Studio Application Developer. Click the highlighted links to perform the described tasks.

Navigating the workbench
A window contains one or more perspectives. A perspective consists of views (e.g. Navigator) and editors for working with your resources.

The shortcut bar at the far left of the window allows you to open new perspectives and move between perspectives that are already open. The perspective you are currently working with is shown in the title of the window and in the shortcut bar as a pushed-in icon.

Customizing your workbench
You can move views and editors around the workbench by dragging their titlebars. You can also add more views to your current perspective by using Window > Show View. To reset the perspective to its original state, choose Window > Reset Perspective.

An outline is not available.

✓	!	Description	Resource	In Folder
		The import com.ibm.mm cannot be res...	AppletCont...	InternalApp/JavaSource
		The import com.ibm.mm cannot be res...	AppletCont...	InternalApp/JavaSource
		The import com.ibm.mm cannot be res...	AppletCont...	InternalApp/JavaSource
		DKConstantICM cannot be resolved o...	AppletCont...	InternalApp/JavaSource

Figure A-12 Initial WebSphere Studio Application Developer setup

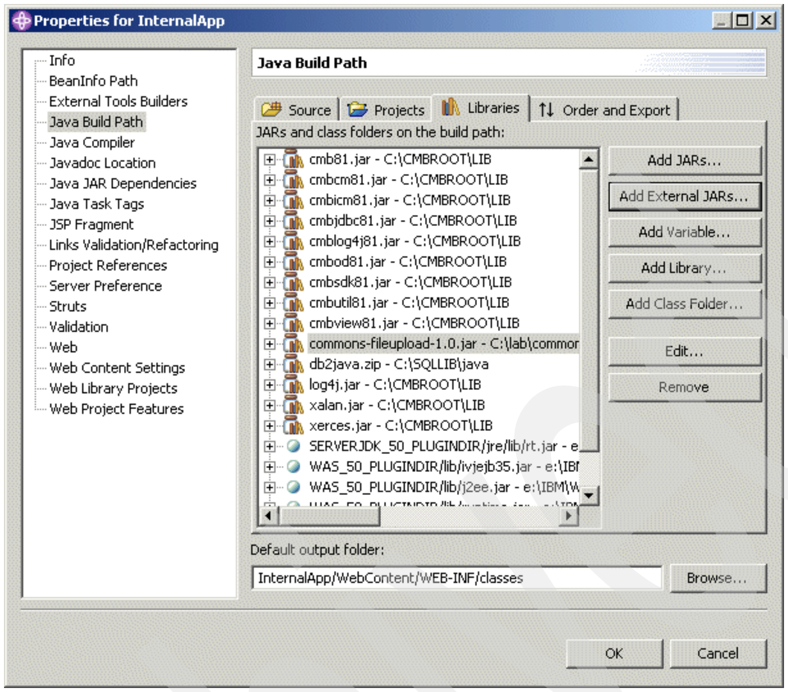
Your workspace contains one Enterprise Project: RedBrookD; and two Web Projects: InternalApp and RedBrook Web.

4. Configure the Java Build Path. Right-click over the **InternalApp** folder and select **Properties** from the context menu.
5. Select the option **Java Build Path** from the panel in the left side, and select **Libraries** at the right panel.
6. Using the Add External JARs options, select the following files (Depending on your CM Setup, your library directory might change.)

   From the c:\cmbroot\lib, select:
   cmb81.jar, cmbcm81.jar, cmbfed81.jar, cmbicm81.jar, cmblog4j81.jar, cmbsdk81.jar, cmbutil81.jar, cmbview81.jar, xalan, xercers, cmbod81.jar, cmbjdbc81.jar

   From your DB2 Home Directory (c:\SQLL`IB\java`), add `db2java.zip`

   From your RedBrook additional material directory, add the Apache library, `commons-fileupload-1.0.jar`. See Figure A-13.



JARs and class folders on the build path:

```
cmb81.jar - C:\CMBROOT\LIB
cmbcm81.jar - C:\CMBROOT\LIB
cmbicm81.jar - C:\CMBROOT\LIB
cmbjdbc81.jar - C:\CMBROOT\LIB
cmblog4j81.jar - C:\CMBROOT\LIB
cmbod81.jar - C:\CMBROOT\LIB
cmbsdk81.jar - C:\CMBROOT\LIB
cmbutil81.jar - C:\CMBROOT\LIB
cmbview81.jar - C:\CMBROOT\LIB
commons-fileupload-1.0.jar - C:\lab\commor
db2java.zip - C:\SQLLIB\java
log4j.jar - C:\CMBROOT\LIB
xalan.jar - C:\CMBROOT\LIB
xerces.jar - C:\CMBROOT\LIB
SERVERJDK_50_PLUGINDIR/jre/lib/rt.jar - e
WAS_50_PLUGINDIR/lib/ivjejb35.jar - e:\IBI
WAS_50_PLUGINDIR/lib/j2ee.jar - e:\IBM\W
```

Default output folder:

InternalApp/WebContent/WEB-INF/classes

Figure A-13 Java Build path setup

7. Click **OK** and repeat the procedure for the second Web Project: *RedBrookWeb*. The compilation problems are gone after this configuration step is done.

Create a Server Project in order to have a WebSphere Application Server runtime environment for our sample application:

1. Select **File** → **New** → **Other**, this action launches the Wizard menu. Select **Server** from the left panel and **Server and Server configuration** from the right panel. Click **Next**.
2. Server name: RedBrookServer
   Server Type: WebSphere version 5.0 -> Test Environment
3. Click **Next**.
4. Use the default port number or specify your preference (through the rest of this example we are referring to the default port 9080).
5. Click **Finish**.

6. Change to the Server perspective and set the JAR files for the runtime environment. **Select Window** → **Open Perspective** → **other**, and from the popup window, select the **Server** perspective. Click **OK**.
7. The **RedBrookServer** is listed at the below left panel; right-click and select **Open** from the context menu. This opens the configuration wizard.
8. Select the **Environment** tab and use the **Class Path** section to add the same external JARs files used within the Web Projects mentioned above. (CM libs, DB2 lib, and Apache lib).
9. You also need to add to the class path the Folder where the file *cmbicmenv.properties* is located. Use the **Add External Folder** option to set your folder (c:\Program Files\IBM\CMgmt).
10. Save the Server project and close the Server configuration wizard.
11. Right-click again over **RedBrookServer** and select **Add and remove projects** from the context menu.
12. Add **RedBrookD** Enterprise project.
13. Click **Finish**.

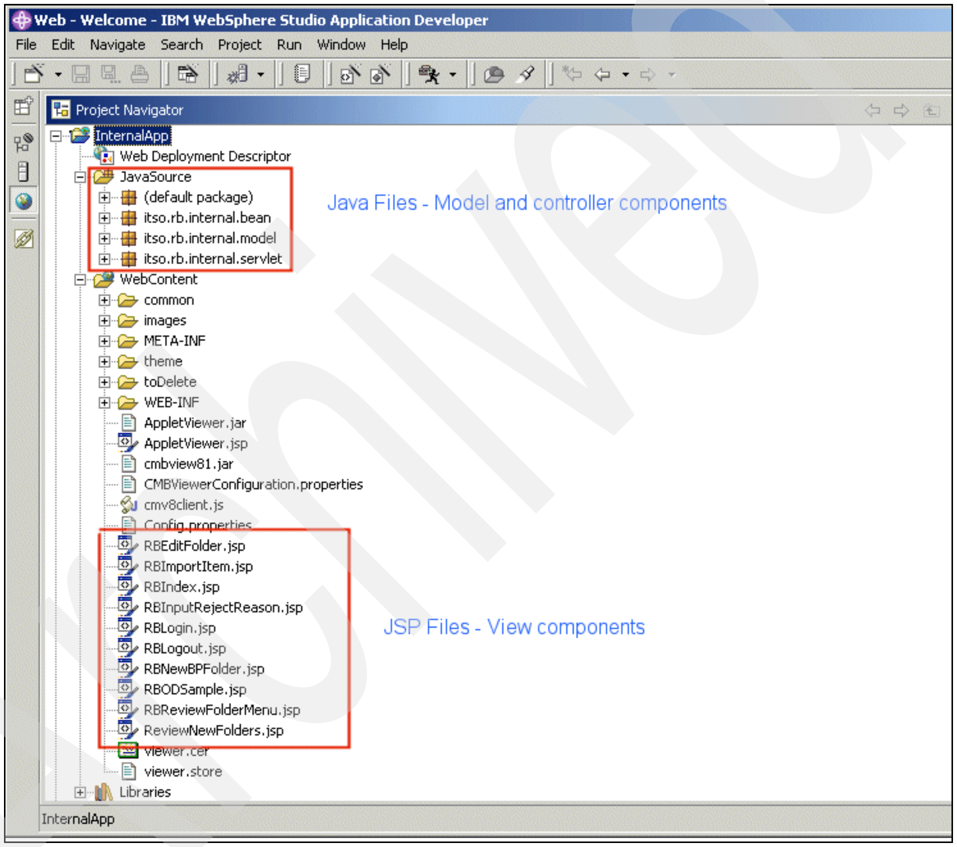
Open the Web perspective as explained in step 6 above for the Server perspective and navigate through the Web project files.

A.5 Getting familiar with project files

The RedBrook County sample application actually has two sample applications:

InternalApp	RedBrook County Intranet sample application
RedBrookWeb	RedBrook County Internet sample application

Both projects have similar file organizations. Open the JavaSource and Web Content folders to list the source code as shown in Figure A-14.


Web - Welcome - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Run Window Help
Project Navigator
Java Files - Model and controller components
JSP Files - View components

```
InternalApp
   Web Deployment Descriptor
   JavaSource
      (default package)
      itso.rb.internal.bean
      itso.rb.internal.model
      itso.rb.internal.servlet
   WebContent
      common
      images
      META-INF
      theme
      toDelete
      WEB-INF
      AppletViewer.jar
      AppletViewer.jsp
      cmbview81.jar
      CMBViewerConfiguration.properties
      cmv8client.js
      Config.properties
      RBEditFolder.jsp
      RBImportItem.jsp
      RBIndex.jsp
      RBInputRejectReason.jsp
      RBLogin.jsp
      RBLogout.jsp
      RBNewBPFolder.jsp
      RBODSample.jsp
      RBReviewFolderMenu.jsp
      ReviewNewFolders.jsp
      viewer.cer
      viewer.store
   Libraries
InternalApp
```

Figure A-14 RedBrook County Web Project files organization

These sample applications were built using the MVC pattern.

A.5.1 Set up Config.properties files

Both projects have, in the *Web Content* folder, a configuration file called *Config.properties*. Open the files and change the parameters properly to fit your server configuration environment.

Example A-1 shows a Config.properties content sample.

Example: A-1 Web sample projects Config.properties file

```
# RedBrook County Intranet System
# config properties file

# ICM Default connection parameters
ICMServer = lsdb
ICMUser   = icmadmin
ICMPasswd = yourpasswd

# DB2 Default connection parameters
DB2DB     = redbdb
DB2User   = Administrator
DB2Passwd = yourpasswd

# OnDemand default connection parameters
ODServer  = namibia
ODUser   = rbcounty
ODPasswd  =
.
.
```

A.5.2 Run the Web sample applications

Make sure you have customized the Config.properties file as mentioned above. Every Web project starts with the *RBIndex.jsp* file.

Right-click the **RBIndex.jsp** JSP file in the *InternalApp* Web project, and select **Run on server** from the context menu. The Server selection wizard is launched; select your **RedBrookServer** and click **Finish** (Figure A-15).

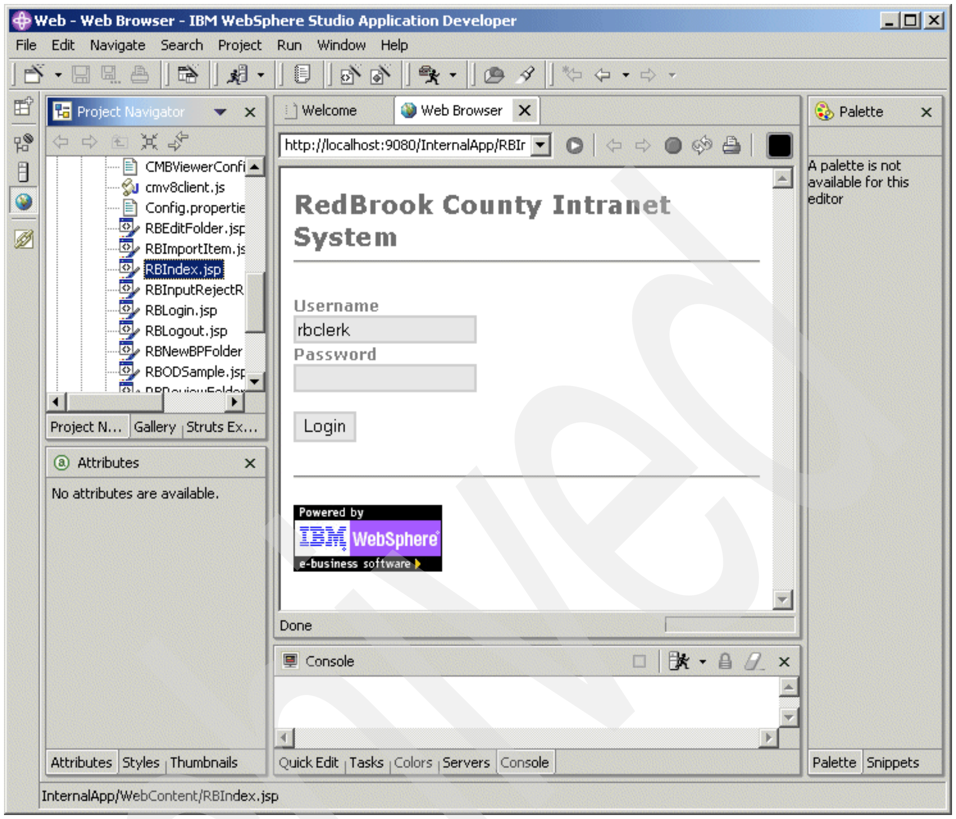

Web - Web Browser - IBM WebSphere Studio Application Developer
File Edit Navigate Search Project Run Window Help
Project Navigator
CMBViewerConfi
cmv8client.js
Config.propertie
RBEditFolder.jsp
RBImportItem.js
RBIndex.jsp
RBInputRejectR
RBLogin.jsp
RBLogout.jsp
RBNewBPFolder
RBODSample.jsp
Project N... Gallery Struts Ex...
Attributes
No attributes are available.
Attributes Styles Thumbnails
Welcome
Web Browser
http://localhost:9080/InternalApp/RBIr
RedBrook County Intranet System
Username
rbclerk
Password
Login
Powered by IBM WebSphere e-business software
Done
Console
Quick Edit Tasks Colors Servers Console
Palette
A palette is not available for this editor
Palette Snippets
InternalApp/WebContent/RBIndex.jsp

Figure A-15 Running RedBrook County intranet sample system

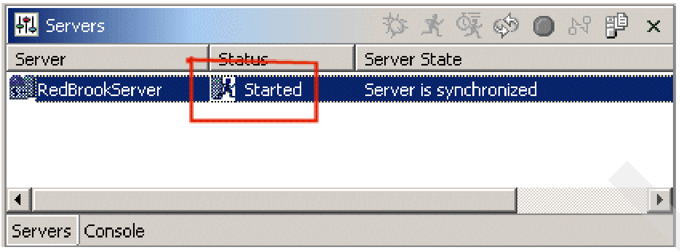
By default, the WebSphere studio application developer uses an internal Web browser. However, you can always change the WebSphere Studio Application Developer preferences to launch an external browser, or just copy and paste the URL to your favorite browser.

Note: This Web sample was only tested within the Microsoft Internet Explorer browser.

A.6 Navigate the Web sample application

The following paragraphs describe the typical navigation flow.

Make sure your WebSphere v5.0 Test Environment status is Started, if not, start the server (Figure A-16).



RedBrookServer | Started | Server is synchronized

Figure A-16 Test environment is started

A.6.1 Internal application navigation

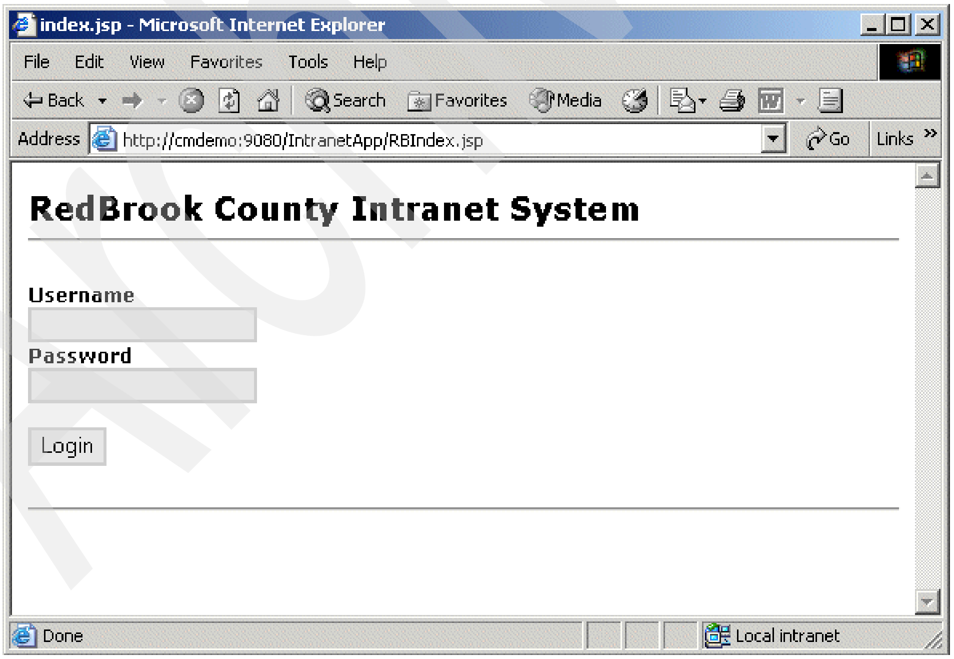
Open your Web browser and customize the following URL according to your configuration environment (Figure A-17).

If you coded the application following the step-by step-instructions:

```
http://localhost:9080/IntranetApp/RBIndex.jsp
```

Otherwise, if you imported the EAR file:

```
http://localhost:9080/InternalApp/RBIndex.jsp
```

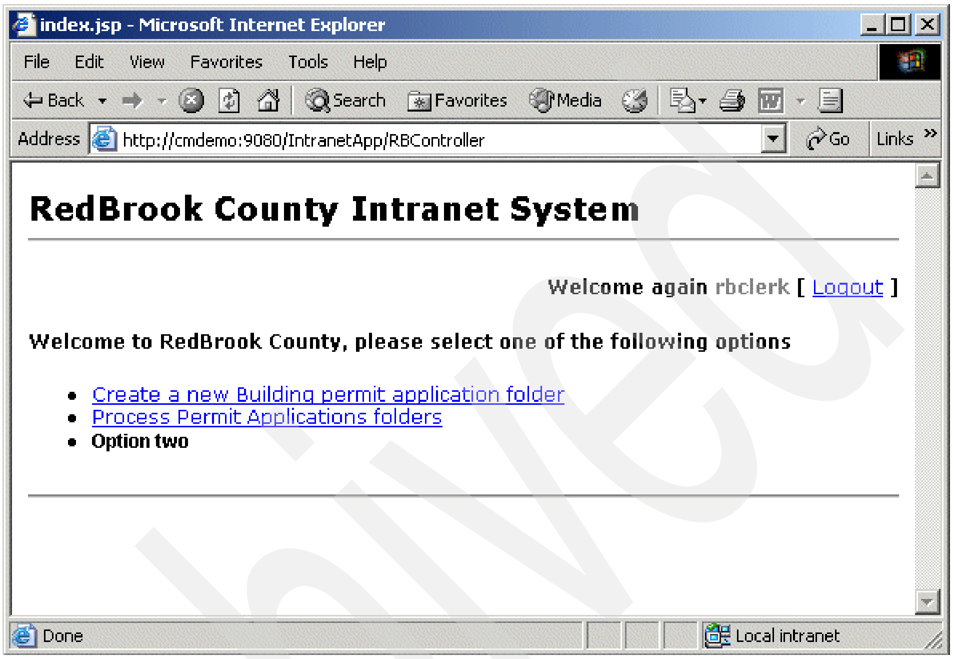


http://cmdemo:9080/IntranetApp/RBIndex.jsp

RedBrook County Intranet System

Figure A-17 Intranet index page

The main screen for the internal application allows you to login using a valid IBM Content Manager user ID. Use either the `rbclerk` or `rbsupervisor` user ID to login (Figure A-18).


index.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController Go Links
Done
Local intranet

RedBrook County Intranet System

Welcome again rbclerk [Logout]

Welcome to RedBrook County, please select one of the following options

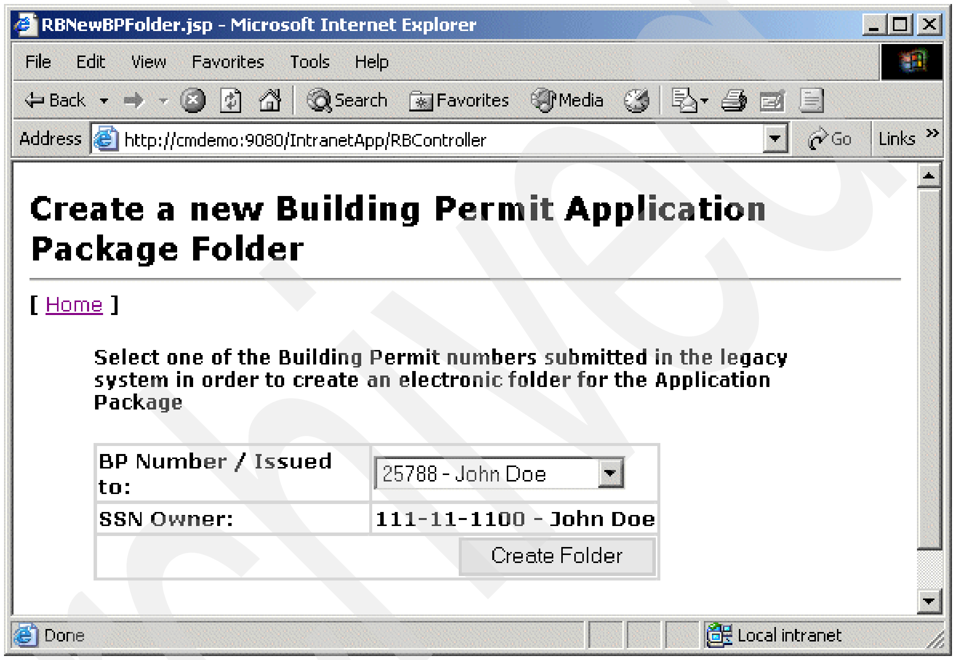
- Create a new Building permit application folder
- Process Permit Applications folders
- Option two

Figure A-18 Intranet main menu

The main menu allows you to:

- Define a new building permit application folder within IBM Content Manager
- Process the building permit application folder within the approval Document Routing process.

Select the first option in order to create a new building permit application folder (Figure A-19).


RBNewBPFolder.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController
Go Links
Done
Local intranet

Create a new Building Permit Application Package Folder

[Home]

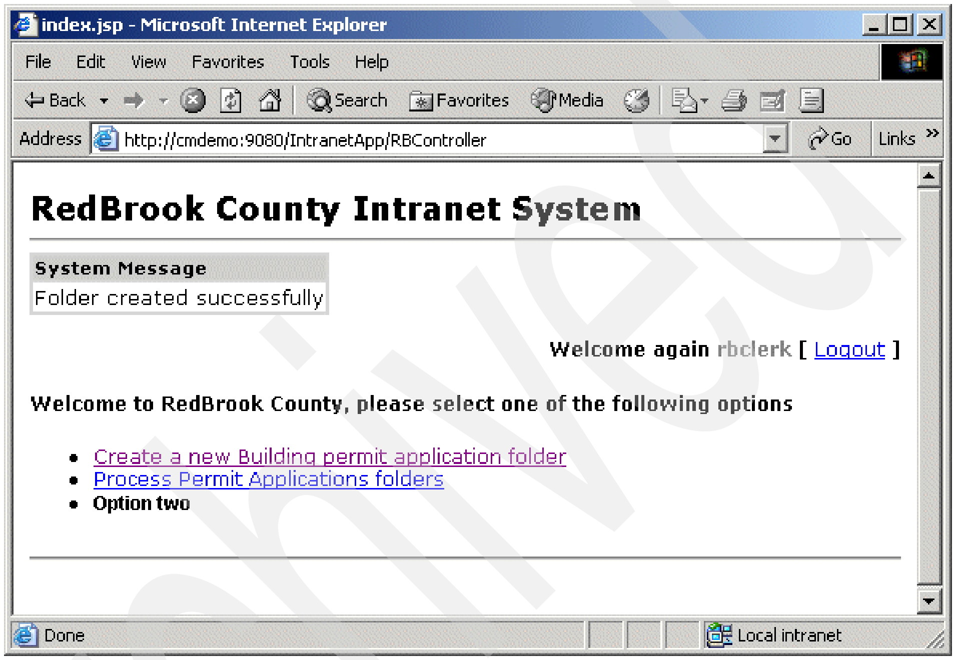
Select one of the Building Permit numbers submitted in the legacy system in order to create an electronic folder for the Application Package

BP Number / Issued to:	25788 - John Doe
SSN Owner:	111-11-1100 - John Doe
	Create Folder

Figure A-19 Building permit application folder definition

The Building permit numbers shown at the select box are taken from the RedBrook DB2 legacy system. They represent building permits that were created within the legacy system and have not been approved yet.

Select a building permit number in order to update the SSN and click *Create Folder* (Figure A-20).



index.jsp - Microsoft Internet Explorer

Address http://cmdemo:9080/IntranetApp/RBController

RedBrook County Intranet System

System Message
Folder created successfully

Welcome again rbclerk [Logout]

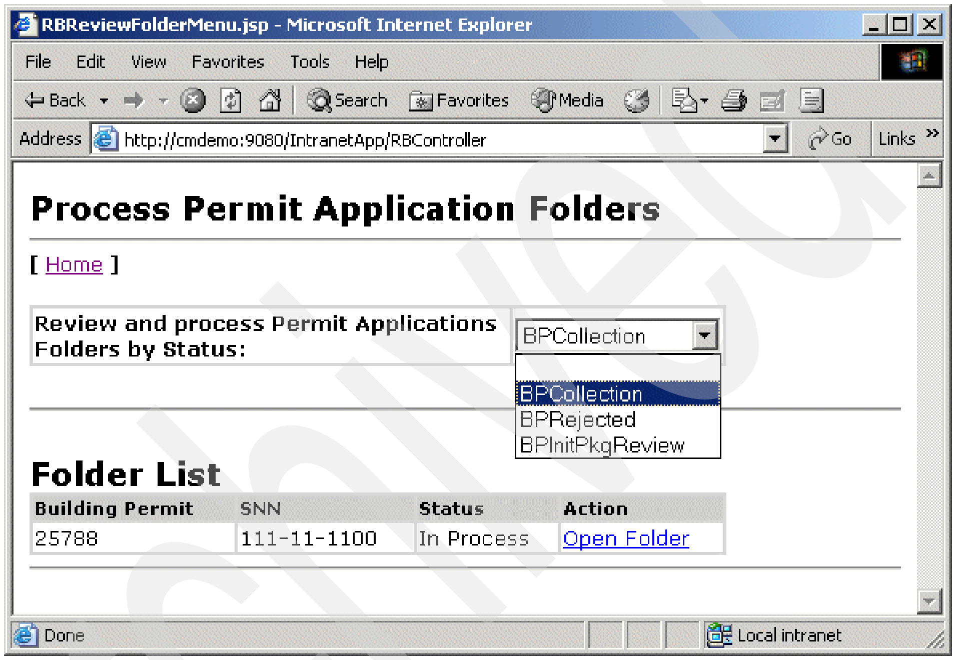
Welcome to RedBrook County, please select one of the following options

- Create a new Building permit application folder
- Process Permit Applications folders
- Option two

Figure A-20 Building permit application folder created successfully

A new BP_Folder item was added to the IBM Content Manager system using the metadata taken from the RedBrook DB2 legacy system and started at the Redbrook County approval process. Also, the legacy system was updated with the status ‘P’ (Pending) status. (*itso.building_permit* db2 table at the *redbdb* database).

Select the Process permit applications folders from the menu (Figure A-21).


RBReviewFolderMenu.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController
Go Links
Done
Local intranet

Process Permit Application Folders

[Home]

Review and process Permit Applications Folders by Status: BPCollection

BPCollection
BPRejected
BPInitPkgReview

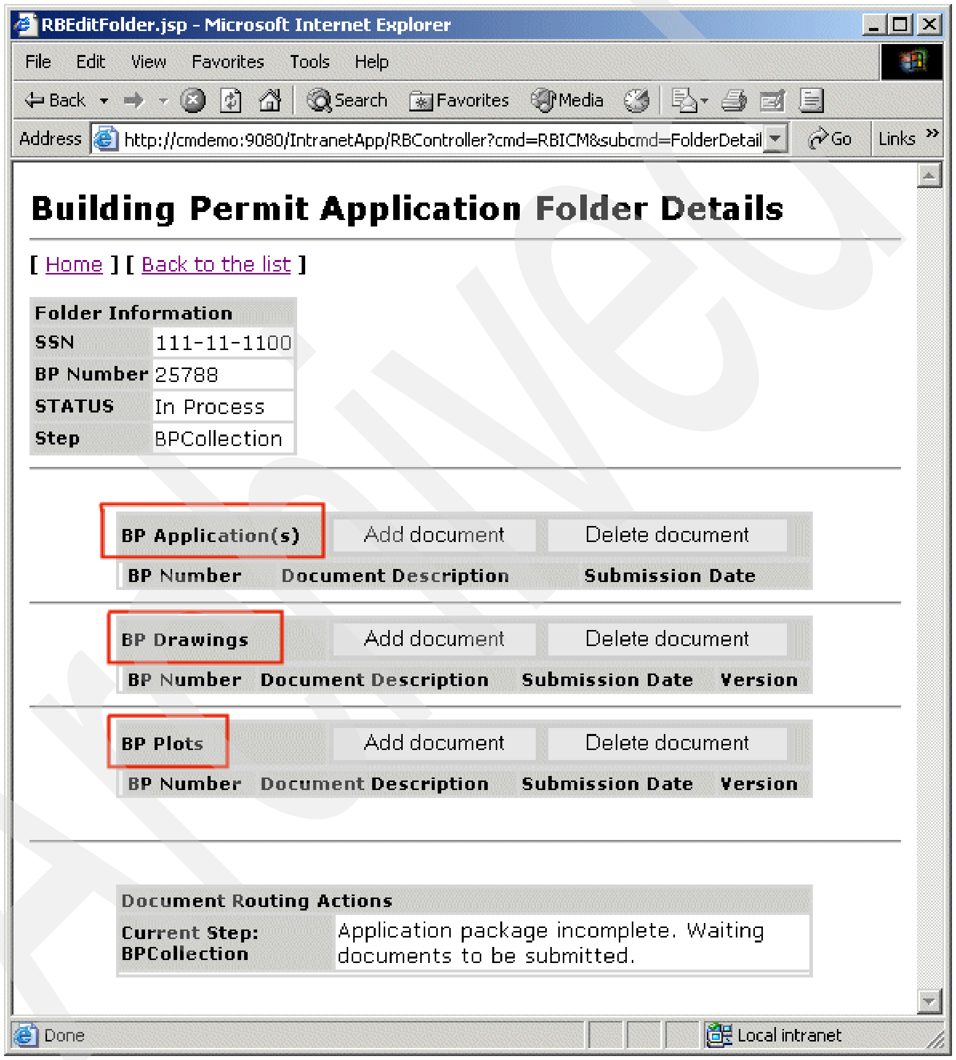
Folder List

Building Permit	SNN	Status	Action
25788	111-11-1100	In Process	Open Folder

Figure A-21 Open worklist

The approval process has four work nodes; BPCollection, BPInitPkgReview, BPPkgReview and BPRejected. For more information about the approval process definition, refer to “Process definition” on page 494.

The created folder is at the first node, BPCollection. Select Open Folder to display the folder detail (Figure A-22).


RBEditFolder.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address
Go Links
Done
Local intranet

http://cmdemo:9080/IntranetApp/RBController?cmd=RBICM&subcmd=FolderDetail

Building Permit Application Folder Details

[Home] [Back to the list]

Folder Information	
SSN	111-11-1100
BP Number	25788
STATUS	In Process
Step	BPCollection

BP Application(s) — Add document — Delete document

BP Number	Document Description	Submission Date

BP Drawings — Add document — Delete document

BP Number	Document Description	Submission Date	Version

BP Plots — Add document — Delete document

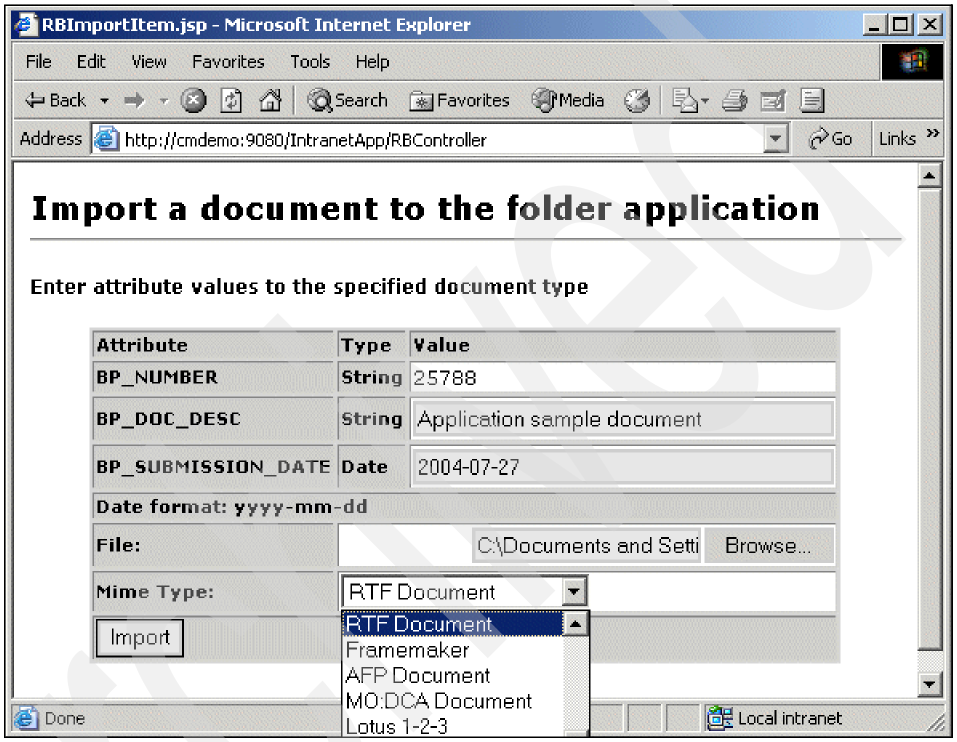
BP Number	Document Description	Submission Date	Version

Document Routing Actions	
Current Step: BPCollection	Application package incomplete. Waiting documents to be submitted.

Figure A-22 BP Folder detail

In order to advance to the next step in the process, the collection node must receive at least one item for the tree defined item types: Application, Drawing, and Plots. At this time, the folder is empty.

Click the **Add document** button from the BP Applications document list (Figure A-23).


RBImportItem.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController
Go Links
Done
Local intranet

Import a document to the folder application

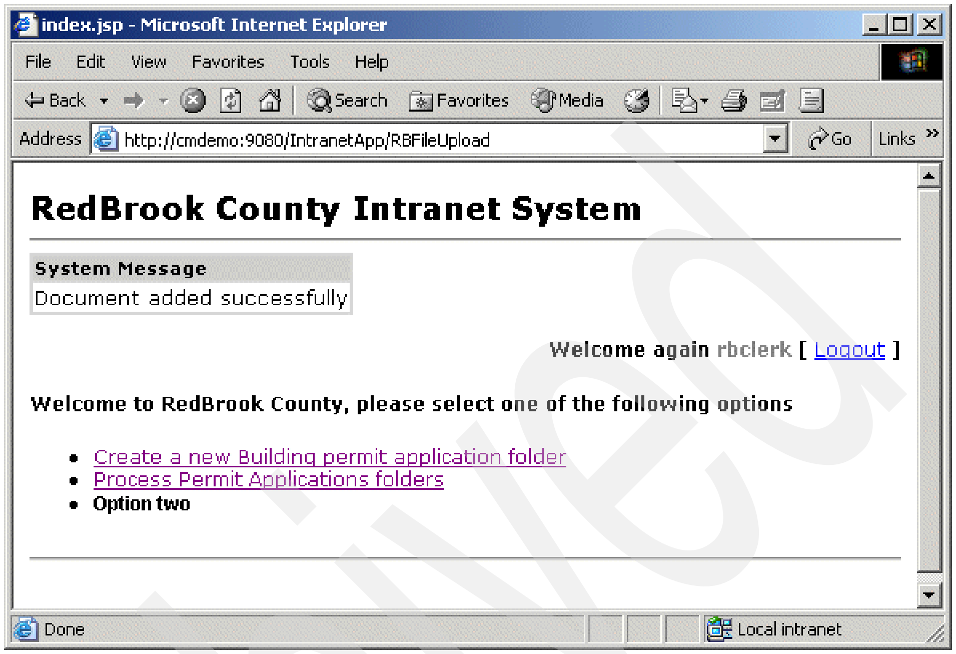
Enter attribute values to the specified document type

Attribute	Type	Value
BP_NUMBER	String	25788
BP_DOC_DESC	String	Application sample document
BP_SUBMISSION_DATE	Date	2004-07-27
Date format: yyyy-mm-dd		
File:		C:\Documents and Setti Browse...
Mime Type:		RTF Document
Import		

RTF Document
Framemaker
AFP Document
MO:DCA Document
Lotus 1-2-3

Figure A-23 Add a new item to the BP Folder

Input the required data and import the document (Figure A-24).


index.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBFileUpload
Go Links
Done
Local intranet

RedBrook County Intranet System

System Message
Document added successfully

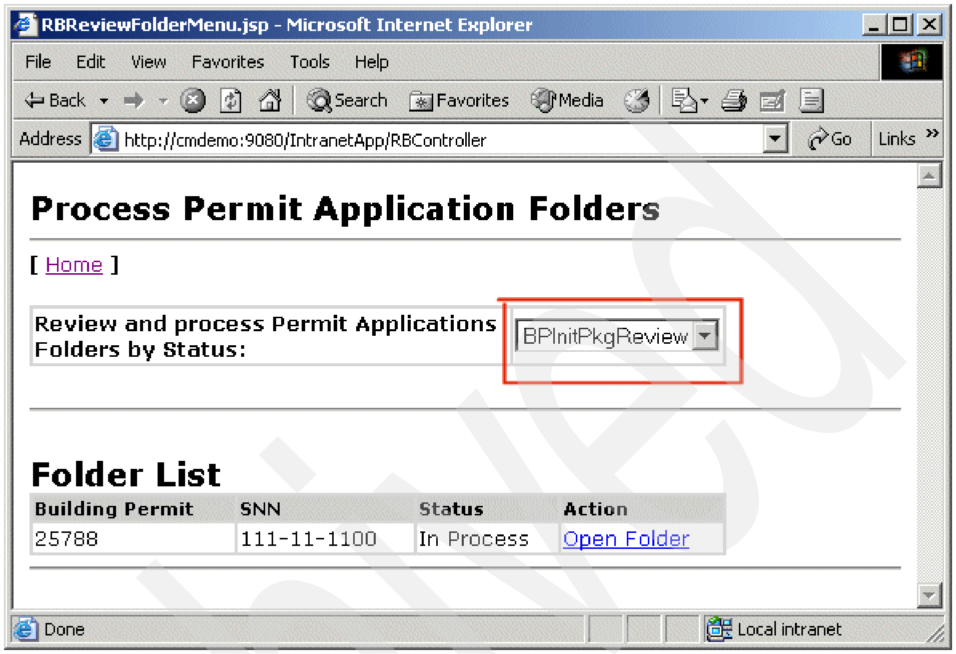
Welcome again rbclerk [Logout]

Welcome to RedBrook County, please select one of the following options

- Create a new Building permit application folder
- Process Permit Applications folders
- Option two

Figure A-24 Document imported successfully

Repeat the import steps for the two remaining item types and verify that the folder is shown at the BPInitPkgReview work node (Figure A-25).


RBReviewFolderMenu.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController
Go Links
Done
Local intranet

Process Permit Application Folders

[Home]

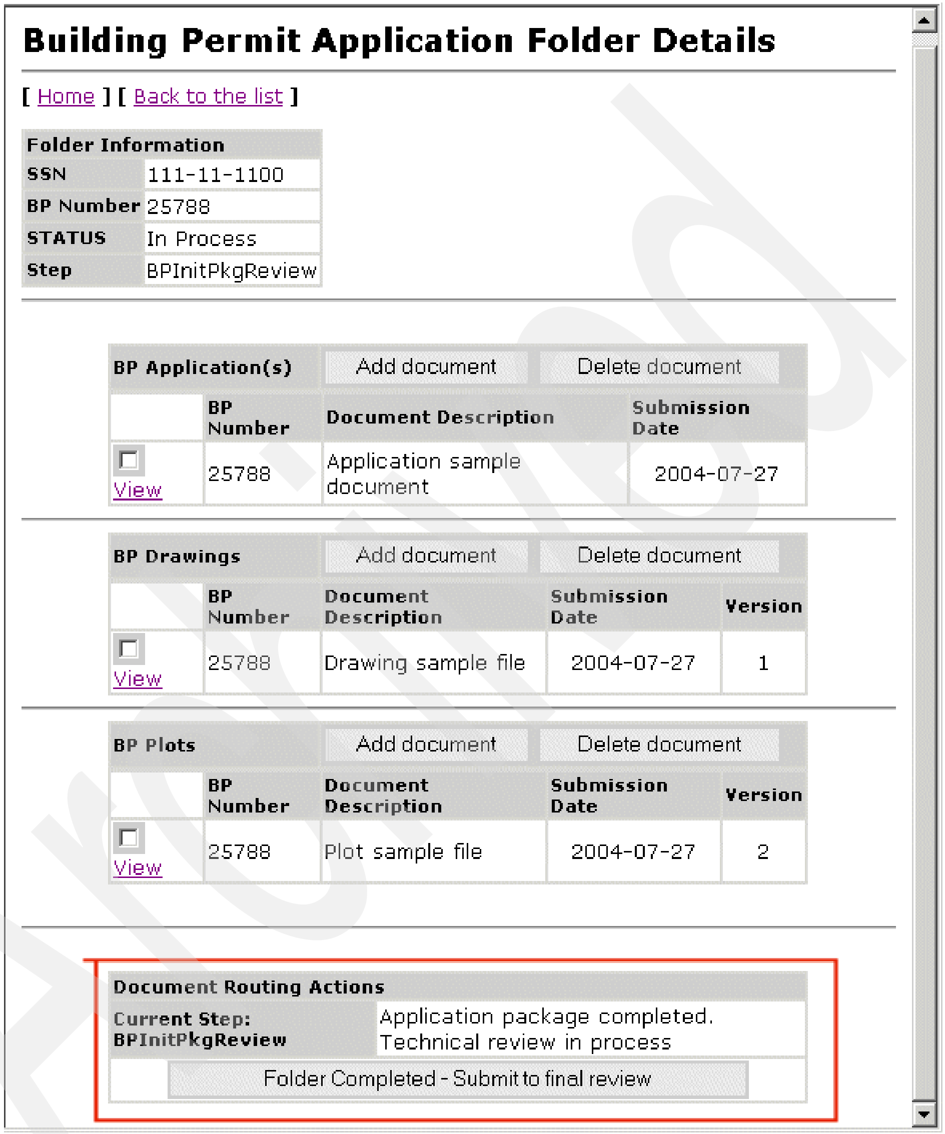
Review and process Permit Applications Folders by Status:	BPInitPkgReview

Folder List

Building Permit	SNN	Status	Action
25788	111-11-1100	In Process	Open Folder

Figure A-25 Continue the BP Folder approval process

At this point, the documentation is reviewed before submitting it for a final review. Select the link, **Open the folder** (Figure A-26)



Building Permit Application Folder Details

[Home] [Back to the list]

Folder Information	
SSN	111-11-1100
BP Number	25788
STATUS	In Process
Step	BPInitPkgReview

BP Application(s) — Add document — Delete document

	BP Number	Document Description	Submission Date
☐ View	25788	Application sample document	2004-07-27

BP Drawings — Add document — Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	25788	Drawing sample file	2004-07-27	1

BP Plots — Add document — Delete document

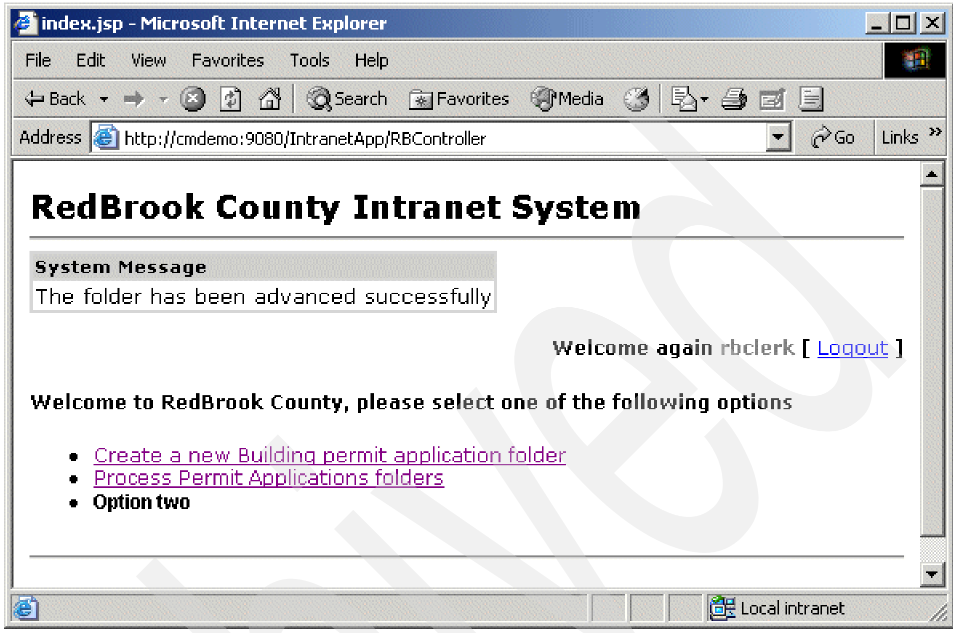
	BP Number	Document Description	Submission Date	Version
☐ View	25788	Plot sample file	2004-07-27	2

Document Routing Actions	
Current Step: BPInitPkgReview	Application package completed. Technical review in process

Folder Completed - Submit to final review

Figure A-26 Initial application package review

Once the Application folder has been reviewed, use the **Action** button to submit the folder for a final review (Figure A-27).


index.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/IntranetApp/RBController
Go Links
Local intranet

RedBrook County Intranet System

System Message
The folder has been advanced successfully

Welcome again rbclerk [Logout]

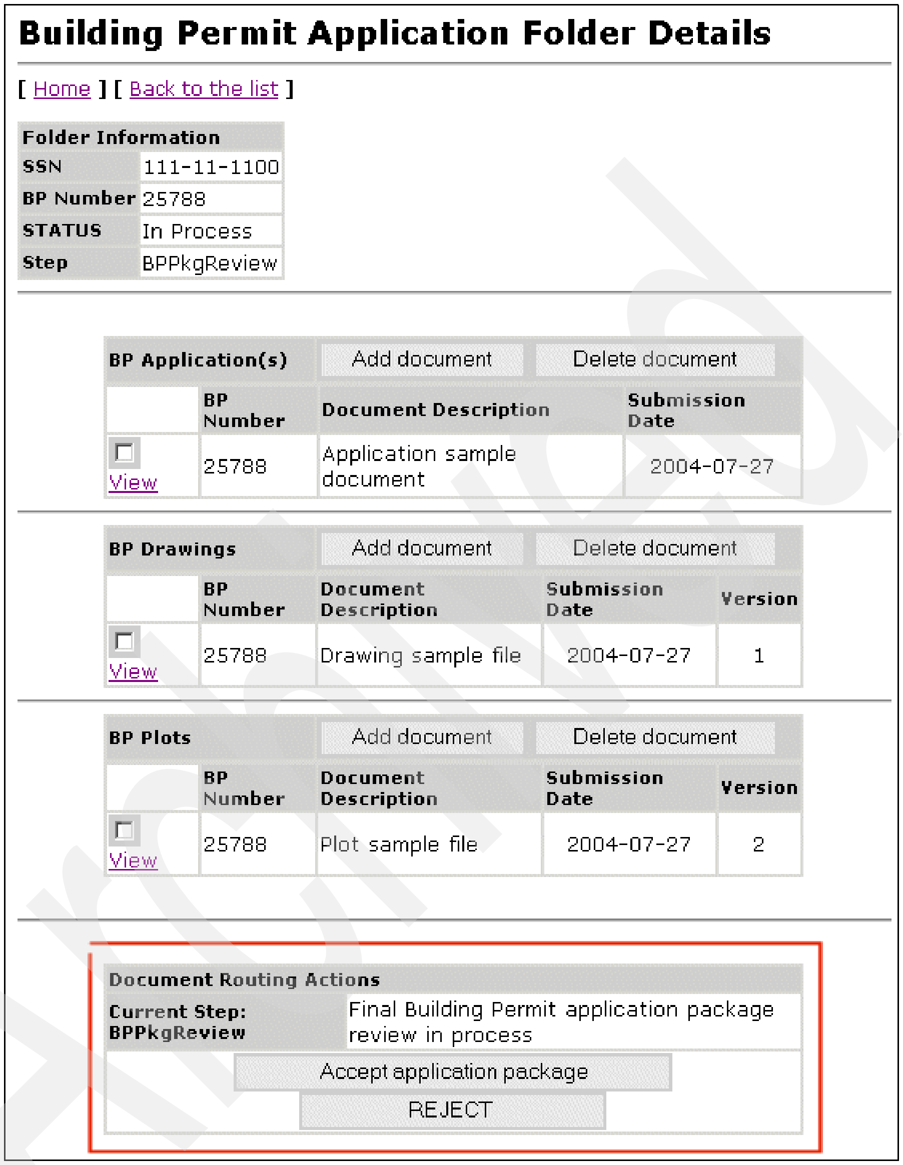
Welcome to RedBrook County, please select one of the following options

- Create a new Building permit application folder
- Process Permit Applications folders
- Option two

Figure A-27 BP Folder advanced successfully

The action advanced the BP Folder to the BPPkgReview work node. If you are using the user ID, `rbclerk`, logout and login using `rbsupervisor`. The `rbclerk` user doesn’t have the privileges to approve the final review.

Open the BP Folder detail in order to approve or reject the application (Figure A-28).



Building Permit Application Folder Details

[Home] [Back to the list]

Folder Information	
SSN	111-11-1100
BP Number	25788
STATUS	In Process
Step	BPPkgReview

BP Application(s) — Add document — Delete document

	BP Number	Document Description	Submission Date
☐ View	25788	Application sample document	2004-07-27

BP Drawings — Add document — Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	25788	Drawing sample file	2004-07-27	1

BP Plots — Add document — Delete document

	BP Number	Document Description	Submission Date	Version
☐ View	25788	Plot sample file	2004-07-27	2

Document Routing Actions	
Current Step: BPPkgReview	Final Building Permit application package review in process

Accept application package

REJECT

Figure A-28 Application folder final review

If the application is rejected, the reviewer must input a reject reason and the folder is routed to the work node named BPRejected.

Accept the application package.

At this point the approval process is complete and the legacy system is updated.

A.6.2 External application navigation

In order to access to the Internet application, use one of the following URLs.

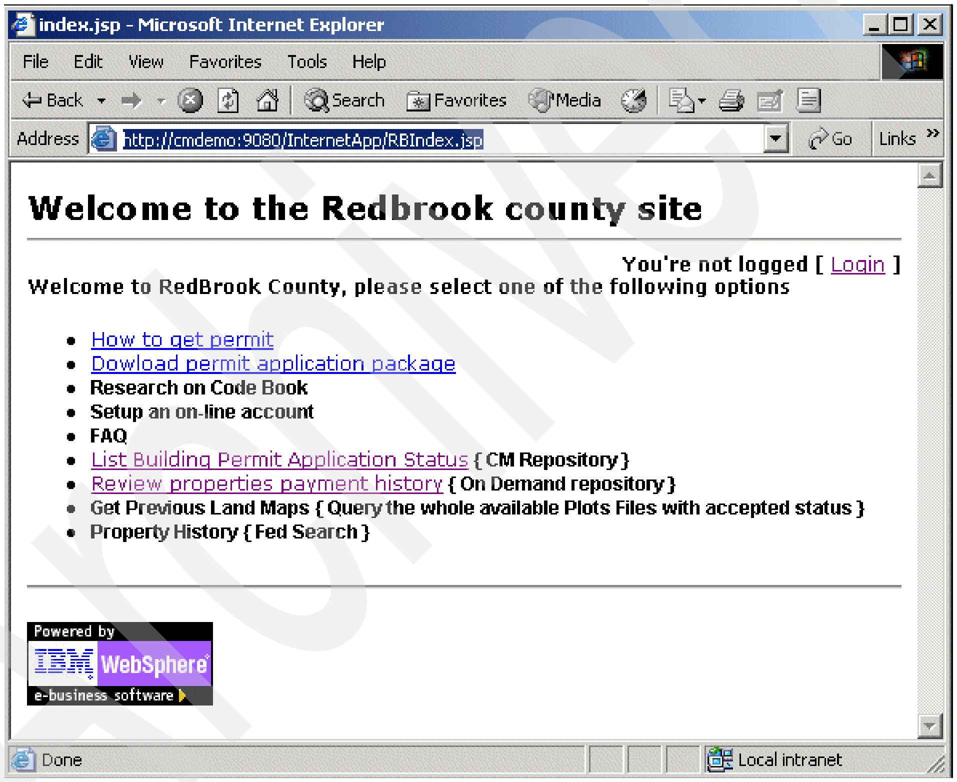
- If you are accessing the sample application imported from the EAR file:

  `http://localhost:9080/RedBrook/RBIndex.jsp`

- If the application was coded following the step-by step-procedure, use the following URL:

  `http://localhost:9080/InternetApp/RBIndex.jsp`

The Welcome screen is shown (Figure A-29).


index.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://cmdemo:9080/InternetApp/RBIndex.jsp
Go Links
Done
Local intranet

Welcome to the Redbrook county site

You're not logged [Login]

Welcome to RedBrook County, please select one of the following options

- How to get permit
- Dowload permit application package
- Research on Code Book
- Setup an on-line account
- FAQ
- List Building Permit Application Status { CM Repository }
- Review properties payment history { On Demand repository }
- Get Previous Land Maps { Query the whole available Plots Files with accepted status }
- Property History { Fed Search }

Powered by IBM WebSphere e-business software

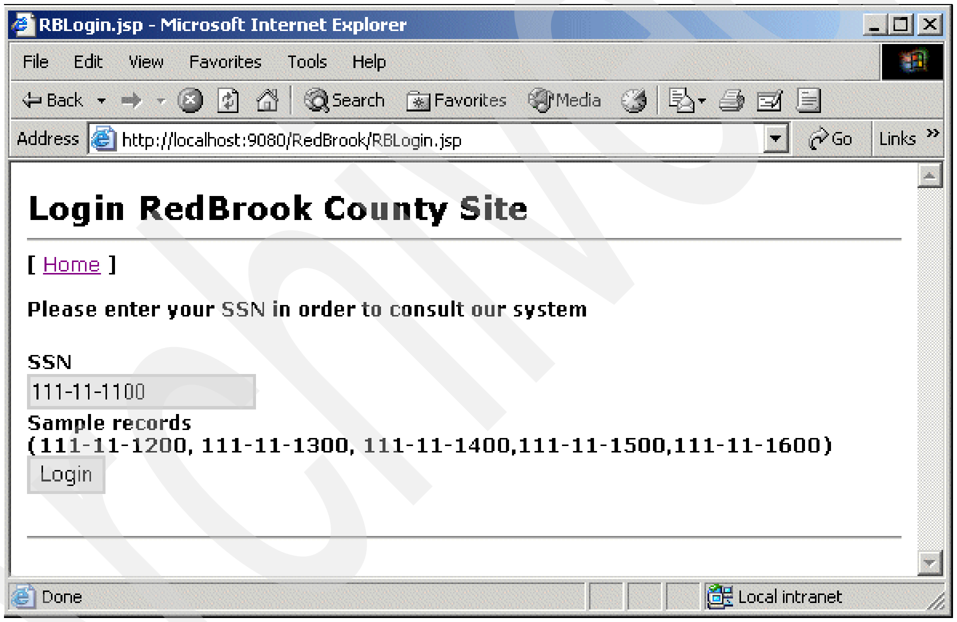
Figure A-29 Welcome screen external app

The implemented services within the Internet application are:

- **Research on Code Book:** Implements a search by content within the BP_CODE_BOOK item type.
- **List building permit application status:** This query displays the building permit folder status within the approval process.
- **Review properties payment history:** This allows you to query the IBM Content Manager OnDemand in order to retrieve some AFP files.

Select **List building permit application status** from the main menu.

If you are not logged in, the application will take you to the login page (Figure A-30).


RBLogin.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/RedBrook/RBLogin.jsp
Go Links
Done
Local intranet

Login RedBrook County Site

[Home]

Please enter your SSN in order to consult our system

SSN

111-11-1100

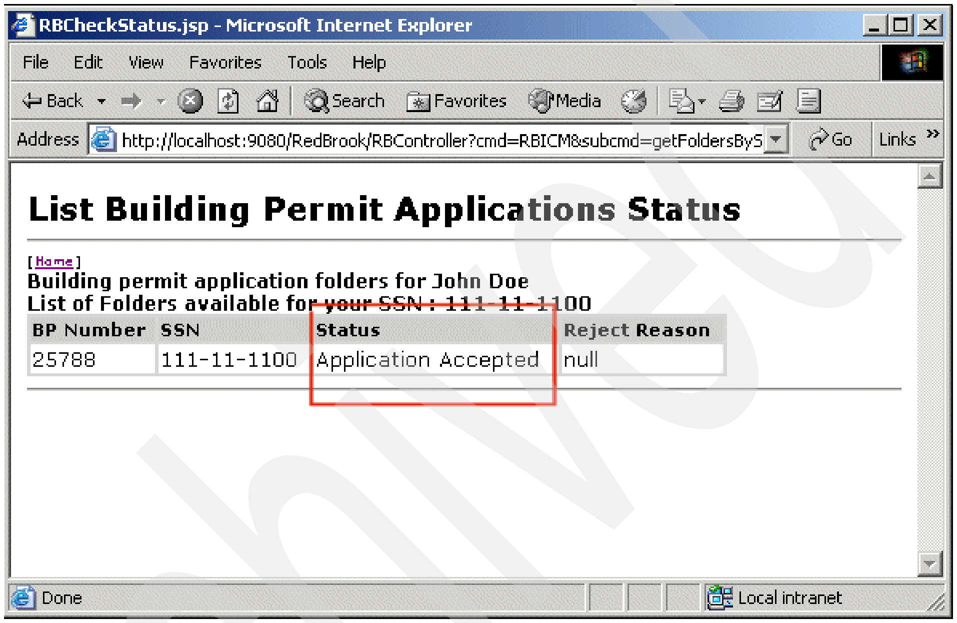
Sample records
(111-11-1200, 111-11-1300, 111-11-1400,111-11-1500,111-11-1600)

Login

Figure A-30 Login page

The user must input a valid SSN registered at the RedBrook DB2 legacy system. If you want to follow the same sample values used during the Intranet system navigation, use the default SSN and click login.

This action will take you to the main page, select again the link, **List building permit application status** (Figure A-31).


RBCheckStatus.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/RedBrook/RBController?cmd=RBICM&subcmd=getFoldersByS
Go Links
Done
Local intranet

List Building Permit Applications Status

[Home]
Building permit application folders for John Doe
List of Folders available for your SSN : 111-11-1100

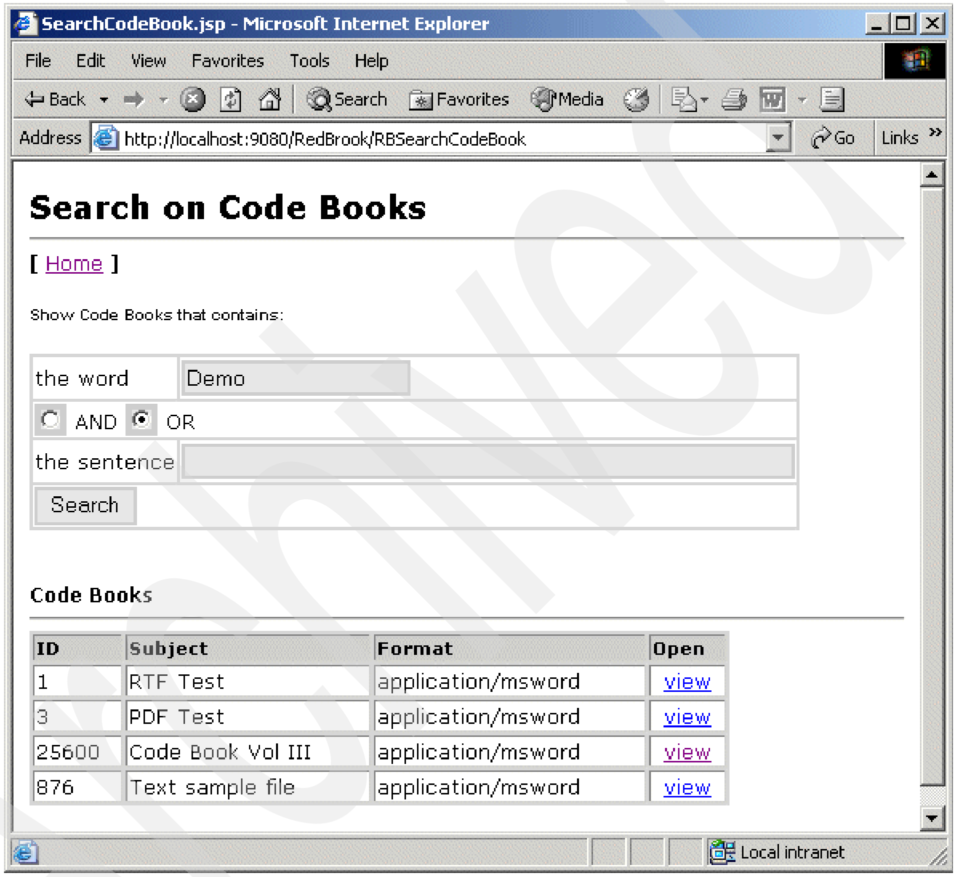
BP Number	SSN	Status	Reject Reason
25788	111-11-1100	Application Accepted	null

Figure A-31 Building permit application status

Click the **Home** link back to the main menu.

If the application package was rejected, the user would need to review the Code Books in order to find out the rules with which he is not compliant.

Select **Research on Code Book** and try some content searches (Figure A-32).


SearchCodeBook.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/RedBrook/RBSearchCodeBook
Go Links
Local intranet

Search on Code Books

[Home]

Show Code Books that contains:

the word: Demo

AND [✓ OR]

the sentence:

Search

Code Books

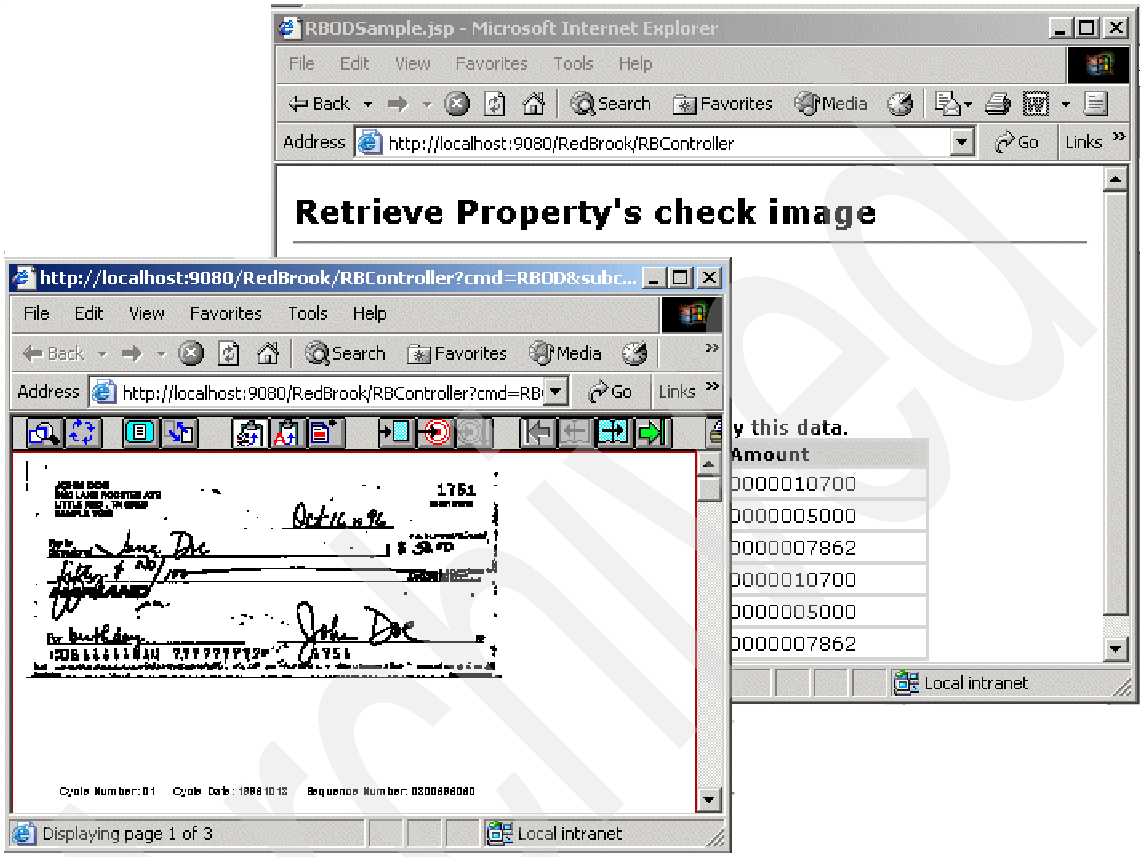
ID	Subject	Format	Open
1	RTF Test	application/msword	view
3	PDF Test	application/msword	view
25600	Code Book Vol III	application/msword	view
876	Text sample file	application/msword	view

Figure A-32 Code books content search page

Click the **Home** link and go to the main menu.

Finally, if you had configured the IBM Content Manager OnDemand system and the RedBrook county sample report data, click the link, **Review properties payment history**.

Logon if you are not already logged. Select a property number registered at the RedBrook DB2 legacy system and click **Go** to retrieve the related documents (Figure A-33).


RBODSample.jsp - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:9080/RedBrook/RBController
Go Links
http://localhost:9080/RedBrook/RBController?cmd=RBOD&subc...
Address http://localhost:9080/RedBrook/RBController?cmd=RB
Displaying page 1 of 3
Local intranet

Retrieve Property's check image

y this data.

Amount
0000010700
0000005000
0000007862
0000010700
0000005000
0000007862

Cycle Number: 01   Cycle Date: 19981018   Sequence Number: 0300888080

Figure A-33   Payment history query result page

For additional information, please refer to Chapter 8, “Building a case study application” on page 207 and Chapter 4, “Case study” on page 61.

A.7 Redbrook County legacy system database schema

The Redbrook County legacy system uses the IBM UDB DB2 relational database. There are eight tables in the legacy system:

- BUILDING_PERMIT: Building permit table
- PROPERTY: Property table
- SALES: Sale information table

- ASSESS: Assessment information table
- OWNER: Owner table
- OWNER_HISTORY: Owner history table
- TAXDISC: Tax discount table
- TAXINFO: Tax information table

Table A-2 is the database schema for the building permit table.

Table A-2 Legacy database table: BUILDING_PERMIT

Name	Data Type	Constraint Type	Constraint Name
ID	CHAR(6)	Primary Key	BP_ID_PK
PROPERTY NBR	CHAR(10)	Foreign Key	BP_PNBR_FK
ELECTRICAL	CHAR(1)		
EINSPECTIONDATE	DATE		
MECHANICAL	CHAR(1)		
MINSPECTIONDATE	DATE		
PLUMBING	CHAR(1)		
PINSPECTIONDATE	DATE		
CERTIFICATE	CHAR(1)		
CINSPECTIONDATE	DATE		
FEES	DOUBLE		
ISSUED_TO	CHAR(30)		

Table A-3 is the database schema for the property table.

Table A-3 Legacy database table: PROPERTY

Name	Data Type	Constraint Type	Constraint Name
PROPERTY_NBR	CHAR(10)	Primary Key	P_PNBR_PK
ADDRESS	VARCHAR(50)		
ZONING	INTEGER		
NBR_OF_ROOMS	INTEGER		
NBR_OF_FLOORS	INTEGER		

Name	Data Type	Constraint Type	Constraint Name
LIVING_FOOTAGE	DOUBLE		
LOT_SIZE	DOUBLE		
YEAR_BUILT	INTEGER		

Table A-4 is the database schema for the sales information table.

Table A-4 Legacy database table: SALES

Name	Data Type	Constraint Type	Constraint Name
PROPERTY_NBR	CHAR(10)	Foreign Key (to Property)	S_PNBR_FK
SALES_OR	CHAR(20)		
WHEN	DATE		
SALE_AMOUNT	DOUBLE		

Table A-5 is the database schema for assessment information table.

Table A-5 Legacy database table: ASSESS

Name	Data Type	Constraint Type	Constraint Name
PROPERTY_NBR	CHAR(10)	Foreign Key (to Property)	A_PNBR_FK
LAND_VALUE	DOUBLE		
LDATE	DATE		
BUILDING_VALUE	DOUBLE		
BDATE	DATE		
MARKET_VALUE	DOUBLE		
MDATE	DATE		
ASSESSED_VALUE	DOUBLE		
ADATE	DATE		
TAXABLE_VALUE	DOUBLE		

Name	Data Type	Constraint Type	Constraint Name
TDATE	DATE		

Table A-6 is the database schema for the owner table.

Table A-6 Legacy database table: OWNER

Name	Data Type	Constraint Type	Constraint Name
SSN	CHAR(12)	Primary Key	O_SSNBR_PK
FIRSTNAME	VARCHAR(20)		
LASTNAME	VARCHAR(20)		
STREET	VARCHAR(30)		
CITY	VARCHAR(20)		
STATE	VARCHAR(20)		
ZIPCODE	VARCHAR(10)		
PHONE	VARCHAR(10)		
EMAIL	VARCHAR(20)		

Table A-7 is the database schema for the owner history table.

Table A-7 Legacy database table: OWNER_HISTORY

Name	Data Type	Constraint Type	Constraint Name
PROPERTY_NBR	CHAR(10)	Foreign Key (to Property)	OH_PNBR_FK
SSN	CHAR(12)	Foreign Key (to Owner)	OH_SSNBR_FK
PURCHASE_DATE	DATE		
AMOUNT_PAID	DOUBLE		
SOLD_DATE	DATE		
AMOUNT_SOLD	DOUBLE		

Table A-8 is the database schema for the tax discount table.

Table A-8   Legacy database table: TAXDISC

Name	Data Type	Constraint Type	Constraint Name
PROPERTY_NBR	CHAR(10)	Foreign Key (to Property)	T_PNBR_FK
TOTAL_TAX	DOUBLE		
TAX_YEAR	CHAR(15)		
YEARLY_DISCOUNT	DOUBLE		
YEARLY_AMOUNT	DOUBLE		
YEARLY_MONTH	CHAR(15)		
QUARTERLY_DISCOUNT	DOUBLE		
QUARTERLY_AMOUNT	DOUBLE		
Q1_MONTH	CHAR(15)		
Q2_MONTH	CHAR(15)		
Q3_MONTH	CHAR(15)		
Q4_MONTH	CHAR(15)		

Table A-9 is the database schema for the tax information table.

Table A-9   Legacy database table: TAXINFO

Name	Data Type	Constraint tYPE	Constraint Name
SSN	CHAR(12)	Foreign Key (to Owner)	TI_SSNBR_FK
PAYMENT_TYPE	CHAR(1)		
AMOUNT_PAID	DOUBLE		
PAYMENT_DATE	DATE		
RECEIPT_NO	CHAR(30)		

A.8 Content Manager system definition

We implement a Content Manager system that stores and manages building permit applications and drawings of the proposed structure and plots the owners who wish to remodel. In addition, we want to use Content Manager to store the Building Permit Code Book.

The Content Manager system we implement for this case study supports the following type of documents:

- Building permit application forms (GIF image)
- Completed building permit application forms
- Structure drawings (GIF image)
- Structure plots (GIF image)
- Code book contents (PDF or doc documents)

Attributes definition

It includes the following attributes:

- BP_NUMBER
- BP_SUB_DATE
- BP_TAX_ID
- BP_DOC_DESC
- BP_DOC_VERSION
- BP_CODEBOOK_ID
- BP_CODEBOOK_SUB
- BP_REJ_DESC
- BP_STATUS

Table A-10 shows the attribute definitions.

Table A-10 Attributes

Attribute	Description	Attribute type	Type /Length
BP_NUMBER	Building permit number	Variable char	Other / 12
BP_SUB_DATE	Submission date	Date	
BP_TAX_ID	Social security number	Variable char	Other / 12
BP_DOC_DESC	Document description	Variable char	Other / 200
BP_DOC_VERSION	Document version	Variable char	Other / 20
BP_CODEBOOK_ID	Code Book ID	Variable char	Other / 12
BP_CODEBOOK_SUB	Code Book description	Variable char	Other / 12
BP_REJ_DESC	BP application reject description	Variable char	Other / 254
BP_STATUS	BP folder process status	Variable character	Other / 20

Folder definition

The Content Manager system uses the following item types:

- BP_FOLDER
- BP_APPLICATION
- BP_DRAWING
- BP_PLOT
- BP_CODE_BOOK

Table A-11 shows the item type definition.

Table A-11 Item types definition

Name	**Description**	**Attributes**
BP_FOLDER (ICMBASE)	Folder that contains all owner's documents	BP_NUMBER[a] BP_TAX_ID BP_STATUS BP_REJ_DESC
BP_APPLICATION (ICMBASE)	BP_APPLICATION document	BP_NUMBER[a] BP_DOC_DESC BP_SUB_DATE
BP_DRAWING (ICMBASE)	BP_DRAWING document	BP_NUMBER[a] BP_DOC_DESC BP_SUB_DATE BP_DOC_VERSION
BP_PLOT (ICMBASE)	Plot drawings for the application	BP_NUMBER[a] BP_DOC_DESC BP_SUB_DATE BP_DOC_VERSION
BP_CODE_BOOK (ICMBASE, ICMBASETEXT)	Code books	BP_CODEBOOK_ID BP_CODEBOOK_SUB

a. Required, represents item.

Auto-linking is set for BP_APPLICATION, BP_DRAWING, and BP_PLOT as follows:

- Link type: Folder contains.
- Link to: BP_FOLDER.
- Associated attributes: BP_NUMBER.

A.9 OnDemand system definition

The OnDemand system we implement for the Redbrook County consists of folders, application groups, and reports. Each folder consists of several reports grouped under a specific application group.

We have three folders for the OnDemand system. They are:

- Property Owner
- Permits
- Tax Information

A.9.1 Property Owner folder

The setup of the Property Owner folder is as follows:

- Folder Name = Property Owner
- Application Group = PropertyOwner
- Applications: Property, Owner hist, and SalesInfo reports

Table A-12 shows the definitions for the Property report.

Table A-12 OnDemand report definition: Property

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
addr	string	var 40	filter	Address
zoning	integer	fixed 2	index	Zone
yearbuilt	date		date	Year Built (mm/dd/yy)

Table A-13 shows the definitions for the Owner History report.

Table A-13 OnDemand report definition: Owner History

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
ssn	string	fixed 10	index	Social Security No
pdate	date		filter	Purchase Date (mm/dd/yy)
solddate	date		filter	Sold Date (mm/dd/yy)

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
fname	string	20	filter	Owner first name
lname	string	20	filter	Owner last name
phone	string	fixed 11	index	Owner phone Nbr

Notice that Table A-13 combines attributes from the Owner and Owner History tables of the database schema stored in DB2. See A.7, “Redbrook County legacy system database schema” on page 525 for a complete description of the database schema.

Table A-14 shows the definition of the Salesinfo report.

Table A-14 OnDemand report definition: Salesinfo

Name	Data Type	Length	Index Type	Folder Name
propnbr	**string**	**var 10**	**index**	**Property Number**
salesor	**string**	**fixed 11**	**index**	**Social Security No**
salesdate	**date**		**filter**	**Sales Date (mm/dd/yy)**
salesamt	**decimal**	**10**	**filter**	**Sales amount**

A.9.2 Permits

The setup of the Permits folder is as follows:

- Folder Name = Permits
- Application Group = Permitsr
- Applications: PropertyOwner and Permits reports

Table A-15 shows the definition for PropertyOwner report.

Table A-15 OnDemand report definition: PropertyOwner

Name	Data Type	Length	Index Type	Folder Name
propnbr	**string**	**var 10**	**index**	**Property Number**
addr	**string**	**var 40**	**filter**	**Address**

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
zoning	integer	fixed 2	index	Zone
yearbuilt	date		date	Year Built (mm/dd/yy)
fname	string	20	filter	Owner first name
lname	string	20	filter	Owner last name
phone	string	fixed 11	index	Owner phone Nbr

Notice that Table A-15 combines attributes from the Property and Owner tables of the database schema.

Table A-16 shows the definition of the Permits report.

Table A-16 OnDemand report definition: Permits

Name	Data Type	Length	Index Type	Folder Name
id	int	fixed 6	index	Permit #
propnbr	string	var 10	index	Fees paid
einspection	char	fixed 1	filter	Electrical inspection flag
edate	date		filter	Electrical inspection date (mm/dd/yy)
minspection	char	fixed 1	filter	Mechanical inspection flag
mdate	date		filter	Mechanical inspection date (mm/dd/yy)
pinspection	char	fixed 1	filter	Plumbing inspection flag

Name	Data Type	Length	Index Type	Folder Name
id	int	fixed 6	index	Permit #
propnbr	string	var 10	index	Fees paid
pdate	date		filter	Plumbing inspection date (mm/dd/yy)
cinspection	char	fixed 1	filter	Certificate inspection flag
cdate	date		filter	Certifi inspection date (mm/dd/yy)

A.9.3 Tax Information folder

The setup of the Tax Information folder is as follows:

- Folder Name = Tax Information
- Application Group = Tax Assessment
- Applications: Property, Assessment, Tax Discount, and CheckImages reports

Table A-17 shows the definition of the Property report.

Table A-17 OnDemand report definition: Property

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
addr	string	var 40	filter	Address

Table A-18 shows the definition of the Assessment report.

Table A-18 Assessment

Name	Data Type	Length	Index Type	Folder Name
propnbr	string	var 10	index	Property Number
taxvalue	decimal	10	filter	Taxable Value
tdate	date		filter	Tax Date (mm/dd/yy)
avalue	decimal	10	filter	Assessment Value

Name	Data Type	Length	Index Type	Folder Name
propnbr	**string**	**var 10**	**index**	**Property Number**
adate	**date**		**filter**	**Assessment Date (mm/dd/yy)**

Table A-19 shows the definition of the TaxDiscount report.

Table A-19 OnDemand report definition: TaxDiscount

Name	Data Type	Length	Index Type	Folder Name
propnbr	**string**	**var 10**	**index**	**Property Number**
totaltax	**decimal**	**10**	**filter**	**Total tax**
taxyear	**string**	**var 4**	**filter**	**Tax Year**

Table A-20 shows the definition of the CheckImages report.

Table A-20 OnDemand report definition: Checkimages

Name	Data Type	Length	Index Type	Folder Name
propnbr	**string**	**var 10**	**index**	**Property Number**
checkamt	**decimal**	**10**	**filter**	**Amount received**
datepaid	**date**		**filter**	**Date received**
ssn	**string**	**fixed 11**	**filter**	**Social Security No**

A.9.4 Attributes mapping among DB2, OnDemand, and CM

Table A-21 shows Content Manager and OnDemand attributes and folders as they relate to DB2 attributes defined in the DB2 database schema (see A.7, “Redbrook County legacy system database schema” on page 525).

Table A-21 DB2 attributes and CM and OnDemand attributes and folders

DB2	OnDemand	Content Manager
PROPERTY NBR	pronbr	
ADDRESS	addr	
ZONING	zoning	
YEAR BUILT	yearbuilt	

DB2	OnDemand	Content Manager
SOCIAL SECURITY NO	ssn	BP_TAX_ID
PURCHASE DATE	pdate	
SOLD DATE	solddate	
FIRST NAME	fname	
LAST NAME	lname	
PHONE	phone	
SALE_OR	saleor	
SALE DATE	salesdate	
AMOUNT SOLD	salesamt	
ID	id	
ELECTRICAL	einspection	
EINSPECTIONDATE	edate	
MECHANICAL	minspection	
MINSPECTIONDATE	mdate	
PLUMBING	pinspection	
PINSPECTIONDATE	pdate	
CERTIFICATE	cinspection	
CINSPECTIONDATE	cdate	
TAXABLE VALUE	taxvalue	
TDATE	tdate	
ASSESSED VALUE	avalue	
TOTAL TAX	totaltax	
TAX YEAR	taxyear	

Sample code

This appendix provides the sample code used in the redbook.

The samples include:

- Quick start sample code
- Generic Web application sample code
- Building permit application folder use case

B.1 Quick start sample code

This section provides some of the sample code used in the quick start for Web development portion of the redbook.

Add item types: CMaddItemType.jsp

Example B-1 shows the complete JSP file that can be used to create the appropriate item types for the Redbrook County building permit application. Note that the Library Server name, user ID, and password are hard coded. Change the values according to your system configuration.

Example: B-1 CMaddItemType.jsp

```
<%@ page import="com.ibm.mm.sdk.server.*,
         com.ibm.mm.sdk.common.*"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
   type="text/css">
<TITLE>CMaddItemType.jsp</TITLE>
</HEAD>
<BODY>
<P>Adding item types.</P>
<%
DKDatastoreICM dsICM = new DKDatastoreICM();
dsICM.connect("LSDB","LSADMIN","lspasswd","");
// create a new Item Type Definition object.
// Get the datastore definition object from the connected datastore.
// Retrieve a reference to the desired attribute definitions from the dtastore
definition
// create a new Item Type Definition object.
// More detailed description than name.
// Add the desired attribute to the item type defintion.
// Create a new item type relation object
// Add / Save the Item Type Definition.
// Set the source item
```

```
DKItemTypeDefICM itemType = new DKItemTypeDefICM(dsICM);
DKItemTypeRelationDefICM itemTypeRel = new DKItemTypeRelationDefICM(dsICM);

////////////////////////////////////////////////////////////
// DEFINING ITEM TYPE BP_FOLDER
////////////////////////////////////////////////////////////
    DKDatastoreDefICM dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);

   itemType = new DKItemTypeDefICM(dsICM);
    itemType.setName("BP_FOLDER");
    itemType.setDescription("BP_FOLDER folder that will contain all owner's
documents ");
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    DKAttrDefICM attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_TAX_ID");
    attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_NUMBER");
    attr.setUnique(false);
    attr.setRepresentative(true);
    attr.setNullable(false);
   itemType.addAttr(attr);

       attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_STATUS");
       itemType.addAttr(attr);

   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_REJ_DESC");
       itemType.addAttr(attr);
   itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
    itemType.add();
   %>
   Item Type BP_FOLDER has been added to Content Manager<br>
   <%

   itemTypeRel = new DKItemTypeRelationDefICM(dsICM);

    itemTypeRel.setSourceItemTypeID(dsDefICM.getEntityIdByName("BP_FOLDER"));
   itemTypeRel.setTargetItemTypeID(dsDefICM.getEntityIdByName("ICMBASE"));

    itemTypeRel.setDefaultACLCode(3);
    dsDefICM.add(itemTypeRel);

////////////////////////////////////////////////////////////
// DEFINING ITEM TYPE BP_PLOT
////////////////////////////////////////////////////////////
    dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    itemType = new DKItemTypeDefICM(dsICM);
    itemType.setName("BP_PLOT");
    itemType.setDescription("BP_PLOT is the type for all joined plots the
application ");
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_NUMBER");
    attr.setUnique(false);
        attr.setNullable(false);

   itemType.addAttr(attr);

   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_DESC");
   attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_SUB_DATE");
   attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_VERSION");
   attr.setUnique(false);
   itemType.addAttr(attr);
   itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
    itemType.add();
   %>
   Item Type BP_PLOT has been added to Content Manager<br>
   <%
   itemTypeRel = new DKItemTypeRelationDefICM(dsICM);
    itemTypeRel.setSourceItemTypeID(dsDefICM.getEntityIdByName("BP_PLOT"));
   itemTypeRel.setTargetItemTypeID(dsDefICM.getEntityIdByName("ICMBASE"));

    itemTypeRel.setDefaultACLCode(3);
    dsDefICM.add(itemTypeRel);

////////////////////////////////////////////////////////////
// DEFINING ITEM TYPE BP_DRAWING
////////////////////////////////////////////////////////////
    dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    itemType = new DKItemTypeDefICM(dsICM);
    itemType.setName("BP_DRAWING");
    itemType.setDescription("BP_DRAWING document ");
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_NUMBER");
    attr.setUnique(false);
        attr.setNullable(false);
   itemType.addAttr(attr);
   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_DESC");
   attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_SUB_DATE");
   attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_VERSION");
   attr.setUnique(false);
   itemType.addAttr(attr);
   itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
    itemType.add();
   %>
   Item Type BP_DRAWING has been added to Content Manager<br>
   <%
   itemTypeRel = new DKItemTypeRelationDefICM(dsICM);
    itemTypeRel.setSourceItemTypeID(dsDefICM.getEntityIdByName("BP_DRAWING"));
   itemTypeRel.setTargetItemTypeID(dsDefICM.getEntityIdByName("ICMBASE"));

    itemTypeRel.setDefaultACLCode(3);
    dsDefICM.add(itemTypeRel);

////////////////////////////////////////////////////////////
// DEFINING ITEM TYPE BP_APPLICATION
////////////////////////////////////////////////////////////
    dsDefICM = (DKDatastoreDefICM) dsICM.datastoreDef();
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    itemType = new DKItemTypeDefICM(dsICM);
    itemType.setName("BP_APPLICATION");
    itemType.setDescription("BP_APPLICATION document ");
   System.out.println("After Retrieving attributes and dsDefICM =" +
dsDefICM);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_NUMBER");
    attr.setUnique(false);
   itemType.addAttr(attr);
   attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_DOC_DESC");
   attr.setUnique(false);
   itemType.addAttr(attr);
    attr = (DKAttrDefICM) dsDefICM.retrieveAttr("BP_SUB_DATE");
   attr.setUnique(false);
   itemType.addAttr(attr);
    itemType.setClassification(DKConstantICM.DK_ICM_ITEMTYPE_CLASS_DOC_MODEL);
    itemType.add();
   %>
   Item Type BP_APPLICATION has been added to Content Manager<br>
   <%
   itemTypeRel = new DKItemTypeRelationDefICM(dsICM);
```

```
itemTypeRel.setSourceItemTypeID(dsDefICM.getEntityIdByName("BP_APPLICATION"));
   itemTypeRel.setTargetItemTypeID(dsDefICM.getEntityIdByName("ICMBASE"));

    itemTypeRel.setDefaultACLCode(3);
    dsDefICM.add(itemTypeRel);


dsICM.disconnect();
dsICM.destroy();

%>


</BODY>
</HTML>
```

Retrieve BP_PLOTs: CMretrievePlots.jsp

Example B-2 shows the complete JSP file that can be used to retrieve the BP_PLOTs item type for the Redbrook County building permit application. Note that the Library Server name, user ID, and password are hard coded. Change the values according to your system configuration.

Example: B-2 CMretrievePlots.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>

<%@ page
   language="java"
   contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"
%>
<!--  IMPORT -->
<%@ page import="com.ibm.mm.sdk.server.*,
         com.ibm.mm.sdk.common.*,
         java.io.*"
%>


<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="/II4CProject/theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>View Plots</TITLE>

<Script Language="JavaScript">
   function ViewObject(fpid)
```

```
      {

window.open("/II4CProject/ViewObject.jsp?PID="+fpid,"_blank","toolbar=0,scrollb
ars=0,location=0,statusbar=1,menubar=0,resizable=1,width=500,height=670");
   return false;
      }
</Script>


</HEAD>
<BODY>
<P>Retrieving and Viewing items of type plots.</P>
<!--  BEGINNING OF CODE -->

<%
DKDatastoreICM dsICM = new DKDatastoreICM();
// Create new datastore object.
dsICM.connect("LSDB","LSADMIN","lspasswd","");
// Connect to the datastore.

%>
retrieving plots
<br>
Connected to datastore <%=dsICM.datastoreName()%> UserName
<%=dsICM.userName()%><br>
<%

String query = "( /BP_PLOT )";
System.out.println("    Query:  "+query);
// Specify Search / Query Options
DKNVPair options[] = new DKNVPair[3];
options[0] = new DKNVPair(DKConstant.DK_CM_PARM_MAX_RESULTS, "0"); // No
Maximum (Default)
                // Specify max using a string value.
options[1] = new DKNVPair(DKConstant.DK_CM_PARM_RETRIEVE,    new
Integer(DKConstant.DK_CM_CONTENT_ATTRONLY));
                // Specify any Retrieval Options desired.  Default is ATTRONLY.
options[2] = new DKNVPair(DKConstant.DK_CM_PARM_END, null);
                // Must mark the end of the NVPair

dkResultSetCursor cursor = dsICM.execute(query,
DKConstantICM.DK_CM_XQPE_QL_TYPE, options);
System.out.println("Executed:  Obtain all Building Permits Plots.");
System.out.println("Accessing Result Set Cursor...");

DKDDO  ddo;
String itemId = "";
String dataValue="";
short attnum ;
```

```
%>
<TABLE BORDER="8">
<%

while((ddo = cursor.fetchNext())!=null)
   {
   System.out.println("In while loop...");
           // Get the next ddo & stop when ddo == null.
   itemId = ((DKPidICM)ddo.getPidObject()).getItemId();
   System.out.println("In while loop... and itemid "+ itemId);
   attnum =ddo.dataCount();
      System.out.println("In while loop... and attnum  "+ attnum);
   //ddo.retrieve(DKConstant.DK_CM_CONTENT_YES);

   //if (retrieveItemType.equalsIgnoreCase("BP_FOLDER"))
   // ddo.retrieve(DKConstant.DK_CM_CONTENT_LINKS_OUTBOUND);
   //else

      ddo.retrieve(DKConstant.DK_CM_CONTENT_YES);
      System.out.println("after ddo.retrieve  "+ ddo);
      //DKParts dkParts = (DKParts)
ddo.getData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS
));
      System.out.println("     - Item ID:  "+itemId+"
("+ddo.getPidObject().getObjectType()+")");
        // Print Item ID & Object Type
      %>
      <TR><TD><HR> Item of type  <%=ddo.getPidObject().getObjectType()%>
      <%
      // DKParts dkParts2 = (DKParts)
ddo.getData(ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS
));

%>
<TABLE BORDER="1">
<%

    for (short i = 1;i <= attnum-1; i++)
    {
      Short type   = (Short)
ddo.getDataPropertyByName(i,DKConstant.DK_CM_PROPERTY_TYPE);
         if (!(ddo.getDataName(i).equals("ICMParts1029")))
         {
         %><TR><td><%=ddo.getDataName(i)%></TD><td>   <%= ddo.getData(i)%>
</td></TR><%
         }//if
      }// end of for loop
%>
</table>
<%

   short dataid =
ddo.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,DKConstant.DK_CM_DKPARTS);
    if(dataid > 0){
    // if a DKParts collection exists, print contents
        DKParts dkParts = (DKParts) ddo.getData(dataid);
      // obtain the DKParts collection
        System.out.println("Parts:");
        System.out.println("Number:  "+dkParts.cardinality());

      dkIterator iter = dkParts.createIterator();
      // Create an iterator to go through Collection
        while(iter.more())
        {
            // While there are children, print list
            DKDDO part = (DKDDO)iter.next();
            part.retrieve();
            DKPidICM PIDx =(DKPidICM) part.getPidObject();
            String pidxdo = PIDx.pidString();
            // Move pointer to next part & return that object.
            System.out.println("Part:     Type:
"+part.getPidObject().getObjectType());
            System.out.println("Item ID:
"+((DKPidICM)part.getPidObject()).getItemId());
            System.out.println("");

            %><td>
              <a style="cursor:hand"
onClick="ViewObject('<%=java.net.URLEncoder.encode(pidxdo)%>');">View this
plot</a>

            <!-- <img
src="/II4CProject/servlet/RetrievePlots?PID=<%=pidxdo%>"></td>--><%
            }//while iter
    }//if dataid>0


  %>
      </TD>
      <%



%>
</TR>
<%
}// end of while
%>
</TABLE>
```

```
<%
                  // IMPORTANT STEP:
cursor.destroy();      // Close & Destroy Cursor, Ending Implied Transaction.
System.out.println("Accessed Result Set Cursor.");
dsICM.disconnect();
dsICM.destroy();
%>
<!--  END OF CODE -->

<HR>
<BR>
<table align="center">
<tr>
<td align="center">
<!-- <a href="/II4CProject/connectCM.jsp">Return to CM Content Server menu</a>
-->
</td>
<td>
<!-- <a href="/II4CProject/index.html">Connect to other content servers</a> -->
</td>
</tr>
</table>
<br>
<IMG src="/II4CProject/images/poweredby_WebSphere.gif" width="116" height="52"
border="0">
</BODY>
</HTML>
```

View objects: ViewObject.jsp

Example B-3 shows the JSP file that can be used as a container for the image (object) to display. It retrieves the part ID passed as a request parameter, and calls a servlet to returns the array of bytes that form that object.

Example: B-3 ViewObject.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<HTML>
<HEAD>
<%@ page
language="java"
contentType="text/html; charset=ISO-8859-1"
pageEncoding="ISO-8859-1"
%>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="theme/Master.css" rel="stylesheet"
   type="text/css">
```

```
<TITLE>ViewObject.jsp</TITLE>
</HEAD>
<BODY>

<%
String pidxdo = request.getParameter("PID");
%>
<P><%=pidxdo%><P>


<img src="/II4CProject/servlet/RetrievePlots?PID=<%=pidxdo%>">

</BODY>
</HTML>
```

Retrieve plots servlet: RetrievePlots.java

Example B-4 shows the Java servlet file that can be used to retrieve plots. The servlet retrieves the part using its part ID and gets its contents in an array of bytes that will be returned to the calling JSP and displayed inside an <img> tag.

Note that the Library Server name, user ID, and password are hard coded. Change the values according to your system configuration.

Example: B-4 RetrievePlost.java

```
import com.ibm.mm.sdk.common.*;
import com.ibm.mm.sdk.server.*;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class RetrievePlots extends HttpServlet
    implements DKConstantDL
{

    public RetrievePlots()
    {
    }

    public void destroy()
    {
        try
        {
          dsICM.disconnect();
          dsICM.destroy();
        }
        catch(DKException dkexception)
        {
            System.out.println(dkexception.name() + "<br>" +
dkexception.getMessage());
            dkexception.printStackTrace();
        }
        catch(Exception exception)
        {
            System.out.println("Exception message " + exception.getMessage());
            exception.printStackTrace();
        }
    }

    public synchronized void doGet(HttpServletRequest httpservletrequest,
HttpServletResponse httpservletresponse)
        throws ServletException, IOException
    {
        ServletOutputStream servletoutputstream =
httpservletresponse.getOutputStream();
        httpservletresponse.setContentType("text/html");
        try
        {

          DKDatastoreICM dsICM = new DKDatastoreICM();
          dsICM.connect("LSDB","LSADMIN","lspasswd","");

            javax.servlet.http.HttpSession httpsession =
httpservletrequest.getSession(false);
            String s = httpservletrequest.getParameter("PID");
          System.out.println("s " + s);
          DKPidICM pidx = new DKPidICM(s);
          System.out.println("pidx " + pidx);
            //DKPidXDODL dkpidxdodl = new DKPidXDODL(s);
          System.out.println("dsICM " + dsICM);
            //DKBlobDL dkblobdl = new DKBlobDL(dsICM);
          DKLobICM blob =new DKLobICM(dsICM);
          blob.setDatastore(dsICM);
          blob.setPidObject(pidx);
          blob.retrieve();
            httpservletresponse.setContentType(blob.getMimeType());
          System.out.println("blob.getMimeType() " + blob.getMimeType());

            long nl =blob.getSize();
            int n= (new Long(nl)).intValue();
            byte abyte0[] =new byte[n];
            abyte0 = blob.getContent();
            servletoutputstream.write(abyte0);
        }
        catch(DKException dkexception)
        {
            servletoutputstream.println(dkexception.name() + "<br>" +
dkexception.getMessage());
            dkexception.printStackTrace();
        }
        catch(Exception exception)
        {
            servletoutputstream.println("Exception message " +
exception.getMessage());
            exception.printStackTrace();
        }
    }

    public synchronized void doPost(HttpServletRequest httpservletrequest,
HttpServletResponse httpservletresponse)
        throws ServletException, IOException
    {
        doGet(httpservletrequest, httpservletresponse);
    }

    public void init(ServletConfig servletconfig)
        throws ServletException
    {
        super.init(servletconfig);

    }

   DKDatastoreICM dsICM;
}
```

B.2 Generic Web application sample code

The following sample codes are developed for the generic Content Manager Web application using non-visual Java beans.

Controller servlet: RBController.java

Example B-5 shows the controller servlet file that is used to control the sample application. It does all its work in the doGet method. The doGet method retrieves a parameter named action from the request block. Based on the action value, the controller servlet performs the proper action. It sets the name of a response JSP, and redirects the control to the JSP.

Example: B-5 RBController.java

```
import javax.servlet.*;      // Needed for being a servlet
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*;   // Needed for all CMB beans
import java.io.*;            // Needed for direct writing to
ServletOutputStream

public class RBController extends HttpServlet {
  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }

  public synchronized void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
    doGet(request, response);
  }

  public synchronized void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {

    try {
      String responseJSP = null;
      String action = request.getParameter("action");
      if (action == null) {
        RBServers.list(request);
        responseJSP = "RBLogin.jsp";

      } else if (action.equals("ListEntities")) {
        RBListEntities.list(request);
        responseJSP = "RBListEntities.jsp";

      } else if (action.equals("SearchEntry")) {
        RBSearchEntry.list(request);
        responseJSP = "RBSearchEntry.jsp";

      } else if (action.equals("Search")) {
        request.getSession(false).removeAttribute("wpItems");
        RBSearch.search(request);
        responseJSP = "RBItemList.jsp";

      } else if (action.equals("FolderOpen")) {
        request.getSession(false).removeAttribute("wpItems");
        RBFolderOpen.open(request);
        responseJSP = "RBItemList.jsp";

      } else if (action.equals("Worklist")) {
        RBDocRouting.list(request);
        responseJSP = "RBWorklist.jsp";

      } else if (action.equals("OpenWorklist")) {
        RBDocRouting.openWorklist(request);
```

```
            responseJSP = "RBItemList.jsp";

          } else if (action.equals("SelectProcess")) {
            RBDocRouting.selectProcess(request);
            responseJSP = "RBProcessEntry.jsp";

          } else if (action.equals("StartProcess")) {
            RBDocRouting.startProcess(request);
            responseJSP = "RBItemList.jsp";

          } else if (action.equals("SelectAdvance")) {
            RBDocRouting.selectAdvance(request);
            responseJSP = "RBAdvanceEntry.jsp";

          } else if (action.equals("Advance")) {
            RBDocRouting.advance(request);
            responseJSP = "RBItemList.jsp";

          } else if (action.equals("View")) {
            RBView.view(request, response);
            return;
          } // end if (action == null)
          RequestDispatcher rd =

getServletConfig().getServletContext().getRequestDispatcher(responseJSP);
          rd.forward(request, response);
        } //end try
        //-------------------------------------------------------
        catch (CMBException exc) {
          ServletOutputStream out = response.getOutputStream();
          out.println(exc.name() + "<br>" + exc.getMessage() + "<br>");
          out.println(exc.getErrorId() + "<br>" + exc.getErrorData() + "<br>");
          exc.printStackTrace();
        } catch (Exception exc) {
          ServletOutputStream out = response.getOutputStream();
          out.println(exc.getLocalizedMessage() + "<br>" + exc.getMessage() +
"<br>");
          exc.printStackTrace();
        } catch (Throwable exc) {
          ServletOutputStream out = response.getOutputStream();
          out.println(exc.getLocalizedMessage() + "<br>" + exc.getMessage() +
"<br>");
          exc.printStackTrace();
          exc.printStackTrace();
        }
      }

}
```

List available servers: RBServer.java

Example B-6 shows the source code for RBServer.java file. It lists all the available servers of type Content Manager, OnDemand, and Federated.

Example: B-6 RBServer.java

```
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*;   // Needed for all CMB beans
import java.util.*;          // Needed for Vector

public class RBServers {
  public static void list(HttpServletRequest request)throws CMBException,
Exception {
      RBConnectionPool.destroyPool();
      Vector dsList = new Vector();

     // Setup the dsType array with appropriate values.
      String[] dsType = new String[3];
      dsType[0] = "ICM";
      dsType[1] = "OD";
      dsType[2] = "Fed";

      CMBConnection connection = new CMBConnection();
      //--------------------------------------------------
      for (int i = 0; i < dsType.length; i++) {
      // Set the connection's type to the dsType element
      connection.setDsType(dsType[i]);
      System.out.println("Listing " + dsType[i] + " servers");
      String[] sourceNames = connection.listDataSources();

        // Loop through the array and add the dsType and name to the vector
        for (int j = 0; j < sourceNames.length; j++) {
          dsList.add(dsType[i] + " " + sourceNames[j]);
        }
      }
      request.setAttribute("list", dsList);
  }
}
```

Display login screen: RBLogin.jsp

Example B-7 shows the source code for RBLogin.jsp. The generates a login form where user can enter user ID, password, and select the back-end repository server. Upon submission, it executes RBController, with a hidden action value of “ListEntities” and user ID and password values.

Example: B-7 RBLogin.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE> Login to a Server</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
</HEAD>
<body>
<table bgcolor="#f7f7f7" BORDER="4">
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="ListEntities">

<tr><td>Userid </td><td><INPUT type=text name=userid   value=""></td></tr>
<tr><td>Password </td><td><INPUT type=password name=password value="">
</td></tr>
<tr><td>Server </td><td>
<SELECT NAME=server SIZE="1" >
<%
int vSize = 0;
Vector vStat = (Vector) request.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) { %>
<OPTION value="<%=vStat.elementAt(i) %>"><%=vStat.elementAt(i) %>
<%}%>
</SELECT>
 </td></tr>
<tr><td>&nbsp</td><td><input TYPE="submit" VALUE="Logon" tabindex="7"></table>
</body>
</html>
```

Connecting to a server: RBConnectionPool.java

Example B-8 shows the source code for RBConnectionPool.java. It uses CBMConnectionPool to maintain the connection across sessions and requests to the application from the same user.

Example: B-8   RBConnectionPool.java

```
import java.net.*;
import java.beans.*;
import javax.servlet.http.*;
import com.ibm.mm.beans.*;

/**
 * Connection pooling.
 */
public class RBConnectionPool {
  private static CMBConnectionPool connectionPool;
  /**
    * Obtains a connection from the connection pool.  If a connection does
    * not exist, a new one is established.
    */
  public static synchronized CMBConnection getConnection(HttpServletRequest
request)
    throws
      IllegalAccessException, MalformedURLException, InstantiationException,
      ClassNotFoundException, PropertyVetoException, CMBException, Exception
      {

    HttpSession session = null;

    if (connectionPool == null) {
     System.out.print("Creating new bean connection Pool ... ");
      connectionPool = new CMBConnectionPool();
      connectionPool.setConnectionType(CMBBaseConstant.CMB_CONNTYPE_LOCAL);

connectionPool.setServiceConnectionType(CMBBaseConstant.CMB_CONNTYPE_LOCAL);
      connectionPool.setClientURLString(null);
      connectionPool.setCsURLString(null);
      connectionPool.setServiceClientURLString(null);
      connectionPool.setServiceCsURLString(null);

      // Parse the string server into the server name and server type
     String server = request.getParameter("server");
     int space = server.indexOf(" ");
     String dsType = server.substring(0, space);
     String dsName = server.substring(space + 1);

     if(dsType.equals("OD"))
      connectionPool.setConfigurationString("ENTITY_TYPE=TEMPLATES");

      connectionPool.setDsType(dsType);
      connectionPool.setServerName(dsName);
      // Set up a new ssession object.
     session = request.getSession(true);
     session.setAttribute("userid", request.getParameter("userid"));
     session.setAttribute("password", request.getParameter("password"));
     session.setAttribute("server", dsName);
     session.setAttribute("dsType", dsType);
    }
    else {
      session = request.getSession(false);
    }

    System.out.println("Returning CMBConnection");
    CMBConnection connection = connectionPool.getConnection((String)
session.getAttribute("userid"),
     (String) session.getAttribute("password"));

    return connection;
  }

  /**
   * Returns a connection to the connection pool.
   */
  public static synchronized void freeConnection(CMBConnection connection) {
    connectionPool.freeConnection(connection);
  }
  public static synchronized void destroyPool() throws CMBException {
    if (connectionPool != null) {
      connectionPool.destroy();
      connectionPool = null;
    }
  }
}
```

Listing entities: RBListEntities.java

Example B-9 shows the source code for RBListEntities.java. It lists either the entities if the connection is for Content Manager Version 8, or the search templates if it is for OnDemand or Federated server.

Example: B-9 RBListEntites.java

```
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*; // Needed for all CMB beans
import java.util.*; // Needed for Vector

public class RBListEntities {
  public static void list(HttpServletRequest request) throws CMBException,
Exception {
    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();
    // Create a new schema management from the connection
    CMBSchemaManagement schema = connection.getSchemaManagement();

    // Check the dsType to see if we want searchTemplates or entities
    if (connection.getDsType().equals("OD") ||
connection.getDsType().equals("Fed")) {
      // OD and FED use searchTemplates
      CMBSearchTemplate searchTemplates[] = schema.getSearchTemplates();

      // Sort the array based on what is in the getKey method of RBSortArray
      RBSortArray sort = new RBSortArray();
      sort.sort(searchTemplates, true);
      for (int i = 0; i < searchTemplates.length; ++i) {
        vector.add(
          java.net.URLEncoder.encode(searchTemplates[i].getName())
            + ";"
            + searchTemplates[i].getName());
      }
    } else {
      // ICM and most other dsTypes use entity and attributes.
      CMBEntity entities[] = schema.getEntities();

     // Sort the array based on what is in the getKey method of RBSortArray
      RBSortArray sort = new RBSortArray();
      sort.sort(entities, true);
      for (int i = 0; i < entities.length; ++i) {
        vector.add(
          java.net.URLEncoder.encode(entities[i].getName()) + ";" +
entities[i].getDisplayName());
      }
    }
    session.setAttribute("list", vector);
   RBConnectionPool.freeConnection(connection);
  }
}
```

RBListEntities.jsp

Example B-10 shows the source code for RBListEntities.jsp. It generates a table that has the links to start a search on the selected entity.

Example: B-10 RBListEntities.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
```

```
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Entities</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
</HEAD>
<body>
<h1> <%=session.getAttribute("server")%> Entities</h1>
<TABLE cellpadding="1" cellspacing="1" border="1" bgcolor="#f7f7f7">
<%if (session.getAttribute("dsType").equals("ICM")) {
// Worklists only valid for CM V8 datastores %>
<tr><td><a href="RBController?action=Worklist">*Open Worklists</TD></tr>
<%}%>
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) {
 String element = (String) vStat.elementAt(i);
 // Parse the string in the vector into entity 'Name' and 'DisplayName'
 String name = element.substring(0, element.indexOf(";"));
 String displayName = element.substring(element.indexOf(";") + 1);
%>
<tr><td><a
href="RBController?Entity=<%=name%>&action=SearchEntry"><%=displayName %>
</TD></tr>
<%}%>
</table>
</body>
</html>
```

RBSearchEntry.java

Example B-11 shows the source code for RBSearchEntry.java. It lists the attributes of entities and the search criterion of search templates.

Example: B-11 RBSearchEntry.java

```
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*; // Needed for all CMB beans
import java.util.*; // Needed for Vector
```

```
public class RBSearchEntry {
  public static void list(HttpServletRequest request) throws CMBException,
Exception {
    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

   String entityName = request.getParameter("Entity");
    session.setAttribute("Entity", entityName);

   // Create a new schema management from the connection
    CMBSchemaManagement schema = connection.getSchemaManagement();
    //---------------------------------------------
    //Check the dsType to see if we want searchTemplates or entities
    if (connection.getDsType().equals("OD") ||
connection.getDsType().equals("Fed")) {
      Vector searchCrits =
schema.getSearchTemplate(entityName).getSearchCriteria();

      for (int i = 0; i < searchCrits.size(); i++) {
        CMBSTCriterion sCrit = (CMBSTCriterion) searchCrits.elementAt(i);
        vector.add(java.net.URLEncoder.encode(sCrit.getName()) + ";" +
sCrit.getDisplayName());
      }
    } else {
      CMBEntity entity = schema.getEntity(entityName);
      CMBAttribute[] attrs = entity.getAttributes();

      RBSortArray sort = new RBSortArray();
      sort.sort(attrs, true);

      for (int i = 0; i < attrs.length; ++i) {
        vector.add(
          java.net.URLEncoder.encode(attrs[i].getName()) + ";" +
attrs[i].getDisplayName());
      }
    }
    session.setAttribute("attrlist", vector);
   RBConnectionPool.freeConnection(connection);
  }
}
```

RBSearchEntry.jsp

Example B-12 shows the source code for RBSearchEntry.jsp It displays a single drop-down box to select the Boolean operator for the search, and a table with attribute names and entry fields for values of the search.

Example: B-12 RBSearchEntry.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Enter Search Criteria</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
<title>Search</title></head>
<body>
<br><h2>Enter Attribute values for Parametric Query</h2>
<form method="POST" action="RBController" target=_top>
<input type=hidden name=action value="Search">
<TABLE bgcolor="#f7f7f7">
<TR><TH>Comparison<br>Operator</TH><td>
<SELECT NAME="_compop" SIZE=1 >
<OPTION  SELECTED value=<%=CMBBaseConstant.CMB_OP_EQUAL%>> Equals
<OPTION  value=<%=CMBBaseConstant.CMB_OP_NOT_EQUAL%>> Not Equals
<OPTION  value=<%=CMBBaseConstant.CMB_OP_GREATER%>> Greater Than
<OPTION  value=<%=CMBBaseConstant.CMB_OP_LESS%>> Less Than
<OPTION  value=<%=CMBBaseConstant.CMB_OP_GREATER_EQUAL%>> Greater or Equal
<OPTION  value=<%=CMBBaseConstant.CMB_OP_LESS_EQUAL%>> Less or Equal
<OPTION  value=<%=CMBBaseConstant.CMB_OP_LIKE%>> Like
</SELECT></td></tr>
</table>
<br>
<TABLE border="2" bgcolor="#f7f7f7">
<TR><TH>Attribute</TH><TH>Search Value</TH></TR>
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("attrlist");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) {
  String element = (String) vStat.elementAt(i);
  int parse = element.indexOf(";");
  String name = element.substring(0, parse);
  String displayName = element.substring(parse + 1);
  %>
  <tr><td><%=displayName%></TD><TD><input type=text size=45 name=<%=name%>
Value=""></TD></TR>
  <%}%>
</table>
<br>
<input type=submit value=Search></form></body>
</body>
</html>
```

RBSortArray.java

Example B-13 shows the source code for RBSortArray.java. It implements the dkSort abstract class but overrides the *getKey* method to return the value of the property used to sort on.

Example: B-13 RBSortArray.java

```
import com.ibm.mm.sdk.common.*;
import com.ibm.mm.beans.*;

public class RBSortArray implements dkSort, DKConstant, DKMessageId,
   java.io.Serializable
{
   /**
    * Recieves an object of any type, and returns a string that will be used
as the
    * key to search on.
    */
   public Object getKey(Object anObject) throws DKUsageError
   {
      if (anObject != null && (anObject instanceof CMBEntity))
         return ((CMBEntity) anObject).getDescription();
      if (anObject != null && (anObject instanceof CMBAttribute))
         return ((CMBAttribute) anObject).getDescription();
      if (anObject != null && (anObject instanceof CMBSearchTemplate))
         return ((CMBSearchTemplate) anObject).getName();
      if (anObject != null && (anObject instanceof CMBSTCriterion))
         return ((CMBSTCriterion) anObject).getName();


      throw new DKUsageError(DKMessage.getMessage(DKMSG_INVOBJTYPE) +
         this.getClass().getName() + ".getKey()",
         DKMSG_INVOBJTYPE);
   }

   /**
    * Compares the key of a generic object with another.
    * Returns a number less than zero if the first key is smaller than the
```

```
     * second, zero if they are equals,
     * and greater than zero if the first key is greater than the other.
     * This function call the above getkey() method to obtains object's key.
     * @see #getKey
     * @param first   the first  object to compare.
     * @param second  the second object to compare.
     * @return  an integer less than zero if first key is less than the second,
     *          zero if first key is equal to the second, or an integer greater
     *          than zero if first key is greater than the second.
     * @exception      DKUsageError
     *                 the given objects are not of the right type or
structure.
     */
    public int compare(Object first, Object second) throws DKUsageError
    {
       return ((String) getKey(first)).compareTo((String) getKey(second));
    }

    /**
     * Compares the key of a generic object if less than the other.
     * This function call the above getkey() method to obtains object's key.
     * Returns true if the key of the first object is less than the second.
     * @see #getKey
     * @param first    the first object to compare.
     * @param second   the second object to compare.
     * @return         true if the key of the first object is less than the
second.
     * @exception      DKUsageError
     *                 the objects are not of the right type or structure.
     */
    public boolean lessThan(Object first, Object second) throws DKUsageError
    {
       return (((String) getKey(first)).compareTo((String) getKey(second)) <
0);
    }

    /**
     * Compares the key of a generic object if greater than the other.
     * This function call the above getkey() method to obtains object's key.
     * Returns true if the key of the first object is greater than the the
second.
     * @see #getKey
     * @param first   the first object to compare.
     * @param second  the second object to compare.
     * @return        true if the key of the first object is greater than the
second.
     * @exception     DKUsageError
     *                the objects are not of the right type or structure.
     */
    public boolean greaterThan(Object first, Object second) throws DKUsageError
    {
       return (((String) getKey(first)).compareTo((String) getKey(second)) >
0);
    }

    /**
     * Compares the key of a generic object if equals to the other.
     * This function call the above getkey() method to obtains object's key.
     * Returns true if the key of the first object is less than the second.
     * @see #getKey
     * @param first   the first object to compare.
     * @param second  the second object to compare.
     * @return        true if the first object is equals to the second.
     * @exception     DKUsageError
     *                the objects are not of the right type or structure.
     */
    public boolean equals(Object first, Object second) throws DKUsageError
    {
       return ((String) getKey(first)).equals((String) getKey(second));
    }

    /**
     * Sorts an array of objects based on their keys using the one of the above
     * compare functions. The key is obtained from each object using
     * the above getKey() method.
     * It returns the same array with the objects sorted in the specified
order.
     * @see #getKey
     * @param arrayOfObjects  an array of object to be sorted.
     * @param order           the desired order, true = ascending, false =
descending.
     * @exception             DKUsageError
     *                        the objects are not of the right type or
structure.
     */
    public void sort(Object[] arrayOfObjects, boolean order) throws
DKUsageError {
       int size = arrayOfObjects.length;
       String[] keys = new String[size];
       int i;
       String val = null;
       for (i = 0; i < size; i++) {
          val = (String) getKey(arrayOfObjects[i]); // Added to handle null
values
          if (val == null) // treat nulls as null terminated string;
             keys[i] = "";
          else
             keys[i] = (String) getKey(arrayOfObjects[i]);
```

```
      }

      DKSortString.quickSort(keys, arrayOfObjects, 0, size - 1, order);
   }
}
```

RBSearch.java

Example B-14 shows the source code for RBSearch.java. It is called from the RBSearchEntry JSP with the name of the entity or search template to use, the boolean operator to use, (=, >, <, etc.) and the attribute names and values to use.

Example: B-14 RBSearch.java

```
import javax.servlet.http.*; // Needed for HttpServletRequest
import com.ibm.mm.beans.*;   // Needed for all CMB beans
import java.util.*;          // Needed for Vector

public class RBSearch {

  public static void search(HttpServletRequest request) throws CMBException,
Exception {

    HttpSession session = request.getSession(false);
    CMBConnection connection = RBConnectionPool.getConnection(request);
    String strEntity = (String) session.getAttribute("Entity");
    short opCode = (short)
Integer.valueOf(request.getParameter("_compop")).intValue();

    if (connection.getDsType().equals("ICM")) {
      // booleanOperator will take the short opCode and return a character
string for the
      // boolean operator (=,>=,<=, etc)
      String opStr = booleanOperator(opCode);
      String condString = "";

      // Loop thru the attributes of the CMBEntity and construct the
condidtional part
      // of teh XPATH query string.  It will be of the form '[@attr_name opStr
"value"]
      CMBEntity entity = connection.getSchemaManagement().getEntity(strEntity);
      CMBAttribute attrs[] = entity.getAttributes();
      for (int i = 0; i < attrs.length; i++) {
      // search values are passed into the servlet with the name of the
attribute they are for
        String dataValue =
request.getParameter(java.net.URLEncoder.encode(attrs[i].getName()));
        if (dataValue != null && dataValue.length() > 0) {
          if (condString.equals(""))
          condString = "[@" + attrs[i].getName() + opStr + "\"" + dataValue +
"\"";
         else
           condString += " AND @" + attrs[i].getName() + opStr + "\"" +
dataValue + "\"";
        }
      }

      // If an attribute value was provided to be searched on, we need to close
the bracket
      if (!condString.equals(""))
        condString += "]";
      String queryString = "/" + strEntity + condString;

      // Create a CMBQueryService (queryBean) on the connection, and prepare it
for sync
      // searching
      CMBQueryService queryBean = connection.getQueryService();
      queryBean.setAsynchSearch(false);

      queryBean.setQueryString(queryString, CMBBaseConstant.CMB_QS_TYPE_XPATH);
      System.out.println("Performing " + queryString);
      queryBean.runQueryWithCursor();

      // setDisplayEnabled affects the CMBItem objects that will be returned in
the result set
      // by having the attribute names be the display names when retrieved.
      connection.setDisplayNamesEnabled(true);

      CMBSearchResults resultBean = new CMBSearchResults();
      resultBean.setConnection(connection);
      resultBean.newResults(queryBean.getResults());
      // Sort on the values in the first defined attribute
      resultBean.sort(entity.getAttributeNames()[0], true);

      System.out.println("Results " + resultBean.getCount());
      session.setAttribute("itemList", resultBean.getItems());
      connection.setDisplayNamesEnabled(false);
      RBConnectionPool.freeConnection(connection);

      // *** OD or Fed Connectors ****
    } else if (connection.getDsType().equals("OD") ||
connection.getDsType().equals("Fed")) {
      // Create a CMBSearchTemplate (searchTemp) on the schema bean using the
passed in
      // entity name
      CMBSearchTemplate searchTemp =
connection.getSchemaManagement().getSearchTemplate(strEntity);
```

```
      // Set the search criteria of the template based on the parameters passed
in
      Vector critList = searchTemp.getSearchCriteria();
      for (int i = 0; i < critList.size(); i++) {
        CMBSTCriterion critObj = (CMBSTCriterion) critList.elementAt(i);
        String dataValue =
request.getParameter(java.net.URLEncoder.encode(critObj.getName()));
        if (dataValue != null && dataValue.length() > 0) {
          critObj.setOperator(opCode);
          critObj.setValue(dataValue);
        }
      }

      // Run the search, and put the results to the session object
      searchTemp.setAsynchSearch(false);
      searchTemp.runQuery();
      CMBSearchResults resultBean = new CMBSearchResults();
      resultBean.setConnection(connection);
      resultBean.newResults(searchTemp.getResults());

      session.setAttribute("itemList", resultBean.getItems());
      RBConnectionPool.freeConnection(connection);
    }
  }
  //-----------------------------------------------------------------------
  // Take the numeric opCode and return the string equivalent
  private static String booleanOperator(short opCode) {
    switch (opCode) {
      case CMBBaseConstant.CMB_OP_EQUAL :
        return (" = ");
      case CMBBaseConstant.CMB_OP_NOT_EQUAL :
        return (" != ");
      case CMBBaseConstant.CMB_OP_GREATER :
        return (" > ");
      case CMBBaseConstant.CMB_OP_LESS :
        return (" < ");
      case CMBBaseConstant.CMB_OP_GREATER_EQUAL :
        return (" >= ");
      case CMBBaseConstant.CMB_OP_LESS_EQUAL :
        return (" <= ");
      case CMBBaseConstant.CMB_OP_LIKE :
        return (" like ");
    }
    return (" ");

  }
}
```

RBItemList.jsp

Example B-15 shows the source code for RBItemList.jsp. It displays entity name, attribute name, and attribute values.

Example: B-15   RBItelmList.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<%@ page import="com.ibm.mm.beans.workflow.*"%>
<%!
private static String itemType(CMBItem item)
throws CMBException{
    // Return the item type of the item
    switch (item.getItemType()) {
      case CMBBaseConstant.CMB_TYPE_DOCUMENT :
        return("Document");
      case CMBBaseConstant.CMB_TYPE_FOLDER :
        return("Folder");
      case CMBBaseConstant.CMB_TYPE_ITEM :
        return("Item");
      default :
        return("Undefined");
    }
}
private static String itemAttrString(CMBItem item)
throws CMBException{
   String itemDesc = item.getEntityName()+" ";
   // Concat the attribute names and values onto the string
   for (int i = 0; i < item.getAttrCount(); i++){
      if (i>0)
        itemDesc += ", ";
      itemDesc += item.getAttrName(i) + "=" + item.getAttrValue(i);
   }
   return(itemDesc);
} %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Item List</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
```

```
</HEAD>
<body>
<h1><%=session.getAttribute("server")%> Item List</h1>
<%CMBItem items[] = (CMBItem[]) session.getAttribute("itemList");
CMBWorkPackageICM wpItems[] = (CMBWorkPackageICM[])
session.getAttribute("wpItems"); %>
<%=items.length%> Items in list.
<br><TABLE width="100%" cellpadding="1" cellspacing="1" border="1"
bgcolor="#f7f7f7">
<tr><TH>Type</th><th align="left">Entity Name,  Attribute and Attribute
Values</th><th>Actions</th></tr>
<%// For every CMBItem in the array:
for (int i = 0; i < items.length; ++i) {%>
  <TR><td><%=itemType(items[i])%></td><TD>
  <%=itemAttrString(items[i])%></td><TD>
  <form method=POST action=RBController target=_top>
  <input type=hidden name=PID
value=<%=java.net.URLEncoder.encode(items[i].getPidString())%>>
  <%if (wpItems != null) { %>
  <input type=hidden name=wpPID
value=<%=java.net.URLEncoder.encode(wpItems[i].getPidString())%>>
  <%}%>
  <table><tr><td>
  <SELECT NAME=action SIZE=1 >
  <% if (items[i].getItemType() == CMBBaseConstant.CMB_TYPE_FOLDER) { %>
    <OPTION value="FolderOpen">Folder Open
  <%} else {%>
     <OPTION SELECTED value="View">View
  <%}%>

  <% if (session.getAttribute("dsType").equals("ICM")) { %>
    <OPTION value="SelectProcess">Start Process
    <OPTION value="SelectAdvance">Advance Process
  <%}%>
  </SELECT></td><td>
  <input type=submit value='Go'>
  </td></tr></table></form></td></tr>
<%}%>
</table>
</body>
</html>
```

RBFolderOpen.java

Example B-16 shows the source code for RBFolderOpen.java. It opens Content Manager folders, and presents a member list in the RBItemList.jsp.

Example: B-16 RBFolderOpen.java

```
import javax.servlet.http.*;
import com.ibm.mm.beans.*;
import java.util.*;

public class RBFolderOpen {
  public static void open(HttpServletRequest request) throws CMBException,
Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);

    CMBDataManagement dataManagement = connection.getDataManagement();
    // Create a CMBItem (itemBean) using the passed in PID.
    // Set its connection, and hook it up to the dataBean
    CMBItem itemBean = new
CMBItem(java.net.URLDecoder.decode(request.getParameter("PID")));
    itemBean.setConnection(connection);
    //-------------------------------------------------------------
    // Process contents of folder
    // If the item is of type CMB_TYPE_FOLDER, get the vector (itemList)
    if (itemBean.getItemType() == CMBBaseConstant.CMB_TYPE_FOLDER) {
      Vector itemList = itemBean.listFolderItems();
      if (itemList.size() > 0) {
        Iterator ild_items = itemList.iterator();
        CMBItem[] items = new CMBItem[itemList.size()];

        // Iterate through the itemlist, and add to the CMBItem array
        // to be passed to the JSP.
        int i = 0;
        while (ild_items.hasNext()) {
          items[i] = (CMBItem) ild_items.next();
          i++;
        }
        session.setAttribute("itemList", items);
      }
    }
  }
}
```

RBView.java

Example B-17 shows the source code for RBView.java. It is called by RBController to display a document for viewing.

Example: B-17 RBView.java

```
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.gui.*;
import com.ibm.mm.viewer.*;
import com.ibm.mm.sdk.common.*;
import java.io.*;
import java.lang.String.*;
import java.net.URLDecoder;

public class RBView extends HttpServlet implements CMBBaseConstant {
  public static void view(HttpServletRequest request, HttpServletResponse 
response)
    throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();
    try {
      CMBConnection connection = RBConnectionPool.getConnection(request);
      HttpSession session = request.getSession(false);

      try {
        //Create a CMBDataManagement (dataBean) on the connection
        CMBDataManagement dataManagement = connection.getDataManagement();

        // Create a CMBItem (itemBean) using the passed in strPID.  Set its
        // connection, and hook it up to the dataBean
        CMBItem itemBean = new 
CMBItem(URLDecoder.decode(request.getParameter("PID")));
        itemBean.setConnection(connection);
        dataManagement.setDataObject(itemBean);

        // Retrieve the item's information
        dataManagement.retrieveItem();
        CMBDocumentServices documentServices = new CMBDocumentServices();
        documentServices.setDataManagement(dataManagement);

        view_doc(response, dataManagement, //CMBDataManagement dataManagement,
        documentServices, //CMBDocumentServices documentServices,
        itemBean, //CMBItem item,
        1, 0, true); //double scale, rotation, boolean annotate,

        RBConnectionPool.freeConnection(connection);

      } catch (CMBException ex) {
        ex.printStackTrace();
        Object data = ex.getErrorData();
        if (data != null && data instanceof Exception)
```

```
              ((Exception) data).printStackTrace();
          }
        }

        //----------------------------------------------------
        catch (java.lang.Throwable exc) {
          exc.printStackTrace(System.out);
        }
      }
      private static void view_doc( HttpServletResponse response, CMBDataManagement 
    dataManagement,
        CMBDocumentServices documentServices,
        CMBItem item,
        double scale, int rotation, boolean annotate) throws CMBException, 
    Exception {
        try {
          dataManagement.setDataObject(item);
          CMBDocument doc = documentServices.loadDocument(item);
          if (doc.getCanPaginate()) {
            CMBPage[] pages = doc.getPages();
            doc.setPreferredScale(scale);
            doc.setRotation(rotation);
            doc.setShowAnnotations(annotate);
            for (int j = 0; j < doc.getPageCount(); j++) {
              viewPage(response, pages[j]);
            }
          } else {
            // View a document.  It is converted to a browser-friendly format if 
    possible.
            String mimeType = doc.getWriteMimeType();
            try {
              response.setContentType(mimeType);
              doc.write(response.getOutputStream());
            } catch (Exception e1) {
              e1.printStackTrace();
            }
          }

          documentServices.dropDocument(doc);
        } catch (CMBNoContentException e) {
          System.out.println(e.getMessage());
        }
      } // view

      /**
       * View a page.  Scale is default 1.
       * @param page a CMBPage object representing the page to view.
       */
      public static void viewPage(HttpServletResponse response, CMBPage page) {
```

```
    String mimeType = page.getWriteMimeType();
    try {

      response.setContentType(mimeType);
      page.write(response.getOutputStream());

    } catch (Exception e1) {
      e1.printStackTrace();
    }
  } //viewPage
}
```

RBDocRouting.java

Example B-18 shows the source code for RBDocRouting.java. It allows the end user to start an item in a process, list worklists, open a worklist, view the item in a work package, and advance a work package to its next work node or to completion.

Example: B-18 RBDocRouting.java

```
import javax.servlet.http.*;
import com.ibm.mm.beans.*;
import com.ibm.mm.beans.workflow.*;
import java.util.*;

public class RBDocRouting {
  public static void list(HttpServletRequest request) throws CMBException,
Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // List the worklists names for display in the jsp.  The names need
    // to be passed to the next servlet for opening
    String worklists[] = drQService.getWorkListNames();

    for (int i = 0; i < worklists.length; ++i) {
      vector.add(worklists[i]);
    }
    session.setAttribute("list", vector);
    RBConnectionPool.freeConnection(connection);
  }
  //---------------------------------------------------------------
  public static void openWorklist(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    // Get the name of the worklist we are opening
    String worklist = request.getParameter("worklist");
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // Get the work packages that are in the worklist for display
    Collection workpackages = drQService.getWorkPackages(worklist, "");
    Iterator wpIterator = workpackages.iterator();

    int i = 0;
    CMBWorkPackageICM[] wpItems = new CMBWorkPackageICM[workpackages.size()];
    CMBItem[] items = new CMBItem[workpackages.size()];

    while (wpIterator.hasNext()) {
      wpItems[i] = (CMBWorkPackageICM) wpIterator.next();

      // Construct an Item bean for the item that is CONTAINED in this
      // workpackect, not the item that IS the workpacket.
      items[i] = new CMBItem(wpItems[i].getItemPidString());
      items[i].setConnection(connection);
      i++;
    }
    session.setAttribute("wpItems", wpItems);
    session.setAttribute("itemList", items);
    RBConnectionPool.freeConnection(connection);
  }
  //-----------------------------------------------------------------------
  public static void selectProcess(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();

    // List the worklists names for display in the jsp.  The names need
    // to be passed to the next servlet for opening
    String processNames[] = drQService.getProcessNames();

    for (int i = 0; i < processNames.length; ++i) {
      vector.add(processNames[i]);
    }
    session.setAttribute("list", vector);

    CMBDataManagement dataManagement = connection.getDataManagement();
    // Create a CMBItem (itemBean) using the passed in strPID.  Set its
    // connection, and hook it up to the dataBean
    CMBItem itemBean = new
CMBItem(java.net.URLDecoder.decode(request.getParameter("PID")));
    itemBean.setConnection(connection);
    dataManagement.setDataObject(itemBean);
    dataManagement.retrieveItem();
    session.setAttribute("item", itemBean);
    RBConnectionPool.freeConnection(connection);
  }
  //------------------------------------------------------------
  public static void startProcess(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    CMBDocRoutingDataManagementICM drManagement =
connection.getDocRoutingDataManagementICM();

    String wpItemID =
      drManagement.startProcess(
        (String) request.getParameter("process"),
        (String) java.net.URLDecoder.decode(request.getParameter("PID")),
        1,
        connection.getUserid());
    System.out.println("New Packet " + wpItemID);
    RBConnectionPool.freeConnection(connection);
  }
  //-------------------------------------------------------------------------
  public static void selectAdvance(HttpServletRequest request) throws
CMBException, Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    Vector vector = new Vector();

    // Create a CMBItem (itemBean) using the passed in PID.  Set its
connection, and hook it up to
    // the dataBean to retrieve the item information.
    CMBItem itemBean = new
CMBItem(java.net.URLDecoder.decode(request.getParameter("PID")));
   CMBDataManagement dataManagement = connection.getDataManagement();

   itemBean.setConnection(connection);
```

```
    dataManagement.setDataObject(itemBean);
    dataManagement.retrieveItem();
    session.setAttribute("item", itemBean);

    CMBDocRoutingQueryServiceICM drQService =
connection.getDocRoutingQueryServiceICM();
    // Create a workpackage using the passed in PID
    CMBWorkPackageICM wp =

drQService.getWorkPackage(java.net.URLDecoder.decode(request.getParameter("wpPI
D")), true);

    // Create a CMBProcessICM form the name in ithe workpackage and the routes
it has
    CMBProcessICM process = drQService.getProcess(wp.getProcessName());
    Collection route = process.getRoute();
    Iterator routeIterator = route.iterator();

    while (routeIterator.hasNext()) {
      CMBRouteListEntryICM rlEntry = (CMBRouteListEntryICM)
routeIterator.next();
      if (rlEntry.getFrom().equals(wp.getWorkNodeName())) {
      // For the node in this process, put the names of the routes in the
vector.
        vector.add(rlEntry.getSelection());
      }
    }
    session.setAttribute("list", vector);
    RBConnectionPool.freeConnection(connection);
  }

//-----------------------------------------------------------------------------
--
  public static void advance(HttpServletRequest request) throws CMBException,
Throwable {

    CMBConnection connection = RBConnectionPool.getConnection(request);
    HttpSession session = request.getSession(false);
    CMBDocRoutingDataManagementICM drManagement =
connection.getDocRoutingDataManagementICM();

    String wpItemID =
      drManagement.continueProcess(
        (String) java.net.URLDecoder.decode(request.getParameter("wpPID")),
        (String) request.getParameter("selection"),
        connection.getUserid());
    RBConnectionPool.freeConnection(connection);
  }
```

```
}
```

RBProcessEntry.jsp

Example B-19 shows the source code for RBProcessEntry.jsp. It displays the attributes of the item the same as RBItemList.jsp does, with a selection drop-down box with the process names.

Example: B-19 RBProcessEntry.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<%!
private static String itemType(CMBItem item)
throws CMBException{
    // Return the item type of the item
    switch (item.getItemType()) {
      case CMBBaseConstant.CMB_TYPE_DOCUMENT :
        return("Document");
      case CMBBaseConstant.CMB_TYPE_FOLDER :
        return("Folder");
      case CMBBaseConstant.CMB_TYPE_ITEM :
        return("Item");
      default :
        return("Undefined");
    }
}
private static String itemAttrString(CMBItem item)
throws CMBException{
   String itemDesc = "";
   // Concat the attribute names and values onto the string
   for (int i = 0; i < item.getAttrCount(); i++){
       if (i>0)
         itemDesc += ", ";
       itemDesc += item.getAttrName(i) + "=" + item.getAttrValue(i);
   }
   return(itemDesc);
} %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Worklists</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
```

```
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
</HEAD>
<body>
<h1><%=session.getAttribute("server")%> Select Process</h1>
<TABLE width="100%" cellpadding="1" cellspacing="1" border="1"
bgcolor="#f7f7f7">
<tr><TH>Type</th><th align="left">Entity Name,  Attribute and Attribute
Values</th></tr>
<%CMBItem item = (CMBItem) session.getAttribute("item");%>
<TR><td><%=itemType(item)%></td><TD>
<%=item.getEntityName()%>&nbsp<%=itemAttrString(item)%>
  </td></tr></table>
  <br>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="StartProcess">
<input type=hidden name=PID
value=<%=java.net.URLEncoder.encode(item.getPidString())%>>
<SELECT NAME=process SIZE=5 >
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) { %>
<OPTION value="<%=vStat.elementAt(i)%>"><%=vStat.elementAt(i) %>
<%}
%>
</SELECT><br><br><input type=submit value='Select'></form>
</body>
</html>
```

RBWorklist.jsp

Example B-20 shows the source code for RBWorklist.jsp. It puts the name of the worklists from the vector to the selection box and is called by the RBController servlet.

Example: B-20 RBWorklist.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
```

```
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Worklists</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
</HEAD>
<body>
<h1><%=session.getAttribute("server")%> Worklists</h1>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="OpenWorklist">
<SELECT NAME=worklist SIZE=5 >
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
}
for (int i = 0; i < vSize; i++) { %>
<OPTION value="<%=vStat.elementAt(i)%>"><%=vStat.elementAt(i) %>
<%}%>
</SELECT><br><br><input type=submit value='Select'></form>
</body>
</html>
```

RBAdvanceEntry.jsp

Example B-21 shows the source code for RBAdvanceEntry.jsp. It displays the attribute information of the item being advanced, and a list of the advancing options (such as continue, accept, reject, and escalate) from the list vector.

Example: B-21   RBAdvanceEntry.jsp

```
<%@ page import="com.ibm.mm.beans.*" %>
<%@ page import="java.util.*" %>
<%!
private static String itemType(CMBItem item)
throws CMBException{
    // Return the item type of the item
    switch (item.getItemType()) {
      case CMBBaseConstant.CMB_TYPE_DOCUMENT :
        return("Document");
      case CMBBaseConstant.CMB_TYPE_FOLDER :
        return("Folder");
      case CMBBaseConstant.CMB_TYPE_ITEM :
        return("Item");
      default :
        return("Undefined");
    }
}
private static String itemAttrString(CMBItem item)
throws CMBException{
   String itemDesc = "";
   // Concat the attribute names and values onto the string
   for (int i = 0; i < item.getAttrCount(); i++){
      if (i>0)
        itemDesc += ", ";
      itemDesc += item.getAttrName(i) + "=" + item.getAttrValue(i);
   }
   return(itemDesc);
} %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<HEAD>
<META http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<META name="GENERATOR" content="IBM WebSphere Studio">
<META http-equiv="Content-Style-Type" content="text/css">
<LINK href="../theme/Master.css" rel="stylesheet" type="text/css">
<TITLE>Worklists</TITLE>
<META name="Cache-Control" content="no-cache">
<META name="Pragma" content="no-cache">
<META name="Expires" content="-1">
<META content="no-cache" http-equiv="Cache-Control">
<META content="no-cache" http-equiv="Pragma">
<META content="-1" http-equiv="Expires">
</HEAD>
<body>
<h1><%=session.getAttribute("server")%> Select Process</h1>
<TABLE width="100%" cellpadding="1" cellspacing="1" border="1"
bgcolor="#f7f7f7">
<tr><TH>Type</th><th align="left">Entity Name,  Attribute and Attribute
Values</th></tr>
<%CMBItem item = (CMBItem) session.getAttribute("item");%>
<TR><td><%=itemType(item)%></td><TD>
<%=item.getEntityName()%>&nbsp<%=itemAttrString(item)%>
  </td></tr></table>
  <br>
<form method=POST action=RBController target=_top>
<input type=hidden name=action value="Advance">
<input type=hidden name=wpPID value=<%=request.getParameter("wpPID")%>>
<%
int vSize = 0;
Vector vStat = (Vector) session.getAttribute("list");
if (!(vStat == null)) {
  vSize = vStat.size();
```

```
}
for (int i = 0; i < vSize; i++) { %>
<input type=radio name=iType value="<%=vStat.elementAt(i)%>"
<%if (i==0){%> checked <%}%>><%=vStat.elementAt(i) %>
<%}
%>
<br><br><input type=submit value='Select'></form>
</body>
</html>
```

B.3 Building permit application folder use case

In this section, we include all the source code referenced in the building permit application folder use case implementation.

A list of source code provided here includes:

- "RBDB2Model.java" on page 581
- "ICMConnectionPool.java" on page 586

RBDB2Model.java

Example B-22 contains the source code for RBDB2Model.java that is used when implementing the building permit application folder use case.

Example: B-22 RBDB2Model.java

```
package itso.rb.intranet.model;

import com.ibm.mm.sdk.server.*;
import com.ibm.mm.sdk.common.*;

import java.util.Vector;
import java.sql.Date;
import itso.rb.intranet.bean.*;

public class RBDB2Model {

   DKDatastoreDB2 dsDB2=null;
   dkQuery pQry = null;

   String dsName, userName, password, connectString;

   public void setConnectParams(String pdsName, String puserName,
            String ppassword, String pconnectString){
      dsName = pdsName;
      userName = puserName;
      password = ppassword;
      connectString = pconnectString;
   }
   public void connect() throws DKException, Exception {
      if(dsDB2 == null) {
         dsDB2 = new DKDatastoreDB2();
      }
      dsDB2.connect(dsName, userName, password, connectString);

      System.out.println("Connected to: " + dsDB2.datastoreName() );
   }
   public void disconnect() throws DKException, Exception {

      if(dsDB2 != null && dsDB2.isConnected()) {
         dsDB2.disconnect();
      }
   }
   /**
    * find Building Permit owner by BPNumber. Return a vector
    * with 2 String objects
    * - 1. String: Owner SSN
    * - 2. String: SSN + Owner Name
    *
    * @param pBuildingPermit
    * @return java.util.Vector
    */
   public Vector findOwnerOfBuildingPermit(String pBuildingPermit) {
      // In order the have this code sample simple, the display format
      // is done by this bean
      Vector result = new Vector();
      try {

         // Check connection
         if(dsDB2 == null || !dsDB2.isConnected() ) {
            this.connect();
         }

         // Define query
         String cmd = "select a.SSN, a.firstName, a.lastname " +
                 " from itso.owner a, itso.owner_history b, 
itso.building_permit c " +
                 " where b.SSN = a.SSN and b.PROPERTY_NBR = c.PROPERTY_NBR and 
c.id = '" +
             pBuildingPermit + "'";
         pQry = dsDB2.createQuery(cmd,
                  DKConstant.DK_CM_SQL_QL_TYPE,
                  null);

         pQry.execute(null);
         DKResults pResults = (DKResults)pQry.result();
         dkIterator iter = pResults.createIterator();

         DKDDO    item = null;
         Short    sVal;
         short    itemType = 0;
         String   dataName;

         while(iter.more()) {

            item = (DKDDO) iter.next();

            if (item != null) {

               String SSN = (String) item.getDataByName("SSN");
               String DisplaySSN = SSN + " - " +
item.getDataByName("FIRSTNAME") + " " +
                                     item.getDataByName("LASTNAME");

               result.add(SSN);
               result.add(DisplaySSN);

            }//if
         } //while

         dsDB2.commit();

      } catch (DKException e) {
         System.out.println("Error Code: " + e.errorCode() + "\n" +
e.getMessage());
      } catch (Exception e1) {
         System.out.println(e1.getMessage());
      }

      return result;
   }
   /**
    * retrieves unprocessed building permit applications records
    *
    * @return java.util.Vector
    */
   public Vector getBusinessPermitByCertificate(String pCertificate) {

      Vector resultTable = new Vector();
      DKResults pResults = null;

      try {
         // Check connection
         if(dsDB2 == null || !dsDB2.isConnected() ) {
            this.connect();
         }
         // Define query
         String cmd = "SELECT * FROM itso.building_permit WHERE certificate =
'"
             + pCertificate + "'";
         pQry = dsDB2.createQuery(cmd,
                      DKConstant.DK_CM_SQL_QL_TYPE,
                      null);
         pQry.execute(null);
         System.out.println("Number of results: " + pQry.numberOfResults());
         pResults = (DKResults)pQry.result();
         dkIterator iter = pResults.createIterator();

         DKDDO    item = null;
         Short    sVal;
         short    itemType = 0;
         String   dataName;

         while(iter.more()) {
            item = (DKDDO) iter.next();
            if (item != null) {

               RBBuildingPermitBean lbean = new RBBuildingPermitBean();

               lbean.setId( (String) item.getDataByName("ID"));
               System.out.println( (String) item.getDataByName("ID") );
               System.out.println( (String) item.getDataByName("ID") );
               lbean.setProperty_nbr(
                    (String) item.getDataByName("PROPERTY_NBR"));
               lbean.setElectrical(
                    (String) item.getDataByName("ELECTRICAL"));
               lbean.setEInspectionDate(
                    (Date) item.getDataByName("EINSPECTIONDATE"));
               lbean.setMechanical(
                    (String) item.getDataByName("MECHANICAL"));
               lbean.setMInspectionDate(
                    (Date) item.getDataByName("MINSPECTIONDATE"));
               lbean.setPlumbing(
                    (String) item.getDataByName("PLUMBING"));
               lbean.setPInspectionDate(
                    (Date) item.getDataByName("PINSPECTIONDATE"));
               lbean.setPlumbing(
                    (String) item.getDataByName("CERTIFICATE"));
               lbean.setPInspectionDate(
                    (Date) item.getDataByName("CINSPECTIONDATE"));
               lbean.setFees( (Double) item.getDataByName("FEES"));
               lbean.setIssuedTo(
                    (String) item.getDataByName("ISSUED_TO"));
```

```
                  resultTable.add(lbean);
               }//if
            } //while
             dsDB2.commit();
         } catch (DKException e) {
            System.out.println("Error Code: " + e.errorCode() + "\n" +
   e.getMessage());
         } catch (Exception e1) {
            System.out.println(e1.getMessage());
         }
         return resultTable;
      }

      /**
       * @param BPNumber
       * @param Status
       */
      public void updatePermitStatus(String pId, String pStatus) {
         try {

            // Check connection
            if(dsDB2 == null || !dsDB2.isConnected() ) {
               this.connect();
            }
            String cmd = "SELECT * FROM itso.building_permit where id = '" +
   pId.trim() +"'";
            System.out.println(cmd);
            pQry = dsDB2.createQuery(cmd,
                       DKConstant.DK_CM_SQL_QL_TYPE,
                       null);
            pQry.execute(null);
            DKResults pResults = (DKResults) pQry.result();
            dkIterator iter = pResults.createIterator();

            DKDDO    item = null;
            while(iter.more()) {
                item = (DKDDO) iter.next();
                // Update item
               item.setData(
                  item.dataId(DKConstant.DK_CM_NAMESPACE_ATTR,"CERTIFICATE"),
                  pStatus);
               // Saves changes to persistence store
               item.update();
            }
            dsDB2.commit();

         } catch (DKException e) {
            System.out.println("Error Code: " + e.errorCode() + "\n" +
               e.getMessage());
```

```
          try{
           dsDB2.rollback();
          } catch (Exception e1) {
          }
      } catch (Exception e1) {
         System.out.println(e1.getMessage());
      }
   }
}
```

ICMConnectionPool.java

Example B-23 contains the source code for ICMConnectionPool.java that is used when implementing the building permit application folder use case.

Example: B-23 ICMConnectionPool.java

```
import com.ibm.mm.sdk.common.*;
import com.ibm.mm.sdk.server.*;
import java.io.*;
import java.lang.reflect.*;
import javax.servlet.http.*;
/**
 * ICM connection pooling.
 */
public class ICMConnectionPool implements DKConstantICM {

  private static DKDatastorePool connectionPool;
  /**
   * Obtains a connection from the connection pool.  If a connection does
   * not exist, a new one is established.
   */
  public static synchronized DKDatastoreICM
       getConnection(String ICMServer, String ICMUser, String ICMPasswd)
    throws IllegalAccessException, InstantiationException, DKException,
Exception {

    // Create a String (fullClassName) and set it to the
   // ICM datastore class name
    String fullClassName = "com.ibm.mm.sdk.server.DKDatastoreICM";
    String server = ICMServer;
    String userid = ICMUser;
    String password = ICMPasswd;

    if (connectionPool == null) {
      System.out.println("Setting new Connection Pool");

      // Set connectionPool to a new DKDatastorePool using fullClassName
      connectionPool = new DKDatastorePool(fullClassName);
      connectionPool.setDatastoreName(server);
      connectionPool.setConnectString("");
      connectionPool.setMaxPoolSize(10);
      connectionPool.setMinPoolSize(3);

     System.out.println("userid: " + userid + " password: " + password);
      connectionPool.initConnections(userid, password, 5);
      connectionPool.setTimeOut(10);
    }
    DKDatastoreICM connection =
      (DKDatastoreICM) connectionPool.getConnection(userid, password);
    System.out.println("Connection to Datastore " + server + " returned");

    return connection;
  }

  /**
   * Returns a connection to the connection pool.
   */
  public static synchronized void returnConnection(dkDatastore connection)
    throws Exception {
    connectionPool.returnConnection(connection);
  }
  public static synchronized void clearConnections() {
    try {
      connectionPool.clearConnections();
    } catch (Exception exc) {
      exc.printStackTrace();
    }
  }
   public static synchronized void destroyConnections() {
        try {
        if (connectionPool != null) {
          connectionPool.destroy();
          connectionPool = null;
        }
        } catch (Exception exc) {
          exc.printStackTrace();
        }
      }
}
```

C

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/SG246338
```

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, SG246338.

Using the Web material

The additional Web material that accompanies this redbook includes the following files:

File name	*Description*
sg246338.zip	Zipped code samples and setup data files

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	200 MB minimum
Operating System:	Windows
Processor:	Pentium® IV or higher
Memory:	512 MB

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 592. Note that some of the documents referenced here may be available in softcopy only:

- *IBM DB2 Content Manager Implementation and Migration Cookbook*, SG24-7051
- *IBM DB2 Content Manager OnDemand Guide*, SG24-6915
- *WebSphere Studio Application Developer Version 5 Programming Guide,* SG24-6957

Other publications

These publications are also relevant as further information sources:

- *IBM Content Manager for Multiplatforms/IBM Information Integrator for Content, Workstation Application Programming Guide,* C2713471
- *IBM Content Manager Client for Windows - Client for Windows Programming Reference Version 8.2*, SC27-1337
- *IBM Content Manager OnDemand for Multiplatforms: Web Enablement Kit Implementation Guide Version 7.1*, SC27-1000
- *IBM DB2 Content Manager OnDemand for Multiplatforms: Introduction and Planning Guide*, GC27-0839
- *Information Integrator for Content Version 8 Applet Viewer Sample*
- *Developing a DB2 Content Manager Version 8.2 Web Solution*

Online resources

These Web sites and URLs are also relevant as further information sources:

- IBM Redbooks:

  http://www.redbooks.ibm.com/
- OnDemand fixes - PTFs:

  ftp://service.software.ibm.com/software/ondemand/fixes/
- Object Management Group documentation:

  http://www.omg.org
- To obtain a trial version of WebSphere Studio Application Developer:

  http://www.ibm.com/software/awdtools/studioappdev

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

IBM

Redbooks

Implementing Web Applications with CM Information Integrator for Content and ODWEK

(1.0" spine)
0.875"<->1.498"
460 <-> 788 pages

Implementing Web Applications with CM Information Integrator for Content and OnDemand Web Enablement Kit

Introducing Web application implementation basics

Covering Information Integrator for Content and ODWEK

Real-world case study with sample codes

In this IBM Redbook, we deal with implementing Web applications using IBM DB2 Content Manager Information Integrator for Content Version 8 and IBM DB2 Content Manager OnDemand Web Enablement Kit. It is aimed at designers and developers of Content Manager systems.

In Part 1, we provide a brief introduction to Content Manager, OnDemand, Web application basics, and the case study we use throughout the entire redbook when showing and demonstrating how to develop Web applications.

In Part 2, we work specifically with application development with Information Integrator for Content. We provide a brief programming overview and show you how to get a quick start with developing Web applications using the Java OO APIs. Using non-visual Java beans, we show you how to build a generic Content Manager application. Working with the case study, we also show you how to build a customized Content Manager application. Lastly, we show you how you can add text search and document rendering in your application.

In Part 3, we work with ODWEK. We provide a brief overview of Web enabling OnDemand, installing and configuring of ODWEK, and developing a Web application using ODWEK.

There are many sample codes provided along with this redbook. They provide the basic concept and code for developing Content Manager and OnDemand Web applications.

SG24-6338-00 ISBN 0738490393