

# **Documentum Web Development Kit™ Reviewer's Guide**

**Version 5.1  
December 2002**

Copyright 1999-2002 by Documentum, Inc. 6801 Koll Center Parkway Pleasanton, CA 94566

DOCUMENTUM, NOW YOU KNOW, UNITING THE WORLD THROUGH CONTENT, GMPHARMA, GXPHARMA, GDPHARMA, GSPHARMA and the Corporate Logo are trademarks or registered trademarks of Documentum, Inc. in the United States and throughout the world. All other company and product names are used for identification purposes only and may be trademarks of their respective owners. Copyright © 1994-2002 Documentum, Inc. All Rights Reserved Printed in the USA.

# Table of Contents

---

<b>Preface</b> .....	ix
<b>Chapter 1 Introduction</b> .....	1– 1
Welcome .....	1– 1
About Documentum .....	1– 1
The Documentum ECM Platform .....	1– 2
Why is Enterprise Content Management Important? .....	1– 2
Documentum ECM – Managing Ideas from Concept to Content .....	1– 3
Enterprise Integrations and Development Tools .....	1– 4
Development of Content Applications .....	1– 4
Introduction to WDK 5 .....	1– 5
Getting Started .....	1– 5
Contact Information .....	1– 5
Where to Go from Here .....	1– 6
<b>Chapter 2 Additional Development Options</b> .....	2– 1
DFC .....	2– 1
Business Objects Framework .....	2– 2
BOF Overview .....	2– 2
RightSite, Docbasic, and DQL .....	2– 3
Moving forward with WDK .....	2– 3
Model-View-Controller Paradigm .....	2– 4
<b>Chapter 3 Standard Documentum Web Applications</b> .....	3– 1
Documentum Webtop .....	3– 1
Documentum Web Publisher .....	3– 1
Documentum Administrator (DA) .....	3– 2
Documentum Content Services for Portlets .....	3– 2
<b>Chapter 4 WDK 5 Technology Overview</b> .....	4– 1
WDK 5 Architecture .....	4– 1
Service framework .....	4– 2
Presentation model .....	4– 3
Component model .....	4– 3
Application model .....	4– 3
Portals .....	4– 4
Programming model .....	4– 4
Compatibility and Migration .....	4– 4
Documentum Features Exposed in WDK 5 .....	4– 5

	XML .....	4– 5
	Workflow .....	4– 5
	Component Manifest .....	4– 6
<b>Chapter 5</b>	<b>WDK Technology in Depth</b> .....	5– 1
	Introduction .....	5– 1
	Java, J2EE, and tag libraries .....	5– 2
	J2EE .....	5– 3
	JavaServer Pages and Tag Libraries .....	5– 3
	JavaServer Faces and JSR 127 .....	5– 4
	WDK 5 Application Example .....	5– 5
	Component Model .....	5– 6
	Controls .....	5– 7
	Forms .....	5– 7
	Services .....	5– 8
	Anatomy of A WDK 5 Application .....	5–10
	Source File Types .....	5–10
	The WDK Container .....	5–11
	Control Manifest .....	5–12
<b>Chapter 6</b>	<b>Preparing for installation</b> .....	6– 1
	Preparing clients .....	6– 1
	Preparing the application server host for installation .....	6– 2
	Uninstalling Documentum applications and DFC .....	6– 3
	Uninstalling WDK 5 beta or cleaning up a failed WDK 5 installation .....	6– 3
	Installing the Java application server .....	6– 5
	Backing up customizations .....	6– 5
	Preparing Docbases (4.x Docbases only) .....	6– 5
	Preparing information for the installer .....	6– 5
	Preparing Docbroker information .....	6– 6
	Setting Unix environment variables .....	6– 6
	DFC installation directories .....	6– 6
	(WebLogic only) Creating a domain for the application .....	6– 8
	Installing WDK for customization of Webtop or other Web application .....	6– 8
<b>Chapter 7</b>	<b>Installing WDK</b> .....	7– 1
	Installing on Windows .....	7– 1
	Installing on Solaris .....	7– 3
	Testing your deployment .....	7– 4
	Uninstalling WDK .....	7– 6
<b>Chapter 8</b>	<b>Building a WDK 5 Application</b> .....	8– 1
	Building the Application Infrastructure .....	8– 1
	Locating Components and Resources .....	8– 1
	Assembling the Components to Build an Application .....	8– 2
	Building a Custom Component Library .....	8– 2
	Preserving your Changes .....	8– 2
	Common Configurations and Customizations .....	8– 3
	Component Configuration .....	8– 3
	XML Configuration .....	8– 3

	JavaServer Pages .....	8- 6
	Component Customization.....	8- 6
	Customization Tools.....	8- 6
	Using DFC .....	8- 6
	Using Standard Java .....	8- 7
<b>Chapter 9</b>	<b>Preparing the WDK Environment</b> .....	9- 1
	Environment for the WDK Tutorials.....	9- 1
	Testing the Tomcat/JDK Setup.....	9- 1
	Testing the WDK Setup.....	9- 2
<b>Chapter 10</b>	<b>Configuration Tutorials</b> .....	10- 1
	Tutorial 1: Creating a Start Page for the Application.....	10- 1
	Tutorial 2: Configuring the Display of Attributes .....	10- 2
	Overview of the Configuration .....	10- 4
	Creating the Custom Layer .....	10- 4
	Configuring the XML Definition File for Attributes.....	10- 5
	Testing Access to the Custom Layer .....	10- 5
	Removing Properties from Display.....	10- 6
	Adding Existing Docbase Attributes to the Main Info Page .....	10- 7
	Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages.....	10- 8
	Overview of the Customization.....	10- 9
	Creating a Custom Object Type .....	10- 9
	Creating the Custom Layer Files .....	10-11
	Modifying the Info Page.....	10-11
	Modifying the Checkin Page .....	10-13
	Tutorial 4: Adding a Logout Link .....	10-15
	Creating the Custom Layer Files .....	10-16
	Modifying the XML Definition File.....	10-17
	Modifying the Drilldown JSP .....	10-17
	Modifying custom_drilldown_body.jsp .....	10-18
	Creating a Custom Resource File .....	10-18
	Testing the Logout Link .....	10-19
<b>Chapter 11</b>	<b>Customization Tutorial</b> .....	11- 1
	Setting up the WDK Project in Sun ONE Studio .....	11- 1
	Creating a Project for WDK .....	11- 1
	Mounting the WDK Directories .....	11- 1
	Stopping the Internal Tomcat Server .....	11- 2
	Tutorial 5: Converting from a Dialog to a Wizard Display .....	11- 2
	Creating the Custom Layer Files .....	11- 4
	Modifying the Attributes Component Definition .....	11- 4
	Modifying the JSP Pages .....	11- 5
	Extending the Attribute Class.....	11- 6
	Testing the Customization.....	11- 7
<b>Chapter 12</b>	<b>Troubleshooting</b> .....	12- 1
	Did you precompile your Java class? .....	12- 1
	Did you refresh the Configuration Service?.....	12- 1
	Did you remove generated files? .....	12- 1
	Did you clear the browser cache? .....	12- 2

## Table of Contents

---

Did you check the name and location of the XML definition file?.....	12- 2
---	-------

# List of Figures

---

Figure 2–1.	DFC Provides DBO Framework .....	2– 2
Figure 2–2.	Model, View, Controller .....	2– 4
Figure 4–1.	WDK Framework .....	4– 2
Figure 5–1.	Documentum Java Platform .....	5– 3
Figure 10–1.	Post-Login Drilldown Page.....	10– 2
Figure 10–2.	Cabinet or Folder Contents Page.....	10– 3
Figure 10–3.	Standard dm_document Properties .....	10– 3
Figure 10–4.	Added Properties .....	10– 8
Figure 10–5.	Info Page with Custom Attributes .....	10– 9
Figure 10–6.	Checkin Page with Custom Attributes.....	10–15
Figure 10–7.	Drilldown Page with Logout Link .....	10–16
Figure 11–1.	Customized Info Page.....	11– 3



# List of Tables

---

Table 4-1.	Service Framework .....	4- 2
Table 4-2.	Other WDK Services .....	4- 3
Table 4-3.	Core Library Services and Navigation Components .....	4- 6
Table 4-4.	Administration Components.....	4- 6
Table 4-5.	ACL Management Components .....	4- 6
Table 4-6.	Content Transfer Components and Applets .....	4- 7
Table 4-7.	Lifecycle Management Components .....	4- 7
Table 4-8.	Properties Components and Functionality .....	4- 7
Table 4-9.	Rendition Component Functionality .....	4- 7
Table 4-10.	Search Components.....	4- 7
Table 4-11.	Virtual Document Manager Components .....	4- 7
Table 4-12.	Workflow Components .....	4- 8
Table 4-13.	Utility Components and Functionality .....	4- 8
Table 4-14.	XML Support.....	4- 8
Table 4-15.	Component Containers.....	4- 8
Table 4-16.	Component Locators .....	4- 8
Table 4-17.	Services .....	4- 9
Table 4-18.	Developer and Administrator Features .....	4- 9
Table 5-1.	Documentum Java-based Products .....	5- 2
Table 5-2.	Three Tiers of a Web Application .....	5- 4
Table 5-3.	WDK Source File Types.....	5-11
Table 5-4.	General Controls.....	5-12
Table 5-5.	Content repository (Docbase) Enabled Controls .....	5-12
Table 8-1.	Directory Structure of the Custom Layer .....	8- 3
Table 8-2.	Elements of an XML Component Definition File .....	8- 4
Table 8-3.	Component Methods .....	8- 5
Table 10-1.	SOP Application Attributes.....	10-10



## Purpose of the Manual

**Documentum Web Development Kit™ Reviewer's Guide** provides overview information and tutorials for building applications using WDK 5.1. This document contains information for all of the certified platforms. Refer to the product release notes for the list of platforms certified in that particular release.

## Intended Audience

This manual is primarily intended for publishers, managers, and developers who are evaluating WDK 5.1. It assumes some knowledge of the following:

- Java application servers
- Java classes and JavaServer Pages
- Java IDEs
- XML (elements, well-formedness)
- Documentum core concepts: Docbases, DocBrokers, document management. For a technical overview of these concepts, see the PDF manual **Documentum eContent Server™ Fundamentals**.

## Revision History

The following changes have been made to this document:

### Revision History

<b>Revision Date</b>	<b>Description</b>
December 2002	Minor revisions
September 2002	Initial document release

## Organization of the Manual

This manual contains the following chapters. The table below lists the information in each chapter.

### *Manual Organization*

<b><i>Revision Date</i></b>	<b><i>Description</i></b>
Chapter 1 – Introduction	Offers a base understanding of the product in order to provide assistance in creating an evaluation plan.
Chapter 2 – Additional Development Options	Reviews all of the additional development options within Documentum.
Chapter 3 – Standard Documentum Web Applications	Outlines the standard Documentum products that will be built using the WDK 5 infrastructure.
Chapter 4 – WDK 5 Technology Overview	Provides an overview of the underlying technology used in the design and development of WDK 5.
Chapter 5 – WDK 5 Technology in Depth	Explains the technology used within WDK 5 in greater detail.
Chapter 6 – Preparing for Installation	Describes how to prepare your development environment for WDK installation.
Chapter 7 – Installing WDK	The WDK installation procedure.
Chapter 8 – Building A WDK 5 Application	Describes the process for building a WDK application.
Chapter 9 – Preparing the WDK Environment	Outlines the requirements for the tutorials.
Chapter 10 – Configuration Tutorials	Provides tutorials that extend and modify configuration through XML definition files and JSP pages.
Chapter 11 – Customization Tutorial	Provides a tutorial that customizes the application by extending a Java class.
Chapter 12 – Troubleshooting	Offers tips for troubleshooting configuration and customization issues.

## *Conventions*

This manual uses the following conventions:

*Conventions*

<b>Convention</b>	<b>Description</b>
<b><i>Italics</i></b>	Represents a variable name for which you must provide a value, or a defined term.  <b>Note:</b> In this manual, <b><i>wdk5</i></b> is used in pathnames to represent the virtual directory that you assigned when you installed WDK. It is a subdirectory under <b><i>tomcat_home_directory</i></b> /webapps.
monospaced	Represents code samples, commands, user input, and computer output.
[square brackets]	Indicates an optional argument in a command.
{curly braces}	Indicates an optional argument that can be repeated more than once, or a Java code implementation.

*Bug Lists and Documentation Online*

Customers with a software support agreement can access the support area of [www.documentum.com](http://www.documentum.com) to download product documentation. After a product has been commercially released, you can view lists of fixed bugs on the support web page.

To access the support area, follow the links to request a user name and password. Documentum responds to access requests within two business days. Once you have a user name and password, you can access the support area without delay.

*Fixed Bug Lists*

Two weeks after the release, we post a list of customer-reported bugs that have been fixed in that release to the support area of [www.documentum.com](http://www.documentum.com).

*Product Documentation*

When a product is released, we post product documentation to the support area of [www.documentum.com](http://www.documentum.com).

## *Purchasing Manuals*

You can order any of our documentation in the form of bound manuals. We print and stock widely-used manuals, usually several months after we publish the online version. We print other documentation to order. To place an order or to ask about prices, call the documentation order line at (925) 600-6666. You can pay with a purchase order, check, or credit card. We do not sell bound manuals through [www.documentum.com](http://www.documentum.com).

## *Introduction*

This chapter will provide an introduction to WDK 5 and its associated Documentum technology. This will assist you in the preparation of your evaluation. This chapter will cover the following topics:

- Welcome
- About Documentum
- Introduction to WDK 5
- Getting Started
- Where to Go from Here

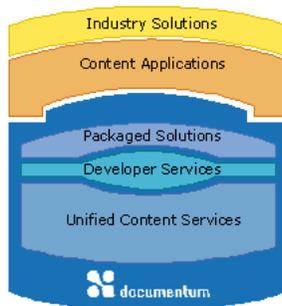
## *Welcome*

Thank you for taking the time to review Documentum Web Development Kit 5.1. This document will provide you with an overview of the product, the technology upon which it is founded and walk you through a tutorial familiarizing you with the process of building a WDK-based application.

## *About Documentum*

Documentum is the industry's leading enterprise content management provider, automating the production, exchange, and personalization of all types of content, making it easier for the Global 2000 to gain competitive advantage by connecting employees, business partners and customers, worldwide. Built on an Internet-scale, XML-enabled and standards-compliant platform, Documentum products manage Web content, power portals, enable collaborative commerce, and solve regulatory content challenges. Over 300 partners across all major industries, including high tech, pharmaceutical, healthcare, consulting services, government, manufacturing, financial services, automotive, retail, and consumer goods, build and implement specialized applications using Documentum's content management infrastructure.

## The Documentum ECM Platform



The Documentum Enterprise Content Management (ECM) platform is the foundation on which content applications and solutions are built. It is the basis for every information-based project you might be interested in – from managing business documents to publishing content on global Web Sites. The Documentum ECM platform provides the core functionality common to each of these deployments.

**Unified content services**, our core functionality, refers to not only our robust repository and library services (including security, workflow, version control, and so on) but also our more advanced capabilities, such as content intelligence, and rich media management.

For Documentum developers, this functionality is exposed as a common set of APIs and standards-based components, or **developer services**. Developers can easily leverage this to build customized solutions. A wide variety of **content applications** are built on Documentum and are designed to meet a wide range of business needs.

Documentum packages its broad set of capabilities into a set of packaged solutions, or Editions, that address specific content management areas. These **packaged solutions** provide a convenient way to determine which products you need to meet a particular objective, such as publishing content to the Web, managing rich media, controlling compliance documentation, or enabling enterprise collaboration. To address the business needs in specific industries, Documentum provides **industry solutions**, which combine Documentum products, partner products, and focused professional services.

For more information, visit Documentum on the Web at [www.documentum.com](http://www.documentum.com).

## Why is Enterprise Content Management Important?

People are undoubtedly the most valuable resource a company has. But after its people, and far exceeding tangibles like physical inventory and equipment, the value of your enterprise exists in its information. The many forms that information takes – product specifications, marketing collateral, Web pages, customer service data, supplier contracts, and so on – is what we call enterprise content. How that information is created, stored, and reused is the job of enterprise content management.

This information, or enterprise content, represents an enormous investment in time and money – easily in hundreds of millions of dollars. Without a comprehensive system to manage enterprise content, the return on that investment will be minimal. If your business isn't managing its content efficiently, it's essentially losing money and wasting time. Worse, if you're not managing your content, you're probably not managing your business to its full potential.

## *Documentum ECM – Managing Ideas from Concept to Content*

Documentum products facilitate the management of content as it progresses through a lifecycle – from creation to publishing to archiving. More than that, though, Documentum products make automation of this lifecycle possible while enabling collaboration at every point along the way. Easy-to-use workflow tools automate business processes by routing content through review and approval cycles and ultimately to production. At the same time, Documentum enables users to collaborate with one another naturally and captures the results of discussions, e-mail, chat, and instant messaging. Combining casual collaboration and structured workflow, Documentum speeds ideas from concept to content.

### *Create/Capture*

Creation encompasses all the ways in which content enters your organization: documents captured through scanning, reports captured from a variety of applications, and original content created as Microsoft Word files, spreadsheets, photographs, digital video, and more.

### *Manage*

Managing content across an extended enterprise requires a range of capabilities, such as business process automation, library services, content security, and collaboration services. These functions are at the core of Documentum's early leadership in document management and the company's continuous evolution as the industry leader in ECM.

### *Deliver*

Creating and managing content are only the beginning. The ultimate objective is to distribute that information wherever it's needed – whether that's to a corporate Web site, a portal, a mobile device, a hardcopy form, or a CD-ROM. Documentum helps get content where it needs to go.

### *Archive*

Documentum automates the process of archiving content. The system enables you to control the lifecycle of content beyond its immediate usefulness – particularly critical in a business climate in which careful management of records has become a paramount concern.

# *Enterprise Integrations and Development Tools*

Through its enterprise integration and development tools, Documentum helps you see beyond creation, management, delivery, and archiving to envision new solutions to business challenges. As a true enterprise platform, Documentum can be integrated with other enterprise applications such as ERP and CRM for more effective use of knowledge and information. Documentum also provides a full set of standards-based development tools to help you create packaged and custom applications that leverage business-critical content.

## *Integrate*

ECM software must be integrated within an organization's overall software environment. At Documentum, we've formed partnerships with other enterprise software makers to ensure that our platform works seamlessly with leading enterprise applications like BEA, PeopleSoft, SAP, and Siebel.

## *Develop*

We place a high value on our developer community, and we work hard to provide technical resources and easy-to-use tools for developers to build packaged and custom content applications based on Documentum. To make development of Documentum applications as easy as possible, we provide integrations with standard development tools and environments.

## *Content is a Competitive Advantage*

With a Documentum ECM solution, content is a competitive advantage. Products get to market faster. Collaboration with partners becomes much easier. Marketing collateral is approved and published quicker. Customer service is more responsive. And Web sites are fresh and up-to-date instead of stale and potentially inaccurate.

## *Development of Content Applications*

When you purchase an enterprise content management solution, you are purchasing a development environment as well. And it is this environment, as much as the core ECM system, that can make or break the overall success of a deployment.

That's why we work closely with the development community to provide programming interfaces (APIs), technical resources, and easy-to-use tools for developers to build custom content-rich applications based on Documentum. A standards-based approach allows developers to work within development environments they're already familiar with. Our developer program Web site, <http://developer.documentum.com> serves as

a centralized location for education, training, product releases, partner information, and migration roadmaps that help you plan projects more effectively. The site is also a one-stop resource for documentation, tips, sample code, white papers, and tutorials that you can use on a daily basis to improve your development skills or solve specific programming challenges.

**Develop** with Documentum.

## *Introduction to WDK 5*

Web application developers are increasingly tasked with building applications that provide more and more sophisticated experiences for end users. To support these efforts, Documentum is transforming our Web-based development environment into a tag-based, server event model that makes development of complex applications easier, faster and more powerful.

The new development environment, WDK 5, represents a major overhaul to existing WDK 4.2.x technology. Documentum has committed to building all future versions of Web-based content management clients using WDK 5 including Documentum Webtop, Documentum Web Publisher, Documentum Administrator™, and Documentum Content Services for Portals.

WDK 5 allows developers to take advantage of recent advances in Web application development technology. WDK 5 supports the notion of segmenting the presentation, application, business and data layers within the Documentum platform stack. This architecture is consistent with the direction that Microsoft is taking with .NET and the direction that Sun is taking with JavaServer Faces.

## *Getting Started*

This document was created to provide you with an overview of the product so that you may prepare yourself and your organization to utilize this valuable web development product. Documentum itself will use WDK, in combination with an application server, as the foundation for its new web clients. For this reason we included both configuration and customization tutorials in this document to prepare you for the tasks that you are likely to complete when you work with our packaged web clients, or build your own web-based content applications.

## *Contact Information*

If you would like further information about WDK, its availability or product plans please contact one of the individuals listed below.

- **Lubor Ptacek:** Director, Platform Product Marketing  
**email:** lubor.ptacek@documentum.com

**Telephone:** (925) 600-6790

- **Kevin O'Connor:** Sr. Product Manager, WDK

**email:** kevin.oconnor@documentum.com

**Telephone:** (925) 600-5733

- **Erin Samuels:** Technical Marketing Manager, WDK

**email:** erin.samuels@documentum.com

**Telephone:** (925) 600-5453

## Where to Go from Here

If you prefer to install software, play with it and learn the details later:

- Start with Chapter 6 and follow the instructions for installing WDK.
- Continue to Chapter 11 and complete the self-guided tutorials.
- If you still have questions about the underlying technology, read Chapter 4 and 5. Most of your questions will be answered there.

If you prefer to read documentation thoroughly and fully understand the subject matter you are working with before starting in with it:

- Read Chapter 2 to get a background on all of the development options within Documentum.
- Continue on to Chapter 3 to learn more about the applications that will be utilizing WDK 5.
- Chapter 4 and Chapter 5 will teach you about WDK architecture and how it works along with Java and J2EE technologies.
- Continue on to Chapter 6 and follow the instructions for installing the GA release
- Read Chapter 10 to learn about building a WDK 5 application and complete the self-guided tutorials in Chapters 11 and 12.

# *Additional Development Options*

Documentum provides a platform for developing and deploying enterprise content management. This chapter will cover the following topics:

- Documentum Foundation Classes (DFC)
- Business Objects Framework (BOF)
- RightSite, Docbasic and DQL
- Moving Forward with WDK
- Model-View-Controller Paradigm

## *DFC*

Documentum Foundation Classes (DFC) is an object-oriented API layer written in Java that provides direct access to a robust and dependable API set, a variety of industry-standard languages and developer tools, and all platform functionality. DFC enables rapid development of scalable, componentized JSP, Web, and custom applications.

DFC exposes the Documentum object model as an object-oriented client library. It is leveraged within the Client/Server architecture, as in Documentum Desktop Client, and within the Web architecture, as in WDK.

DFC consists of the following:

- A set of interfaces that provide the DFC programming model. This programming model specifies how an application interacts with the DFC objects that implement these interfaces and the relationship between these DFC objects. The objects in this model consist of persistent objects stored in a Content repository (Docbase) and non-persistent objects returning from a query or generated by DFC.
- A Java implementation of this model. It consists of several Java packages each contains a number of Java classes and interfaces. It also provides tracing and exception handling.
- A set of Java classes packaged with supporting DLLs and shared libraries. In addition to full Java accessibility, it includes support for accessing DFC from Visual Basic or Visual C++ through COM on Windows platforms.

## Business Objects Framework

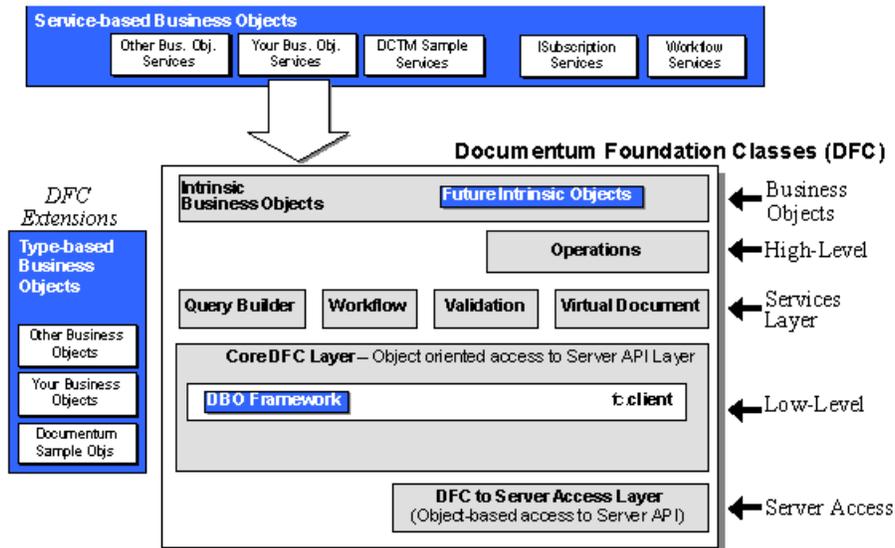
The Documentum Business Objects Framework (BOF), new with DFC 5, provides an object oriented framework for building, testing, discovering, and deploying business logic as Business Objects.

### BOF Overview

BOF is part of the DFC library and consists of several Java classes and interfaces for constructing and running Documentum Business Objects (DBO). There are two types of Documentum Business Objects, type-based business objects, and service-based business objects. A type-based business object can extend a Documentum Server persistent object type (e.g. "com\_accelera\_catalog" or "com\_accelera\_autonumber" might extend "dm\_document") and extend its capabilities by providing new methods for those types (actually, new user defined Java classes) and allowing overriding of existing methods, like checkin() and checkout(). A service-based business object provides methods that perform more generalized procedures that are not usually bound to a specific object type or even Content repository (Docbase).

The specific business logic within the business objects is implemented using the standard DFC framework allowing developers to implement a new DBO without a steep learning curve. Since BOF objects are based on the Documentum Foundation Classes (DFC), the developer can maintain a high degree of compatibility with existing DFC applications.

Figure 2-1. DFC Provides DBO Framework



## *RightSite, Docbasic, and DQL*

Documentum RightSite is an extension of the Documentum system. It was the first solution to satisfy the complex challenges of capturing, managing, and assembling an organization's frequently changing information on corporate Web sites. Documentum RightSite lowered the cost of web application development by automating the laborious, manual tasks involved in keeping content current and easy to manage. Unlike first-generation, static web solutions, Documentum RightSite dynamically assembles Web pages "on the fly" and enables tailoring of web content to users' unique needs -- ensuring that only the information relevant to a user's job function is delivered to the desktop. In addition, Documentum RightSite preserves the many hyperlinks associated with a corporate Web site to ensure that links always point to the appropriate version of a page.

RightSite includes a set of development tools that can be used to enable your Web site to take full advantage of Documentum's content management capabilities, to customize Documentum intranet clients, or build custom applications. These development tools include a macro language (WebQL), a scripting language (Docbasic), and a set of methods that are built into RightSite. You can also use any Web application development languages such as JavaScript, CGI, or Java.

Documentum Query Language (DQL) uses syntax that is a superset of ANSI-standard SQL. It provides additional content management-specific extensions to SQL such as the REGISTER statement. DQL statements operate on objects and sometimes on tables/rows while SQL statements operate on tables or rows only.

Other Web application development toolkits provide the user with only pre-built presentation templates or an API (similar to our DFC) to provide for the retrieval of data from the JSP template or Servlet, which acts as the controller. In either case the Web application developer must either script all of the behavior into the body of the JSP page (combining the presentation layer and application logic) or create their own framework (such as WDK provides) to process user requests and populate presentation components accordingly. Since WDK delivers this functionality, development teams working with it can be much more effective building applications much quicker and easier than from scratch.

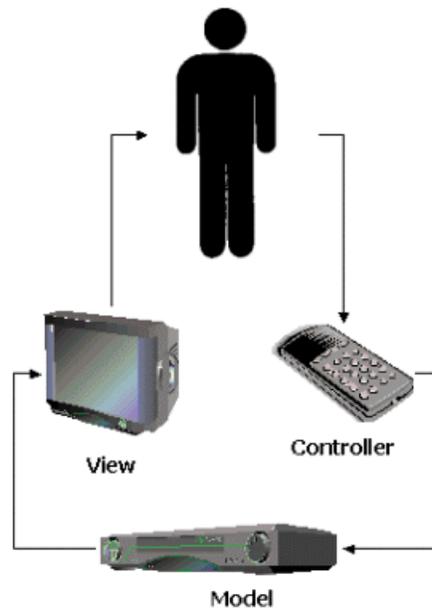
## *Moving forward with WDK*

WDK goes further than any of Documentum's prior Documentum web development tools by providing the front end service framework and components used to build Web applications. The service framework provided in the WDK binds the application to the two key environments for WDK-based applications, the Documentum Content Server platform and the J2EE Application Server. The components provide a JSP implementation of the view to be provided to the end consumer of the application. Separating these two provides a separation of the physical code so that developers of user interfaces, we call Web Designers, can work on those tasks in an environment familiar and comfortable to them. Similarly those developers who are most proficient in the development of server-side code can work on the behavior classes which each component extend.

## Model-View-Controller Paradigm

If you are struggling with the notion of a Model-View-Controller patterned application you can compare it to the diagram below. In this diagram you would see that the user (top) requests data from the model (VCR) by invoking actions on the controller (remote control) that result in a display being presented in the view (TV). WDK works in much the same way, where users send events to the component behavior classes (remote control), which in turn interacts with the Content Server (model) and presents a new/updates view to the end-user via the execution of a JSP template (TV).

Figure 2-2. Model, View, Controller



# *Standard Documentum Web Applications*

This chapter will provide an overview of the applications that will be built on the WDK infrastructure. This chapter will cover the following topics:

- Documentum Webtop
- Documentum Web Publisher
- Documentum Administrator (DA)
- Documentum Content Services for Portlets

## *Documentum Webtop*

Webtop is the standard library services Web client. It provides a browser interface for managing content, which makes this product intuitive and extremely easy to use. Business users access content via check-in and check-out mechanisms that ensure content integrity. Administrators have the ability to deploy Webtop to an entire enterprise with the click of a button.

## *Documentum Web Publisher*

Documentum Web Publisher is a browser-based tool for creating and updating Web content in a highly efficient and scalable manner, engaging all users across the enterprise. The integration to standard authoring tools allows contributors to work in familiar desktop environments and empowers non-technical users to create Web content without requiring them to understand HTML or XML.

Documentum Web Site Manager provides Web administrators with powerful site management functionality, enabling them to review versions of one or more sites at any phase in the staging or production process. Administrators can manage multiple staging and deployment configurations and provide work-in-process and staging areas for development and contribution teams.

## *Documentum Administrator (DA)*

Documentum Administrator provides a universal point of access from any desktop platform for managing and administering all repositories, servers, users, and groups, regardless of their location across the virtual enterprise. By automating administrative tasks, Documentum Administrator protects the integrity of business-critical processes while lowering the cost of owning and maintaining e-business applications.

## *Documentum Content Services for Portlets*

Documentum has pre-packaged the most significant and common content management functions for Portals, such as content contribution, browsing, and workflow management. These services, called Portlets, will be delivered as JavaServer pages, Java classes, and XML files to be exposed through the portal interface. These Portlets are based on the WDK 5 framework. Documentum offers a Portal Integration Kit to guide portal builders who will integrate Documentum Portlets with portal deployments as part of a complete customer solution. For more information on the Portal Integration Kit, please refer to Chapter 5 in this document.

## *WDK 5 Technology Overview*

This chapter will present the architecture and underlying services within WDK 5. This chapter will cover the following topics:

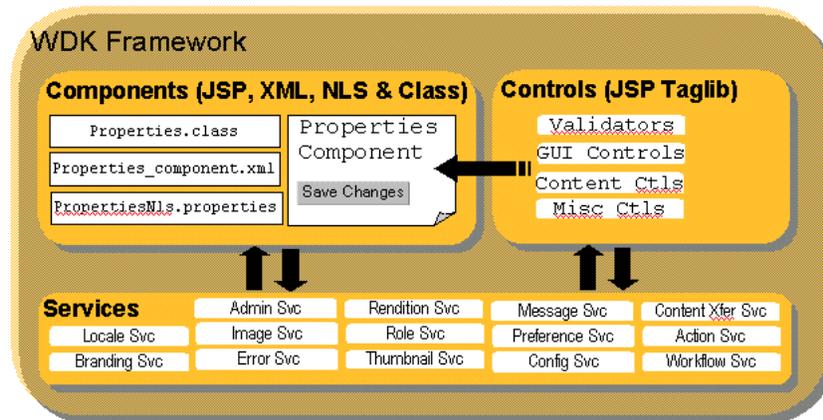
- This chapter will present the architecture and underlying services within WDK 5. This chapter will cover the following topics:
- Service Framework
- Presentation, Component, Application, and Programming Models
- Compatibility and Migration
- Exposing Documentum Platform Features with WDK 5
- Documentum WDK 5 Component Manifest

### *WDK 5 Architecture*

WDK 5 was designed to deliver consistency of presentation and behavior. This effectively joins robust Documentum functionality with the highly visible appearance of the application. This creates a powerful application development and customization model for not only building the user interface, but for the creation of feature-rich web applications built on Documentum. The Documentum behavior is delivered through mechanisms, such as data models and operations, in a manner that provides reusable business logic.

The WDK Framework is shown in the following figure:

Figure 4-1. WDK Framework



## Service framework

The service framework consists of services provided by the Documentum Foundation Classes (DFC) and third parties:

Table 4-1. Service Framework

<b>Service</b>	<b>Description</b>
J2EE	JSP 1.1 and Servlet 2.2 support for JSP pages, servlets, and web applications
Application server	JSP container support for J2EE-compliant web applications
I18N	The Java SDK management of internationalized resources
Data access driver	The JDBC driver that performs direct read-only access to the Docbase
Authentication	J2EE Principal authentication and manual Docbase authentication
Business objects	The DFC classes that model Documentum-specific transactions and session management
Session pooling	The dmcl.ini configuration of WDK as a client of the server connection pooling

## Presentation model

The presentation model defines a methodology for building dynamic web pages. The model separates layout from behavior and allows you to reuse user interface elements without affecting the application behavior. The presentation model is built using JSP tag libraries to generate the corresponding HTML. Through the application server, Java classes on the server model the data that is presented to the user. Data sources include DFC and JDBC connections to the Content Server.

In addition to the presentation tags and behavior classes, the WDK framework provides other important services, including:

Table 4-2. Other WDK Services

<b>Service</b>	<b>Description</b>
· History mechanism	Maintains browser history and navigation
· Configuration service	Looks up configuration contracts for actions and components
· Branding engine	Manages the look and feel for the application
· Action service	Enforces prerequisites for actions
· Content transfer services	Performs content transfer between the client and the Docbase

## Component model

The component model provides a configurable, encapsulated set of related Documentum functions. The framework enforces a contract for the component consisting of parameters that initialize the Component, events that respond to user actions, and properties that interrogate the state of a component.

The component contract is defined in an XML configuration file, and the definition includes a component behavior class. A component may include preconditions (actions), and it may include other components, acting as a container. Components are dispatched based on the component type: WDK 4.x components, WDK 5.x components, and generic components such as HTML, ASP, or raw JSP pages.

## Application model

The application model enforces a consistent look and behavior across all applications contained within the root application. The J2EE-compliant root context may contain member applications that inherit application parameters from other members. For example, the base application is WDK. The WDK application is extended by the web component application, which in turn is extended by the WebTop application. Your application can then extend the WebTop application.

Using the branding engine framework, the application can leverage application themes that provide your application's unique look and feel through icons, images, and style sheets.

You can incorporate your existing Web-based applications as front-end components of a WDK-based application.

### *Portals*

The Portal Integration Kit extends the Documentum content management platform by allowing the development of Portal specific Integration Layers. Each Integration Layer allows WDK-powered Components to be rapidly exposed through the associated Portal framework, leveraging existing Component development.

### *Programming model*

The WDK programming model is object-oriented. It models the HTML form state and lifecycle, which are missing from standard HTML forms. It adds navigation Doabase session management, with data sources provided by DFC and JDBC.

A Documentum web application consists of a set of components that are composed of JSP pages, supporting behavior classes, and XML configuration files. The JSP pages are modeled by form classes that manage state and navigation. The user interface in forms is modeled by controls, which are widgets represented by JSP tags. The WDK framework of services, such as the configuration, action, messaging, and tracing services, supports the application.

### *Compatibility and Migration*

WDK is backwardly compatible to support custom WDK 4.2 components. This includes interoperability between the WDK 4.2 components and the new components. The more than 60 components that are currently available with WDK 4.2 will continue to be available with the release of WDK 5.1. In the new environment, you can mix and match 4.2 with 5.1 components. You may also convert WDK 4.2 components into WDK 5.1 components.

WDK 5.1 pages must include the taglib directive, which maps the custom tag prefix to the appropriate tag library for tags on that page, as in the following example:

```
<%@ taglib uri="/WEB-INF/tlds/dmform_1_0.tld" prefix="dmf" %>
<%@ taglib uri="/WEB-INF/tlds/dmformext_1_0.tld" prefix="dmfx" %>
```

These declarations allow the processing server to link in the classes necessary to support the execution (interpretation) of statements on the JSP page. For example (from advsearch.jsp):

```
<dmf:datadropdownlist name='param1' width='120'
  onselect="onParamSelected">
```

```
<dmf:dataoptionlist>  
  <dmf:option datafield="docid" labeldatafield="name"/>  
</dmf:dataoptionlist>  
</dmf:datadropdownlist>
```

In this example, tags are declared that will be used in the page itself. This allows you to reuse code modules without having to locate them physically within each WDK component or JSP page.

## *Documentum Features Exposed in WDK 5*

### *XML*

Documentum 4i includes end-to-end capabilities for creating, managing, and delivering XML content uncovering new opportunities for business-to-business content exchange. No other vendor provides such complete functionality for XML content management:

- Automatic receipt and mapping of XML documents
- Support for ebXML to enable B2B content exchange
- Web-based templating for XML content creation
- Integrations with leading XML authoring tools
- DTD and schema validation
- Configurable storage of XML content
- Powerful search capabilities across structure, content, and metadata
- Component and link management support
- Dynamic XML document generation and assembly
- Transformation of XML documents with integrated XSLT support

For more information, please refer to ***Managing XML Content in Documentum***.

### *Workflow*

One of the key features of the Documentum enterprise content management system is the ability to manage content through workflows. Workflows provide the capability for content contributors, managers and consumers to be alerted when it's their turn to interact with a content object. The Documentum workflow engine provides for two types of workflow's: document-centric and task-centric. In document-centric workflows involvement is based upon the "state" of the content, in task-centric workflows involvement is based upon the "activities" that are being performed on the content. In both types of workflows there are several states for the content being controlled, including: active, promoted, suspended, demoted and completed.

The WDK components for workflow fully expose the functionality of the Documentum eCMS. Furthermore, the WDK-based applications that Documentum will be providing will make extensive use of the workflow engine and can therefore serve as a reference implementation for anyone interested in building workflow applications using WDK.

## Component Manifest

The following feature-rich components, functionality, and services are examples of what is available with WDK 5.1:

Table 4-3. Core Library Services and Navigation Components

---

Login	Move	Locations
Logout	Multi-file Move	Menu
Create New Document	Copy	Version History
Create New Folder	Multi-file Copy	Object Grid
Create New Cabinet	Link	Prompt
Document List	Multi-file Link	Relationships
Drilldown	Delete	DRL
Home Cabinet	Multi-file Delete	Folder Tree

Table 4-4. Administration Components

---

List Group	List Users	View Groups for a User
Locate Group	Locate Users	Change Password
Create Group	Create Users	Locate Permissions Set
Modify Group Properties	Import Users	View User Permissions
Add Users to Group	Delete Users	View Where Permission Set is Used
Delete Users from Group	Reassign Users	Edit User Permissions
Delete Group	Change User's Home Content repository (Docbase)	Delete Permissions Set
Reassign Group	Change User's State	Delete Users from Permissions Set
Rename Group	Modify User Properties	

Table 4-5. ACL Management Components

---

ACL Information	Delete ACLs
List ACLs	ACL Properties

Table 4-6. Content Transfer Components and Applets

Import	Checkin from File	Multi-file Edit
Multi-file Import	Checkout	View Checked-Out Documents (My Files Component)
Export	Multi-file Checkout	Full Content Transfer Applet
Multi-file Export	Cancel Checkout	Lightweight Content Transfer Applet
Checkin	View	
Multi-file Checkin	Edit	

Table 4-7. Lifecycle Management Components

Apply Lifecycle	Promote Lifecycle	Suspend Lifecycle
Demote Lifecycle	Resume Lifecycle	

Table 4-8. Properties Components and Functionality

Data Dictionary Support	View Primary Attributes	Mandatory Field Flag
Attributes Component	View Secondary Attributes	Properties Sheet Container
Show All Attributes	History (Audit Trails)	

Table 4-9. Rendition Component Functionality

View Rendition	Import Rendition	Create HTML Rendition
Delete Rendition	Export Rendition	Create PDF Rendition

Table 4-10. Search Components

Advanced Search	Save Searches (dm_query objects)	Execute Saved Searches (dm_query objects)
Basic Search	View Saved Searches (dm_query objects)	

Table 4-11. Virtual Document Manager Components

Convert to Virtual Document	Add Child	Reorder Children
Convert to Simple Document	Remove Child	Node Management

Table 4-12. Workflow Components

Start Workflow	Reject Router Task	Workflow Task Information
Pause/Halt Workflow	Reject Workflow Task	Workflow Task History
Stop/Abort Workflow	Repeat Workflow Task	Workflow Task Manager
Resume Workflow	Delegate Workflow Task	Workflow Task Progress
Workflow Status	Rerun Failed Automatic Workflow Task	Workflow Status List
Send to Distribution List	Complete Failed Automatic Workflow Task	Workflow Status Drilldown
Forward Router Task	Remove Attachment	Inbox
Forward Workflow Task	Workflow Task Comments	

Table 4-13. Utility Components and Functionality

My Files Component	Thumbnail Viewing	Clipboard Component
Subscriptions Component	Skins (Color Schemes)	

Table 4-14. XML Support

Checkin XML	Import XML
Checkout XML	Export XML

Table 4-15. Component Containers

(Root) Container	Dialog	Properties Sheet
Combo	Navigation	Wizard

Table 4-16. Component Locators

ACL Object	Subscription Document	Recent User or Group
Add Child Sysobject	Subscription Folder	Recent Workflow Template
Alias Set Object	My Document	Subscription
All Document	My Folder	Sysobject
All Folder	My User Workflow Template	User Only
All Lifecycle	My Workflow Template	User or Group
Lifecycle Folder	Persistent Object	Group Only
All User Workflow Template	Recent Document	Workflow Group Only
All Workflow Template	Recent Folder	Workflow Template Folder

Attachment	Recent Group	Workflow Template Subscriptions
My Object Attachment	Recent Lifecycle	Workflow User or Group from Group
Subscription Attachment	Recent Sysobject	
Document Locator Container	Recent User Only	

*Table 4-17. Services*

Action	File Checker	Locale
Branding	Help	Message
Configuration	History	Preferences
Clipboard	HTTP Sessions	Rendition
Content Transfer	Content repository (Docbase) Sessions	Role
Error Message	Image	Tracing

*Table 4-18. Developer and Administrator Features*

API Message Tester	DQL Query Editor
Component List	Role Support



## *WDK Technology in Depth*

This chapter will provide a description of the technologies being used with WDK 5. The topics of this chapter include:

- Java, J2EE, and Tag Libraries
- WDK 5 Application Example
- Component Model
- Controls
- Forms
- Services
- Anatomy of a WDK 5 Application
- The WDK Container

### *Introduction*

The greatest difference between WDK 1.0.1/4.2.x and WDK 5 is the introduction of a tag library and new server-side event model. We have added the tag library to encapsulate the user interface controls in a more readable and maintainable manner, in addition to simplifying the task of web design. The new server-side event model was introduced to reduce the amount of JavaScript that prior WDK versions used and to improve performance through the JavaServer Faces model that we are using.

Tag-based programming provides a simple way to build reusable code blocks that separate the presentation layer from the application logic layer. WDK 5 provides a set of tags and associated application logic; Web designers need only make use of these tags to build Web applications. If custom application behavior is required, Web developers can create new tags and associated code modules. This approach makes it possible for a Web designer to create powerful Web applications without having extensive Web development skills.

While much has been written about .NET, JavaServer Faces is a relatively recent addition to the technology landscape. JavaServer Faces is a Java Specification Request (JSR) that proposes to “establish a standard API for creating Java Web application GUIs, which will eliminate the burden of creating and maintaining GUI infrastructure for developers”. Documentum holds two positions on the expert group that is working on this JSR. To learn more about JavaServer Faces, see <http://www.jcp.org/jsr/detail/127.jsp>.

## *Java, J2EE, and tag libraries*

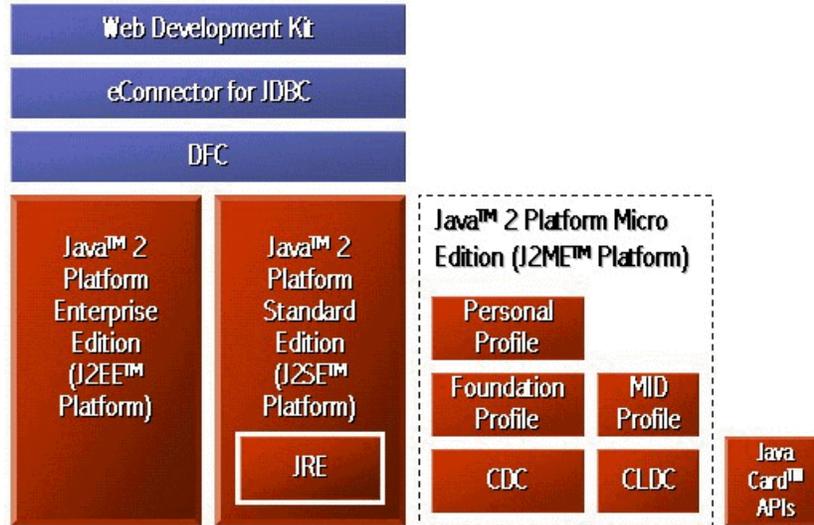
Simply stated Java is one of the most, if not the most, pervasive technologies ever used to develop software. Due to the deployment of Java technology by so many organizations in so many applications, its widespread adoption is impossible to quantify. However, we can tell from this widespread adoption that the technology has gained a prominent position in the development of most modern web applications. In May 2001 the Java technology celebrated its sixth birthday with the announcement of its latest platform, the J2ME for developing Java-based applications for handheld devices. This announcement is significant in that it represents the continuous development of the technology to further extend the reach of Internet and web technologies. Documentum has strongly embraced Java technology by means of its Java-based products, including:

*Table 5-1. Documentum Java-based Products*

<b><i>Product</i></b>	<b><i>Description</i></b>
DFC	A set of foundation classes used by all Documentum applications. The DFC provides developers with a java based API and object model to develop Documentum clients.
eConnector for JDBC	Early in the lifecycle of Java technology development it was decided that a native bridge to RDBMS servers, and the like, was critical to make Java applications as rich and data-driven as possible. Documentum has followed that standard by releasing its own JDBC compliant driver to access the Documentum Content Server. JDBC technology has taken a prominent position in the J2EE specification and is testament to why Documentum products use this technology for Content Server access.
WDK	Documentum delivers not only APIs for Java Development (DFC and JDBC) but also a set of extensive web application development components that provide a full implementation of the user interface (JSP pages), application services (App Server side control classes) and a data-binding service.

This diagram illustrates what the Java Platform looks like today:

Figure 5-1. Documentum Java Platform



WDK is built on J2SE and J2EE, displayed in the top left-hand corner of this diagram.

## J2EE

introduced in its earliest forms, three years ago. That introduction was paramount, as it provided the set of application building tools and technologies that would not only set the standard for building enterprise applications but also promote a design pattern for building such applications (Model-View-Controller). Using the MVC Design pattern web applications were built with a separation of the business or application logic from the data model upon which it is based and the user interface which serves as the navigation mechanism for the application. The sum of all contributions made by the J2EE standard for web application development are too numerous to list here, but for more information regarding the use and successes of Java/J2EE-based web applications see this section of Sun's Java web site: <http://industry.java.sun.com/casestudies/>.

## JavaServer Pages and Tag Libraries

JavaServer Pages share a critical role in both the J2EE platform as well as the WDK. In both platforms JSP provides a dynamic template that is executed by the application server to create the HTML that a browser will display. In both cases these JSP pages get compiled into Servlet code, for optimal performance. JSP pages usually contain a structure similar to a static HTML document with the exception of dynamic execution tags “<%... %>”. Within these tags a developer can insert raw java code to be executed, class import statements, and even tag library definition statements. In WDK you will see that all of the pages include the declaration of associated tag libraries that are going to be referred to in the pages, these declarations will look like this:

```
<%@ taglib uri="/WEB-INF/tlds/dmform_1_0.tld" prefix="dmf" %>
```

```
<%@ taglib uri="/WEB-INF/tlds/dmformext_1_0.tld" prefix="dmfx" %>
```

These declarations allow the processing server to link in the necessary classes which will support the execution (interpretation) of statements within the JSP page, such as this:

```
<dmfx:datadropdownlist name='param4' width='120'  
  onselect="onParamSelected">  
  <dmfx:dataoptionlist>  
    <dmfx:option datafield="docid"  
      labeldatafield="name"/>  
  </dmfx:dataoptionlist>  
</dmfx:datadropdownlist>
```

In this example, we are declaring the tags that will be used in the page itself. This allows the developer the freedom of reusing code modules without having to physically locate them within each WDK component, or JSP page. To find out more about JSP technology refer to Sun's J2EE website at <http://java.sun.com/products/jsp/>.

## *JavaServer Faces and JSR 127*

Today, nearly every application being developed is built for deployment on the web. As this trend has developed, so to have web application development patterns. Most popularly is the three-tier "Model, View, Controller" pattern. This pattern prescribes the separation of a web application into three discrete tiers.

*Table 5-2. Three Tiers of a Web Application*

<b><i>Tier</i></b>	<b><i>Description</i></b>
Model	Data model, which the application modifies
View	Presentation components (JSP/HTML)
Controller	Application logic which governs the interaction with the model and in some cases is solely responsible for transaction integrity

Recently, a proposal has been made to the Java Community Process committee to incorporate a framework into the JSP specification that would govern the interaction between web components and the controller tier. This proposal is Java Community Process request #127. While the specification loosely defines what a component could be comprised of it does strongly suggest the appropriate interaction between components and server side classes to deliver applications to the client.

## WDK 5 Application Example

### *Portal Integration Kit*

The Portal Integration Kit (PIK) is an extension of WDK. It provides the capability of presenting existing WDK components through an off-the-shelf Portal framework with minimal or zero development cost. This is achieved through the development of a Portal Environment Adapter and Portlet Containers.

A single Portal Environment Adapter and set of Portlet Containers (collectively known as a Portal Integration Layer) are required for each Portal framework. The Portal Integration Kit provides all the necessary support to develop the Portal Integration Layer and once built, any WDK Component may be presented through that Portal.

The Portal Environment Adapter maps WDK Environment services to associated Portal services. This allows a WDK Component to leverage the Portal framework without being exposed directly to it. The Portal can then drive the behavior of the WDK Component.

A Portlet Container houses a WDK Component in the Portal User Interface. The container ensures the WDK component behaves appropriately within the Portal when a user interacts with it and also provides any Portal specific adornments. Each Portal may require any number of containers depending on how the WDK Component is housed e.g. a front page, dialog or external application.

To facilitate the development of a new Portal Integration Layer, the Portal Integration Kit provides a reference implementation that may be used as a starting point. Each Portal Environment Adapter complies with a set of straightforward contracts that are defined by the Adapter framework. The Contracts are Authentication, Branding, Rendering, Internationalization, Configuration, Caching and Component Inter-communication. Each Portal Environment Adapter may choose the level of integration by implementing only those contracts that are required.

WDK packaging and deployment features are complemented by the Portal Integration Kit to allow a set of Component based Portlets to be installed as a WDK-powered application. An additional simple registration scheme is provided to associate a Portal Integration Layer with each WDK deployment.

### *Tight Portal Integration through WDK Customization*

By utilizing the WDK customization model it is possible to build specialized versions of a Component where its presentation or behavior (or both) may be tuned to the problem at hand. This feature can be used to take an existing WDK Component and tailor it to a Portal framework. For example, the presentation of an Inbox Component may be customized to add a Portal specific banner, or its behavior may be customized to add support for Portal specific caching.

The combination of WDK customization and Documentum's out-of-the-box EIP Portlets provides a head start when developing Documentum based content-rich Portlets.

## *Component Model*

The component model provides a configurable, encapsulated set of related Documentum functions. It provides a user interface (UI) that is sensitive to the context in which a component is called. For example, a properties component can have several UI representations and behaviors, based on the type of object that is viewed. The specific UI that is presented to the user is driven by parameters that are sent to the component, such as an object ID and a mapping specified in the component definition XML file. The framework enforces a contract for the component, consisting of parameters that initialize the component, events that occur in response to user actions, and properties that interrogate the state of a component.

Components can also present a UI based on the implementation mechanism: JSP, XSLT, WDK 4.2, or WDK 5.1. You can choose the appropriate UI implementation for your application, and you can migrate components from JSP to WDK 4.2 or from WDK 4.2 to WDK 5.1 with minimal impact.

Components are built on top of the WDK infrastructure. The Component class is an extension of the Form class, providing support for configuration lookup, containers, Content Server access, and context-based navigation.

## *Reuse*

Each Component supports a contract or public interface through which all other Components and Containers communicate. The contract consists of the following elements:

- Parameters, which initialize the component
- Events, which support a response to user intervention
- Properties, which interrogate the component state

The contract discourages access to the internals of a component, so that the implementation can change over time without impact to the caller. The contract also insulates the component from dependencies on other components. Individual components can be reused in multiple containers without relying on dependent components.

A component can be extended, for example, to add support for new types, without changing the component's caller. Components inherit or override the contractual behavior of the component that they extend. Thus an extended component doesn't need to modify or copy the existing component.

## *Component dispatcher*

The component dispatcher allows a container to call or include a component. The dispatcher maps the component URL to the appropriate implementation URL. It is implemented as a Java Servlet, which is registered in the J2EE web server web.xml file.

## Controls

### *Overview of controls and tags*

The smallest unit of user interface function is called a control. The control is represented by a JSP tag from one of the Documentum tag libraries. The control tag class sets and gets attributes on the control and renders the HTML output for the tag. The control behavior is defined in a control class.

The application user interface is built from controls. A control is a Java object that models the attributes of HTML UI elements. The user interface state is maintained by setting JSP tag library attributes for a control tag class. The state is mirrored on the server by the control class.

The control fires lifecycle and change events. The control class defines an event handler method to handle the control events. The form processor manages the state of the control.

Each control has a corresponding control tag class that initializes the control and generates the user interface. The tag class implements tag library accessor methods to get and set the control's attributes.

### *How controls and tags work together*

The page designer sets default values for the control attributes on the JSP page. When the client browser requests the JSP page, the UI control display is initialized with the control attribute values on the JSP page and rendered to the browser as HTML and JavaScript.

When a user changes a control value, the new value is submitted when the form is submitted, along with any other control changes. The form then updates the control state on the server and also performs any operation that is triggered by form submission.

## Forms

Forms represent UI pages in WDK-based applications. A form is a specialized control that models a web page on the server. The form can include other forms, but generally there is a one-to-one correspondence between a form and a web page.

The form object performs the following functions:

- Implements event handler methods
- Contains controls and properties
- Exposes a contract to ensure integrity when the form is invoked
- Implements methods for redirection to another form

### *Form model*

A form is an object that models a web page on the server. A form models a single web page, except in the case of included forms or wizards.

Forms are addressable via a URL to the JSP page that includes the form tag.

### *Form processor*

The form processor automatically generates lifecycle events for forms to enable the management of server resources. The lifecycle allows multiple JSP requests:

- First request: Form initialization and rendering. Validation does not occur during form initialization.
- Subsequent requests: Change events, action events, and re-rendering
- Last request: Change events and form exit

### *Services*

One of the key capabilities of the Documentum Web Development Kit is the service layer which provides data to the components for servicing requests. This data can range from application context and configuration data to role and security data. The Documentum WDK provides several services to assist web application developers. The following sections define a few of the services delivered through WDK, more complete documentation regarding these services and their usage will be available in the documentation accompanying the BETA and GA versions of WDK.

#### **Configuration Services**

Configuration is supported by several services:

- Configuration service
- Action service
- Role service

### *Configuration Service*

The configuration service supports application, component, action definition and configuration. The service provides general support for holding and retrieving settings that can have various scopes (for example, Docbase type, role, or application). Settings for the configuration service are maintained in XML files stored in each application's /config directory. These directories store the individual configuration files for both the components and the actions that are used by a WDK-based application. When an application is loaded by the application server, the contents of the configuration files are cached into memory for access by the configuration service. Additionally, the configuration service employs an NLS resource so that values stored within the configuration files can be language-independent.

Here is an example of a component configuration file that is read and used by the configuration service during runtime:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>
<config>
  <scope>
    <component id="menubar">
      <params>
      </params>
      <pages>
        <start>/webtop/classic/menubar/menubar.jsp</start>
      </pages>
      <class>com.documentum.web.formext.component.Component
      </class>
      <nlsbundle>
        com.documentum.webtop.webcomponent.menubar.MenuBarNlsProp
      </nls_bundle>
    </component>
  </scope>
</config>
```

## Action Service

The action service isolates the application logic that is invoked by users. The action service is designed to support both UI operations (navigating within the browser or frames) and Content Server interaction. Actions are bound to UI components (buttons, links, menu items) and governed by their definitions. The definition of an action is declared within an XML definition file. For example, this action element is from `dm_cabinet_actions.xml`:

```
<action id="newcabinet">
  <params/>
  <preconditons>
    <precondition class="com.documentum.webcomponent.library.actions.
      NewCabinetPrecondition"/>
    <precondition class="com.documentum.web.formext.action.RolePrecondition">
      <role>coordinator</role>
    </precondition>
  </preconditions>
  <execution class="com.documentum.web.formext.action.LaunchComponent">
    <component>newcabinet</component>
    <container>newcabinetcontainer</container>
  </execution>
</action>
```

The action is used in `drilldown_body.jsp`:

```
<!--Folder level actions -->
<dmfx:actionlink cssclass='actions' name=newcabinetbtn' nlsid=
  'MSG_NEW_CABINET' action='newcabinet' showifdisabled-'false'>
  <dmfx:argument name='objectId' contextvalue='objectId' />
</dmfx:actionlink>
```

### *Role Service*

The role service provides a way to restrict user access to certain controls based on the group to which they belong. This mechanism is built into filters that can be applied to action and component declarations. This filtering mechanism can grant or deny users access to controls, either directly or through inheritance.

An alternative to filtering is **scoping**. Scoping grants a component or action access based on the role context of the calling user request.

The following examples show the difference between filtering and scoping.

- Filtering (from `attributes_dm_sysobject_component.xml`):

```
<scope type="dm_sysobject">
  <component id="attributes">
    ...
    <!-- Enable show-all-attributes (for administrators) -->
    <filter role="administrator">
      <enableShowAll>true</enableShowAll>
    </filter>
    ...
  </component>
</scope>
```

- Scoping (based on `attributes_dm_folder_component.xml`):

```
<scope type='dm_folder', role="administrator">
  <component id="attributes"
    extends="attributes:webcomponent/config/library/attributes/
    attributes_dm_sysobject_component.xml">
    <!-- Component Layouts -->
    <pages>
      <start>
        /webcomponent/library/attributes/attributes_dm_folder.jsp
      </start>
      <component>discussions</component>
    </pages>
  </component>
</scope>
```

## *Anatomy of A WDK 5 Application*

### *Source File Types*

WDK contains the several kinds of source files to use in your web applications.

Table 5-3. WDK Source File Types

<b>Source File Type</b>	<b>Description</b>
XML	Configuration files for actions, components, sub-applications, and the root application.
Text	Resource files with a .properties extension. These files serve two purposes: One type of properties file is read by framework classes to initialize configurable parameters in those classes; the other type contains English strings that are present in the user interface and can be localized into other languages.
JSP	JavaServer pages that contain HTML, JavaScript, and JSP tags. The JSP tags are instantiated as Java objects on the server.
Images	Icons, bullets, and other types of graphical widgets.
CSS	Cascading Style Sheets, to govern the look and feel of your application.
Java classes	Server classes that provide the framework and building blocks of a Documentum-enabled application. In addition to WDK classes, WDK provides archives (JAR format) that contain Documentum Foundation Classes (DFC), the IBM regular expression parser, and the Documentum JDBC connector.
Native libraries	Platform-specific dynamic code libraries that are used by DFC on the server to perform Docbase operations.

## The WDK Container

In building our own WDK-based applications we determined that a core set of components greatly improved the common look, feel, layout and behavior of our web applications. We have thus decided to group a number of these components into one master component, for ease of use, and deemed it as the 'Container' component. This container implements functionality that should be available in all WDK applications, provided by Documentum or others, including:

- Help System functionality
- Clipboard functionality
- Error Message handling
- Inter-frame messaging
- Branding (set and modify skins)

## Control Manifest

The following controls are examples of what is available with WDK 5.1:

Table 5-4. General Controls

Webform	Radio	Input Mask Validator
Form	Datagrid	Dropdown List
Panel	Datagrid Row	Data Dropdown List
Hidden	Row	Listbox
Label	No Data Row	Data Listbox
Text	Column Panel	Option
Text Area	Data Paging	Data Option List
Password	Data Sort Link	Image
Button	Form Include	Outline Bar
Link	Validation Summary	Breadcrumb
Checkbox	Required Field Validator	Tab Bar
Menu	Required Expression Validator	Tab
Menu Item	Range Validator	Argument
Menu Separator	Compare Validator	Tree
Menu Group	Form URL	Date Value Formatter

Table 5-5. Content repository (Docbase) Enabled Controls

Action Button	Docbase Selector	Checkin Registry Cleanup Applet
Action Link	Browser Tree	Link Detector Applet
Action Menu Item	Component URL	Service Progress Feedback
Action Multi-select	Component Include	ACL Class Value Formatter
Action Multi-select Checkbox	Container Include	Store Value Formatter
Action Image	Argument	Store Status Value Formatter
Action Link List	Parameter	Permission Value Formatter
Action Button List	VDM Tree Grid	Extended Permissions Value Formatter
Thumbnail	VDM Tree Grid Row	Document Format Value Formatter
Docbase Object	Documentum Applet	Document Size Value Formatter

Docbase Attribute	Cancel Checkout Applet	Repeating Attribute Formatter
Docbase Attribute Label	Checkin Applet	Rank Value Formatter
Docbase Attribute Value	Checkout Applet	Folder Exclusion Formatter
Docbase Icon	Edit Applet	VDM Binding Rule Formatter
Docbase Lock Icon	Export Applet	Docbase Attribute Validator
Docbase Priority Icon	Import Applet	Docbase Object Validator
Docbase Workflow State Icon	View Applet	Quote Validator
Docbase Foldertree	Utility Applet	



## *Preparing for installation*

The following topics describe the tasks that will prepare your server, host, and Docbase for WDK installation:

- [Preparing clients, page 6- 1](#)
- [Preparing the application server host for installation, page 6- 2](#)
- [Preparing Docbases \(4.x Docbases only\), page 6- 5](#)
- [Preparing information for the installer, page 6- 5](#)

## *Preparing clients*

You must prepare the client hosts that will access a WDK application.

**Check in checked out objects** — If client hosts have checked out objects using DFC version 4.2 or lower, they will not be located by DFC 5. Documentum began using the HKEY\_CURRENT\_USER registry hive recommended by Microsoft with version 4.3 of Documentum products. If your clients have such objects checked out when they connect to WDK 5, they can manually check in those objects by selecting Checkin from File. It is recommended, however, that clients check in all objects before connecting to a WDK 5 server.

**Remove earlier versions of WDK applets** — Earlier versions of WDK including WDK 5 early access installed applets that must be removed, because they use a different version of DFC than WDK 5.

*To remove client applets from previous WDK versions:*

1. Log in to the client host as the user.
2. Start Internet Explorer and choose Tools⇒Internet Options.
3. On the General tab, in the Temporary Internet Files section, click Delete Cookies.
4. In the same section (Temporary Internet Files), click Delete Files. Check Delete all offline content and then click OK.

**Set all clients to use the same JVM** — All clients (browser host machines) must use the same JVM, that is, either the Sun Java plugin or the Microsoft JVM.

*To set the JVM used by client browsers*

1. Edit the app.xml file in the /wdk directory.
2. To support the Microsoft JVM, set the value of <usepluginforie> to false. To support the Sun Java plugin, set the value to true.
3. To support the Microsoft JVM, disable the Java plugin for Internet Explorer on the client. To support the Sun Java plugin, enable the Java plugin. See .

*To check which JVM is being used by IE*

1. Open Internet Explorer and select Internet Options on the Tools menu.
2. Click the Advanced tab and make sure that all three options under Microsoft VM are checked.
3. Open the Windows Start menu and navigate to Programs -> Accessories -> System Tools -> System Information
4. Open Internet Explorer in the navigation tree. You should see Java VM Version 5.0 or higher listed in the content pane on the right. If the Java VM is not listed, it may have been replaced by the Sun Java plugin.

*To enable or disable the Java plugin*

1. Open Start Menu > Settings > Control Panel
2. Select the Java Plug-in
3. On the basic Tab, uncheck "Enable Java Plug-In" to disable the plugin. Check to enable it.
4. Client system: If there is a browser tab (some versions of the plugin) uncheck "Microsoft Internet Explorer". App server, uncheck both IE and Netscape.
5. Some users have reported that they had to reinstall the Microsoft JVM. Contact Microsoft technical support for a JVM installer.

## *Preparing the application server host for installation*

This topic describes the steps you must follow to prepare the server host for WDK installation:

- [Uninstalling Documentum applications and DFC, page 6- 3](#)
- [Uninstalling WDK 5 beta or cleaning up a failed WDK 5 installation, page 6- 3](#)
- [Installing the Java application server, page 6- 5](#)
- [Backing up customizations, page 6- 5](#)

See the WDK release notes for a list of requirements applicable to the WDK application server host.

## *Uninstalling Documentum applications and DFC*

Before you install WDK, you must uninstall all Documentum applications that use DFC 4.x and uninstall DFC 4.x itself from the application server host before you install WDK 5. WDK installs DFC version 5, which cannot be used by a Documentum product with a version lower than 5.1.

DFC 5 installs to a single location on the host, and that installed version of DFC is used by all DFC clients.

## *Uninstalling WDK 5 beta or cleaning up a failed WDK 5 installation*

You must [manually uninstall the beta release](#) of WDK if you participated in the beta program. The beta release did not contain an uninstaller. You can also follow these steps to [cleanup after a failed installation](#).

The instructions below refer to the location in which you have installed Documentum client support as **DOCUMENTUM**. The default home directory in the Windows installation is C:\Program Files\Documentum. There is no default home directory in the WDK 5 Solaris installation. The examples in the installation instructions suggest a home directory of /export/Documentum.

*To manually uninstall the beta version of WDK 5*

1. Check in all documents that you have checked out, or cancel checkout on those documents.
2. If you have other installed products that use DFC, delete only the directory **DOCUMENTUM/wdk** and then go to step 5. If you are also uninstalling DFC, go to step 3.
3. (Windows DFC cleanup) If you have only one DFC client application on your server machine and you wish to remove DFC, run the DFC uninstaller from the Windows Control Panel (Add or Remove Programs) if it is available in the Control Panel and then delete the directory **DOCUMENTUM** and the (Windows only) the user directory (Default C:\Documentum).
4. (Windows DFC cleanup) Remove the system environment variables that were written by the DFC installer. Go to the Windows Control Panel ⇒ System and click the Advanced tab. Select Environment variables and remove the variables DFC\_HOME and DFC\_DATA, remove the Documentum jar files from the ClassPath variable, and remove the Documentum directories from the Path variable.
5. (Windows DFC cleanup) Remove the registry key \\HKEY\_LOCAL\_MACHINE\SOFTWARE\Documentum
6. (Windows) Remove the Start menu links to WDK 5 documentation by deleting the directory **Root drive**\Documents and Settings\All Users\Start Menu\Programs\Documentum.

7. Remove the directory or Web application archive (\*.war file) that the WDK installer created in your application server deployment directory. For example (Windows): **CATALINA\_HOME**\webapps\**wdk** or **BEA\_HOME**\webapps\**wdk**
8. Remove the generated Java and class files directory under the application server home directory. For example: **CATALINA\_HOME**/work/Standalone/wdk.
9. (Tomcat) Open the Tomcat batch file **CATALINA\_HOME**/bin/catalina.bat and remove the lines that were added by the installer. They are marked by a comment that indicates they were added by the WDK installer.
10. Remove the applets on the client: Navigate to the %Windir%\Downloaded Program Files directory. Right-click on Documentum Content Transfer Applets 5.1 and choose Remove.

*To manually clean up a failed installation of WDK 5*

You may not be able to perform some of the steps below, depending on the reason for the failed installation. Attempt to perform all of them if possible.

1. If you are not uninstalling DFC because you have other installed products that use DFC, delete only the directory **DOCUMENTUM**/wdk and then go to step 5. If you are also uninstalling DFC, go to step 3.
2. (Windows DFC cleanup) Remove the system environment variables that were written by the DFC installer. Go to the Windows Control Panel ⇒ System and click the Advanced tab. Select Environment variables and remove the variables DFC\_DATA, remove the Documentum jar files from the ClassPath variable, and remove the Documentum directories from the Path variable.
3. (Windows DFC cleanup) Remove the registry key \\HKEY\_LOCAL\_MACHINE\SOFTWARE\Documentum
4. (Windows) Remove the registry key \\HKEY\_LOCAL\_MACHINE\SOFTWARE\Documentum
5. (Windows) Remove the Start menu links to WDK 5 documentation by deleting the directory **Root\_drive**\Documents and Settings\All Users\Start Menu\Programs\Documentum.
6. Remove the directory that the WDK installer created in your application server deployment directory. For example (Windows): **CATALINA\_HOME**\webapps\**wdk** or **BEA\_HOME**\webapps\**wdk**
7. Remove the generated Java and class files directory under the application server home directory. For example: **CATALINA\_HOME**/work/Standalone/wdk.
8. (Tomcat only) Open the Tomcat batch file **CATALINA\_HOME**/bin/catalina.bat and remove the lines that were added by the installer. They are marked by a comment that indicates they were added by the WDK installer.
9. (Windows) Remove the applets on the client: Navigate to the %Windir%\Downloaded Program Files directory. Right-click on Documentum Content Transfer Applets and choose Remove.

## *Installing the Java application server*

Before you run the WDK installer, make sure you have installed a supported Java application server on the WDK server host. See the WDK release notes for the list of certified application servers and platforms.

## *Backing up customizations*

If you create customizations based on WDK and wish to reinstall WDK, back up the following before you reinstall:

- JSP files in the custom directories
- Configuration files in the custom directories
- Changed strings in the /wdk/strings, /webcomponent/strings, and /webtop/strings directories
- Branding files in the custom /theme directory
- Compiled custom Java classes in the /WEB-INF/classes directory and subdirectories. Note that after reinstallation, you should recompile your custom classes to make sure that the update has not broken any custom code that depends on basic WDK classes.
- Configured properties files in the /WEB-INF/classes/com/documentum subdirectories

## *Preparing Docbases (4.x Docbases only)*

To use the subscriptions features, you must define the object relations used by the subscription service. These relationships are defined by running scripts in each of the Docbases that your Web application connects to. Run the scripts before any clients connect to a WDK application. The scripts are located in the directory /webcomponent/install. The following scripts are provided:

- subscriptionInstall.dql
- subscriptionUninstall.dql

To use full text search with the simple search in WDK, you must turn on full-text indexing in yourIndexing is turned off by default in 4.2.x Docbases.

## *Preparing information for the installer*

Prepare the following information for the installer before you run the installer:

- [Preparing Docbroker information, page 6– 6](#)
- [Setting Unix environment variables, page 6– 6](#)
- [DFC installation directories, page 6– 6](#)

- (WebLogic only) Creating a domain for the application, page 6– 8

### *Preparing Docbroker information*

The Web application server host must have a dmcl.ini file that names a Docbroker. The Docbroker lists the available Docbases known to that Docbroker. If your server host does not have a dmcl.ini file, the WDK installer will create one with the Docbroker information that you provide.

Determine the name of the DocBroker host and port on the application server that listens for the Docbroker. The default DocBroker port is 1489. You will be prompted for the name and port during installation if the installer doesn't find a valid dmcl.ini file (Windows) or a DMCL\_CONFIG environment variable (Unix) on your system.

### *Setting Unix environment variables*

On Solaris, log in as the user who is the application server instance owner. Do not log in as root. Set the following environment variables and create the corresponding directories if they do not exist:

- DOCUMENTUM: Specifies the path to Documentum home directory in which Documentum applications are installed. For example:  
`DOCUMENTUM=/export/home/Documentum`
- DOCUMENTUM\_SHARED: Specifies the path to the shared Documentum directory in which DFC and any other shared files are installed. This directory can be the same as the Documentum home directory. For example:  
`DOCUMENTUM_SHARED=$DOCUMENTUM`
- DFC\_DATA: Specifies the path to the parent of the Documentum configuration directory. The config directory is located by default under the Documentum home directory. For example:  
`DFC_DATA=$DOCUMENTUM`
- DMCL\_CONFIG: Specifies the path to the dmcl.ini file. If the WDK installer does not find this environment variable, it prompts you for the docbroker host name and port (default 1489) and then creates a dmcl.ini file in the \$DOCUMENTUM directory.

### *DFC installation directories*

The WDK installer runs a DFC installer that will prompt you with choices for the DFC installation. If a previous version of DFC 4.x was installed, you must remove it before running the WDK installer. If the installer finds a previous version of DFC 5.1, the installer updates DFC in the previous installation location and optionally installs the DFC documentation. The DFC documentation is installed in C:\Program Files\Documentum\Shared\help\DFC (Windows) or \$DOCUMENTUM/help/DFC (Unix).

If you are installing DFC for the first time on the WDK server, you see two dialog windows:

- Select Documentum home directory: The installer creates this directory and a /config subdirectory that contains DFC configuration files. The installer puts dctm.jar into the system classpath, adds a DFC environment variable to the Windows system, and puts the DFC install directory in the Windows system path. The standard location for Windows is C:\Program Files\Shared for Windows, and for Solaris the value of your DOCUMENTUM environment variable.
- Select Documentum user directory: DFC client applications use the user directory for user files such as checked out files when the host runs as a client. (If you use the host for development, the host machine can also connect to the Web application as a client.) The default user directory is C:\Documentum.

The following jar files and DLLS are installed in the Windows DOCUMENTUM directory:

dexpn40.dll  
dmcl40.dll  
dfc.jar  
DfRegistryWin32.dll  
DfVDMPlatformUtils.dl  
DcDJCB.dll  
DfcStubs.dll  
bsf.jar  
log4j.jar  
xercesImpl.jar  
xmlParserAPIs.jar  
xalan.jar  
xml-apis.jar  
docncm22.dll  
docncx22.dll  
docndc22.dll  
docndd22.dll  
docndg22.dll  
docnol22.dll  
docnpb22.dll  
docnrn22.dll  
docntl22.dll  
docnuasm.dll  
DfCDExcel97.dll  
DfCDLegacy.dll  
DfCDPowerPoint97.dll  
DfCDWord97.dll  
Dfc.dll

### *(WebLogic only) Creating a domain for the application*

BEA WebLogic requires a separate domain for the WDK application. You must create this domain before you run the installer. When you install and deploy on BEA, the installer prompts you for the domain and server alias for the WDK application.

*To create a WebLogic domain:*

1. Start the BEA configuration wizard from the Windows Start menu.
2. Select a domain template such as WDL Domain, and name the template. (The following examples use the domain name dctm.) Click Next.
3. Choose the server type, for example, single server (standalone server).
4. Choose the domain location, for example, (Windows) C:\bea\user\_projects or (Solaris) **BEA\_HOME**/user\_projects. The server creates a directory under the WebLogic user projects directory for the new domain. For example, if the new domain is named dctm, WebLogic creates **BEA\_HOME**/user\_projects/dctm
5. Configure the server:
  - a. Server name: Server reference that is used internally by BEA, for example, myserver. You can assign any name you want to the server as long as it is not the same as your domain name.
  - b. Server listen address: Server alias, such as localhost.
  - c. Server listen port: Enter the number of the port on which the server will listen for HTTP requests. The default port is 7001.
  - d. Server SSL listen port: Enter the number of the port on which the server will listen for secure HTTPS requests. The default port is 7002.

### *Installing WDK for customization of Webtop or other Web application*

If you plan to customize Webtop or another Web application, you must first install the application and then run the WDK installer. Select the option "Customize an existing application." This option copies the Webtop application, or any other Web application that you choose at the location you specify, to another location that you specify. If the existing Web application is in war file format, the WDK installer expands the application and updates all WDK files. The WDK installer also installs the WDK help files and adds a shortcut to the help to the Windows start menu.

For example, you have installed Webtop in the WebLogic dctm domain with all of the default installation options. Webtop is located at /**BEA\_HOME**/user\_projects/dctm/applications/webtop-config. In the customization selection screen, you enter the following information:

In the server details screen, you enter the following information:

Domain name  
dctm

Server name  
myserver  
Virtual directory  
webtop-custom

The installer will copy the application in /webtop-config to a new application directory named webtop-custom and will update the WDK files within the new application directory.



**Caution** Customization overwrites the WDK application files if they exist in the target (not source) location. Make a backup copy of all files in the Web application before you customize it with the WDK installer.



## *Installing WDK*

The WDK installer will deploy a WDK base application to an application server or to a standalone directory on your file system. If you run the installer to update a custom application, the installer will copy your application to your selected location and add the required files for a WDK application.

The following topics describe the installation procedures for WDK 5:

- [Installing on Windows, page 7- 1](#)
- [Installing on Solaris, page 7- 3](#)
- [Testing your deployment, page 7- 4](#)
- [Uninstalling WDK, page 7- 6](#)

## *Installing on Windows*

The following section describes the steps required to install WDK on Windows.



**Caution** Do not install WDK as root.

### *Installation procedure*

1. Download the installer from the [Documentum download site](#) to a temporary directory on your application server host. If you cannot locate your password for the download site, use the password finder on the bottom of the download page. Use your email address as your user name.
2. Expand the installer to your temporary directory.
3. Stop all running programs including any Java application servers.
4. Start the WDK installer by double-clicking on wdkSetup.exe. You see three introductory screens: Welcome, License, and a reminder to stop any running application servers.
5. If you have never installed DFC, you see a dialog to choose the destination directory for WDK and other DFC client applications. The default location is C:\Program Files\Documentum.
6. Select any optional DFC features to install. If DFC has been installed previously on your system, DFC will be upgraded in the previous installation location. See [DFC installation directories, page 6- 6](#) for a description of the DFC installation options.

7. If you have never installed DFC, you see a dialog to choose the user directory, which will be used when the server host is used as a client. The default is C:\Documentum.
8. If the installer does not find a dmcl.ini file on the local host it prompts you to enter the Docbroker host name and port. See [Preparing Docbroker information, page 6– 6](#) for more information.
9. Choose the setup type for the WDK installation:
  - Customize an existing Web application: Copies the selected application to a new location and adds the installer version of WDK files and help documents. The screens for customizing a Web application are the same as the screens for creating a new Web application, with the additional selection of the application to be customized (step 10 below)
  - Create a new Web application: Deploys WDK to the selected application server or as a standalone directory on the file system and copies the help files and DFC files to the file system.
10. (Customization only) Select an existing Web application (war file or directory) and then browse to the location of the existing application to be copied and customized. For example (WebLogic), if you are customizing Webtop in the default location, select the file webtop.war file. See [Installing WDK for customization of Webtop or other Web application, page 6– 8](#) for more details on customizing a Web application..
11. Select an application server and enter application server details.
  - a. Select one of the supported application servers from the list.
  - b. Enter the application server detail. Go to step c for Tomcat. Go to step d for BEA.
  - c.
    - i. Enter the application server location, which is the full path on the host machine to the Tomcat root directory. The default Tomcat root directory is:  
`C:\Program Files\Apache Tomcat 4.0`
    - ii. Enter the virtual directory, which is also called the context root or the alias that will be used in URLs for your web application. For example, for a Web application with the URL `http://server_name/wdk5/...`, the context root is “wdk5”. To avoid confusion when you are working with file paths and URLs, name the context root something other than wdk because there is a wdk directory under the root directory. The WDK tutorial uses wdk5 for the virtual directory name.
  - d.
    - i. Enter the application server location. The default is C:\bea\weblogic700 (Windows).
    - ii. Enter the app server user projects location. The default is C:\bea\user\_projects (Windows).
12. You see a prompt to confirm your selections. After you confirm, the installation will proceed.
13. After installation you may be prompted to reboot the system. A reboot is required to initialize DFC if the installer encountered locked files during installation.

## Installing on Solaris

The following section describes the steps required to install WDK on Unix.

### *Installation procedure*

Before you install WDK, you must create the environment variables described in [Setting Unix environment variables, page 6– 6](#) .

1. Download the installer from the [Documentum download site](#) to a temporary directory on your application server host. If you cannot locate your password for the download site, use the password finder on the bottom of the download page. Use your email address as your user name.
2. Expand the installer to your temporary directory.
3. Stop all running Java application servers.
4. Navigate to the installer directory and make the installer executable. Enter the following command:

```
chmod +x wdkSetup.bin
```

At the command line, start the WDK installer by entering the following command:

```
./wdkSetup.bin
```

You see three introductory screens: Welcome, License, and a reminder to stop any running application servers.

5. You see a dialog to select the location for DFC installation. Enter the location that DFC will be installed to.
6. You see an option to install the DFC documentation. If DFC has been installed previously on your system, DFC will be upgraded in the previous installation location. See [DFC installation directories, page 6– 6](#) for a description of the DFC installation options.
7. You see a dialog in which you must enter the Docbroker host name and port. The installer creates a dmcl.ini file for you. See [Preparing Docbroker information, page 6– 6](#) for more information.
8. Choose the setup type for WDK installation:
  - Customize an existing Web application: Copies the selected application to a new Web root directory and installs the WDK libraries into the application. The screens for customizing a Web application are the same as the screens for creating a new Web application, with the additional selection of the application to be customized (step 9 below).
  - Create a new Web application: Deploys WDK to the selected application server or as a standalone directory on the file system and copies the help files and DFC files to the file system. Go to step 10 below.
9. (Customization only) Select an existing Web application (war file or directory) and then browse to the location of the existing application to be copied and customized. For example (WeLogic), if you are customizing Webtop in `\usr\local\bea\user_projects\dctm\webtop`, select that directory as your source Web application. See [Installing WDK for customization of Webtop or other Web](#)

[application, page 6– 8](#) for more details on customizing a Web application. Proceed to step 10.

10. Select an application server and enter application server details.
  - a. Select one of the supported application servers from the list. See the release notes for the application servers that are certified with this release.
  - b. Enter the application server details:
    - i. Enter the application server location, which is the full path on the host machine to the BEA root directory. The default BEA root directory is:  

```
/usr/local/BEA/weblogic700
```
    - ii. Enter the application server projects directory, which contains the domain that you set up for Documentum Web applications. This directory is created by the WebLogic configuration wizard.  
(Solaris) For example:  

```
/usr/local/BEA/user_projects
```
11. Enter additional WebLogic details:
  - a. Domain name: Enter the name of the domain that you created for Documentum Web applications. Suggested domain name: `dctm`
  - b. Server name: Enter the name of the BEA server. You can use `myserver` or any other server that you have defined in the WebLogic configuration wizard.
  - c. Enter the virtual directory, which is also called the context root or the alias that will be used in URLs for your web application. For example, for a Web application with the URL `http://server_name:port/wdk5/...`, the context root is “`wdk5`”. To avoid confusion when you are working with file paths and URLs, name the context root something other than `wdk` because there is a `wdk` directory under the root directory. The WDK tutorial uses `wdk5` for the virtual directory name.
12. You see a summary of what will be installed, and then installation begins.

## Testing your deployment

The following instructions describe how to start the application server and test the WDK deployment.

If you have installed WebLogic or Tomcat as a Windows service, the WDK installer modifies the service startup parameters to include WDK parameters. You can start the service to test your deployment.



**Caution** Launch your application server from the batch file that was created (BEA) or customized (Tomcat) by the WDK installer. Do not launch the application server from the Windows Start menu or from the Windows services control panel. The default startup file for the server will not contain WDK classpath information, and WDK applications will not run.

To test WDK installed in WebLogic:

1. Start the WebLogic server from the command line using the batch file startWDK.cmd (Windows) or startWDK.sh (Solaris) in the **BEA\_HOME**/user\_projects/**DOCUMENTUM\_DOMAIN** directory.

2. From the local server, view the default home page URL. The default home URL is:

```
http://localhost:7001
```

3. Test the WebLogic examples following the instructions in the WebLogic documentation.

4. Enter a WDK test URL in your browser. Test the following sample pages (assumes a Web alias of wdk5):

```
http://localhost:7001/wdk5/wdk/samples/dumpRequest.jsp
http://localhost:7001/wdk5/wdk/samples/menuZoo.jsp
http://localhost:7001/wdk5/wdk/samples/treeTest.jsp
http://localhost:7001/wdk5/wdk/samples/sessionZoo.jsp
http://localhost:7001/wdk5/component/login?startUrl=/wdk/samples/index.jsp
```

To test WDK installed in Apache Tomcat:

1. Start the Tomcat server from the command line using the batch file catalina.bat in CATALINA\_HOME/bin. (This batch file is modified by the WDK installer to include the WDK classpath.)

2. At the command prompt, navigate to the **CATALINA\_HOME**/bin directory and enter:

Windows:

```
startup
```

Solaris:

```
./startup.sh
```

Windows tip: You can start Tomcat with the command “catalina run” to have the output appear in a DOS window. If Tomcat crashes, you can see the output in the DOS window.

3. Open the default server home page in a browser. For example:

```
http://localhost:8080
```

4. Click on the JSP Examples link on the left side of the Tomcat home page. Execute one of the examples to make sure that Tomcat is properly configured to serve JSP pages.

5. Enter a WDK test URL in your browser. Test the following sample pages (assumes a Web alias of wdk5):

```
http://localhost:8080/wdk5/wdk/samples/dumpRequest.jsp
http://localhost:8080/wdk5/wdk/samples/menuZoo.jsp
http://localhost:8080/wdk5/wdk/samples/treeTest.jsp
http://localhost:8080/wdk5/wdk/samples/sessionZoo.jsp
http://localhost:8080/wdk5/component/login?startUrl=wdk/samples/index.jsp
```

# Uninstalling WDK

The uninstaller removes files and directories created by the WDK installer, except that it does not remove DFC. DFC may be used by other Documentum applications, so the uninstaller does not remove it.

See also the instructions on [manually uninstalling the WDK beta release](#) or for [cleanup after a failed installation](#).

*To uninstall WDK from Windows:*

1. Navigate to the directory DOCUMENTUM\uninst\wdk.
2. Double-click uninstall.exe.

*To uninstall WDK from Solaris:*

1. Navigate to the \$DOCUMENTUM/\_uninst/wdk directory.
2. Enter the following command:  
`./uninstall.bin`

## *Building a WDK 5 Application*

This chapter will outline the powerful, yet simple to implement, configuration and customization options available with WDK 5. It will also provide some guidance in assembling the components to build your first application. This chapter will cover the following topics:

- Building the application infrastructure
- Building a custom component library
- Component configuration
- Component customization

### *Building the Application Infrastructure*

The following sections provide more information on how to locate and assemble WDK components.

### *Locating Components and Resources*

This release of WDK contains several types of source files that can be used in your web applications. This chapter focuses on the following types of files:

- XML definition files
- JavaServer Pages
- Java classes

XML definition files are stored in two directories: ***wdk5***/wdk/config and ***wdk5***/webcomponent/config, where ***wdk5*** is the directory in which WDK is installed.

JSP pages are stored in multiple directories throughout the WDK file structure:

```
wdk5
 /wdk
  /container
 /webcomponent
  /admin
  /componentTestbed
  /environment
  /library
  /navigation
```

The component library of component JSP files are located in **wdk5**/webcomponent/library, in folders identified by the names of the components. The behavior classes for the components are located in the **wdk5**/WEB-INF/classes/com/documentum/webcomponent/library in folders identified by the names of the components.

The component source files (with a .java extension) are available to developers in a separate zip file. The files provide an example of how to work with DFC from within the WDK components.

**Note:** It is a best practice to extend classes rather than relying on uncompiled Java code when you customize applications.

## *Assembling the Components to Build an Application*

Assembling WDK application components involves the following considerations:

- Your application directory must reside in **wdk5**/custom. The WDK installer creates this directory for you.
- An app.xml file must be created to define the application. The path for this file is **wdk5**/custom/app.xml.
- Keep your configurations and customizations in a separate folder. See [Table 8-1, page 8-3](#) for a description of how the custom layer is organized.
- It is good practice to use a default page, such as index.html or index.jsp. (See [Tutorial 1: Creating a Start Page for the Application, page 10-1](#).)

## *Building a Custom Component Library*

### *Preserving your Changes*

Creating a custom layer for configured and customized components enables you to reuse base components in multiple applications while safeguarding changed components.

When you begin a customization, copy the files you need to customize into the appropriate location in the custom layer, as described in the following table.

Table 8-1. Directory Structure of the Custom Layer

<b>File type</b>	<b>Customization Directory</b>	<b>Example</b>
XML definition	<b>wdk5</b> /custom/config/	<b>wdk5</b> /custom/config/ attributes_dm_document_ component.xml
JSP page	<b>wdk5</b> /custom/ <b>component_</b> <b>name</b> /	<b>wdk5</b> /custom/attributes/ custom_attributes_dm_ document.jsp

## Common Configurations and Customizations

Documentum WebTop is a general-purpose library services client. Most common customizations are required to change look and feel, including framesets, colors, logos, and related images. Use the WDK branding service for this type of requirement. For more information see ***Configuring Documentum Web Development Kit Applications*** and ***Customizing Documentum Web Development Kit Applications***.

Other customizations require changing one or more WDK components. You can change components by **configuring** them, which requires changes to either the JSP pages or XML configuration files, or by **customizing** them, which requires changes to the component class.

## Component Configuration

When you configure a component, you change the user interface using JSP pages and XML definition files. Both the layout and behavior of the component can be modified declaratively without rebuilding the component. Generally, each configured component is tied to a different calling context.

## XML Configuration

Configure components and actions by using XML definition files in WDK. A component definition file defines the contract for a component, such as required and optional parameters, JSP pages, and the component behavior class.

A component configuration can extend the contract for another component, providing component inheritance. Actions can be configured as a kind of filter for Docbase transactions based on context, such as object type, user role, or a user-defined criterion.

## XML Definition Files

The following table describes the major XML elements and attributes that are used in a component definition file, such as `attributes_dm_sysobject_component.xml`.

Table 8-2. Elements of an XML Component Definition File

<b>Element Name</b>	<b>Attributes</b>	<b>Function</b>	<b>Notes</b>
<code>&lt;scope&gt;</code>	type, role, user-defined	Context mapping	
<code>&lt;component&gt;</code>	id	Logical name used by component caller	
<code>&lt;params&gt;</code>		Component contract	
<code>&lt;param&gt;</code>	name, required	Specific parameters	required="true" specifies that parameter is mandatory
<code>&lt;pages&gt;</code>	logical_name	Configurable element name; value is a URL	
<code>&lt;pages&gt;</code>		Presentation pages	
<code>&lt;class&gt;</code>		Component class that defines presentation behavior of the component JSP pages (fully qualified name)	Valid class values: <ul style="list-style-type: none"> <li>• No class element (for raw JSP or HTML)</li> <li>• WDK4Component</li> <li>• component_class_name</li> </ul>
<code>&lt;nlsbundle&gt;</code>		Class that contains externalized strings for the component class and JSP pages	Properties file in bundle can be localized.
<code>&lt;custom_tag&gt;</code>		User-defined tags that control component configuration	Requires user-written code to implement the tag.

## Parameters

The parameters that are expected by the component are defined in the XML definition file by the `<params>` tag. The component class ensures that the appropriate parameters are passed to the component when it is first called. The component class enforces the use of required parameters (as specified by the `required="true"` attribute).

For example, `attributes_dm_sysobject_component.xml` specifies three parameters:

```
<params>
  <param name="objectId" required="true"></param>
  <param name="readOnly" required="false"></param>
  <param name="enableShowAll" required="false"></param>
</params>
```

The component dispatcher passes the parameters to the attributes component. The component class then gets the parameters in the `onInit` event handler method:

```
public void onInit(ArgumentList args)
{
  super.onInit(args);
  String strObjectId = args.get("objectId");
}
```

User-defined tags can be added to the contract if you extend a component class.

## Custom Elements

Use the following component methods to look up configuration element values:

Table 8-3. Component Methods

<code>lookupString()</code>	Gets the String value of an element
<code>lookupInteger()</code>	Gets the Integer value of an element
<code>lookupBoolean()</code>	Gets the Boolean value of an element

For example, the browser tree class in Webtop calls the `lookupString` method for the variable `strDefaultTab` as follows:

```
strDefaultTab = lookupString("nodes.docbasenodes.entrynode");
```

The value for the custom element `<entrynode>` would be obtained from the following configuration in the XML definition file:

```
<component...>
  <nodes>
    ...
    <docbasenodes>
      ...
      <entrynode></entrynode>
    </docbasenodes>
  </nodes>
</component>
```

In this particular example, the `<entrynode>` element has no value.

## Scope

A qualifier such as application name, object type, or user role can scope configuration definitions. The configuration service also supports user-defined scopes.

The configuration service supports multiple scopes and multiple component definitions within each scope. For example, a properties component can have a component

definition for `dm_document` and one for a custom object type. For an example, see [Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages, page 10– 8](#).

### *Inheritance*

The elements and values of a configuration definition can be inherited from a different configuration definition. There is no limit to the number of levels of inheritance. For example, a base properties component scoped on the `dm_sysobject` object type can represent the default behavior of all type-scoped properties components. The properties component can then be extended to define a properties component scoped to a user-defined type. The extended component inherits its definition from the base component and overrides values or adds parameters specific to the new scope.

### *JavaServer Pages*

JSP pages are the main mechanisms for controlling layout. JSP pages implement controls and include any other presentation layer features that an application requires.

### *Component Customization*

Customization is the process of using DFC and standard Java programming to extend the behavior of components. A component is customized through code development. This allows the component to provide additional or alternative behavior based on the calling context.

### *Customization Tools*

In order to customize WDK, you need a compiler. You can use any J2EE-compliant compiler, such as the one provided by Sun, or you can use the compiler integrated into any J2EE-compliant IDE.

### *Using DFC*

Customization using DFC functions and calls is a common requirement for enhanced functionality. You can write custom code using DFC that will work seamlessly with your WDK application. The customization tutorial in this manual shows a simple customization using DFC.

## *Using Standard Java*

It is very important that custom classes be compliant with the J2EE code standards that WDK employs. This will ensure a smooth integration with the product as well as with the J2EE application server that is being used.

### *Servlet 2.2*

The Servlet 2.2 specification describes web applications. WDK provides the following web application content:

- JSP pages
- Servlets: Server-side classes that encapsulate behavior and state
- Client-side applets: Java classes that run in the client browser and access the local file system and registry
- Resources, including images and tag libraries

A web application runs in a J2EE-compliant JSP container, which provides the Java Runtime Environment (JRE) and, usually, the JSP translator (compiler). Each JSP page is translated into a servlet class and instantiated whenever the JSP page is requested.



## *Preparing the WDK Environment*

### *Environment for the WDK Tutorials*

The procedures in this guide are based on the following environment:

- Sun Java JDK 1.3.1 (or a later release of 1.3.1, such as 1.3.1\_04), available from <http://java.sun.com/j2se/1.3/>
- Tomcat 4.0.4, available from <http://jakarta.apache.org/tomcat/>
- WDK 5.1, installed at the Tomcat 4.0.4 location (and this tutorial uses the virtual directory name wdk5)

See the WDK release notes for hardware requirements and software certifications. See the WDK installation guide for information on installing WDK and deploying to an application server.

- Sun ONE Studio 4, Community Edition (formerly Forte for Java Community Edition), available from <http://www.sun.com/software/sundev/jde/buy/index.html>

**Note:** You may use any application server certified for WDK, and you do not need Sun ONE Studio to run and customize WDK Web applications. These tutorials use a text editor for the configuration tutorials. You can use any J2EE-compliant IDE to complete your configuration and customization tasks, or even a text editor plus a command-line compiler for the customizations. Instructions for creating a project in Sun ONE Studio for the customization tutorials are provided in [Setting up the WDK Project in Sun ONE Studio, page 11–1](#).

### *Testing the Tomcat/JDK Setup*

After installing the J2SE JDK and Tomcat, you can test your environment to make sure it is set up properly.

*To run a Tomcat test:*

1. Start Tomcat by navigating to the /bin subdirectory of your Tomcat installation and double-clicking startup.bat.

**Note:** These tests may fail if you start up Tomcat from the Windows Start menu or as a Windows service. Use startup.bat to start Tomcat.

2. Open a browser and type the following URL:

`http://localhost:8080/`

You should see the default Tomcat index.html page.

3. Click **JSP Examples**.

You should see a page of examples.

**Note:** If you do not see the main Tomcat page or get a 404 error when you click **JSP Examples**, it means that Tomcat is not installed correctly.

4. Click the **Execute** link next to any of the examples.

## Testing the WDK Setup

After installing WDK, you can make sure it is set up properly by accessing some of the JSP pages.

To run the WDK-specific tests:

1. If it is not already running, start Tomcat by navigating to the /bin subdirectory of your Tomcat installation and double-clicking startup.bat.

2. In the browser type the following URL.:

```
http://localhost:8080/wdk5/wdk/samples/dumpRequest.jsp
```

where **wdk5** is the name of the virtual directory that you provided when you installed WDK. (You can find the name of the virtual directory in the webapps directory in Tomcat.)

You should see a page that lists information about Method, Url, Session, and Request Parameters.

3. Type the following URL:

```
http://localhost:8080/wdk5/wdk/samples/menuZoo.jsp
```

You should see a page that displays test text menus. If you click the Standard Menu or Cascading Menu Bars you will see drop-down boxes.

4. Type the following URL:

```
http://localhost:8080/wdk5/wdk/samples/sessionZoo.jsp
```

5. Enter a valid Docbase user name, password, and Docbase name, then click **Create Connection**.

You should see the Docbase listed in the **All Connected Docbases** section of the page, and the Status message line should start with "Successfully connected to docbase **Docbase\_name**."

6. Type the following URL:

```
http://localhost:8080/wdk5/wdk/samples/treeTest.jsp
```

You should see a page that displays the title **Tree Control Test**, which consists of a tree of generic hierarchical nodes.

## Configuration Tutorials

This chapter will demonstrate how to build a WDK application, including configuration and customization. This chapter will cover the following topics:

- [Tutorial 1: Creating a Start Page for the Application, page 10– 1](#)
- [Tutorial 2: Configuring the Display of Attributes, page 10– 2](#)
- [Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages, page 10– 8](#)
- [Tutorial 4: Adding a Logout Link, page 10–15](#)

### *Tutorial 1: Creating a Start Page for the Application*

Most Web applications are accessed through an `index.html` or `default.html` page. This is the starting point of your application. This tutorial does not require an index page, but it is a best practice to have one.

In this file, you will define a welcome page with a JavaScript function that points to the drilldown component. The drilldown component is a navigation component that will be the starting point of the new application. If the user has not logged in to the application, then the login component will be invoked from the drilldown component, and the user will be prompted to log in.

**Note:** Any component that requires a session will launch the login dialog if the user does not have a session. You do not need to call the login component explicitly.

Use the following procedure to create an `index.html` start page for your application.

*To create a start page:*

1. Create a new file called `index.html` in the ***tomcat\_home\_directory***/`webapps/wdk5` directory, where ***tomcat\_home*** is the directory in which Tomcat is installed, and ***wdk5*** is the virtual directory into which you installed WDK.
2. Add the following text:

```
<head>
<script>
function redirect()
{
    var strPath = window.location.pathname;
    var nIndex1 = strPath.indexOf("/");
    var nIndex2 = strPath.indexOf("/", nIndex1 + 1);
    var strVirtualDir = strPath.substring(nIndex1 + 1, nIndex2);
```

```
        window.location.replace("/" + strVirtualDir +
            "/component/drilldown?Reload=" +
            new Date().getTime());
    }
</script>
</head>
```

This JavaScript defines a function called “redirect,” which gets the root URL for the Web application, adds to it the drilldown component URL, and adds an argument of the current date and time, which will force a reload of the page every time it is accessed.

3. Add the following <body> tag:

```
<body onload='redirect() '>
</body>
```

This tag contains a call to the JavaScript function that you just created

4. Click File→Save.
5. Test your page by starting a browser and entering the following address:

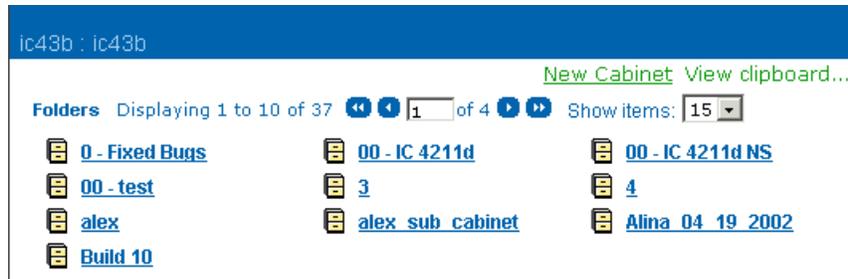
```
http://localhost:8080/wdk5
```

The address should immediately change to

```
http://localhost:8080/wdk5/component/drilldown
```

If you have not previously logged in, the login page will be displayed. After log in, you see the drilldown page, shown in the following figure.

Figure 10-1. Post-Login Drilldown Page

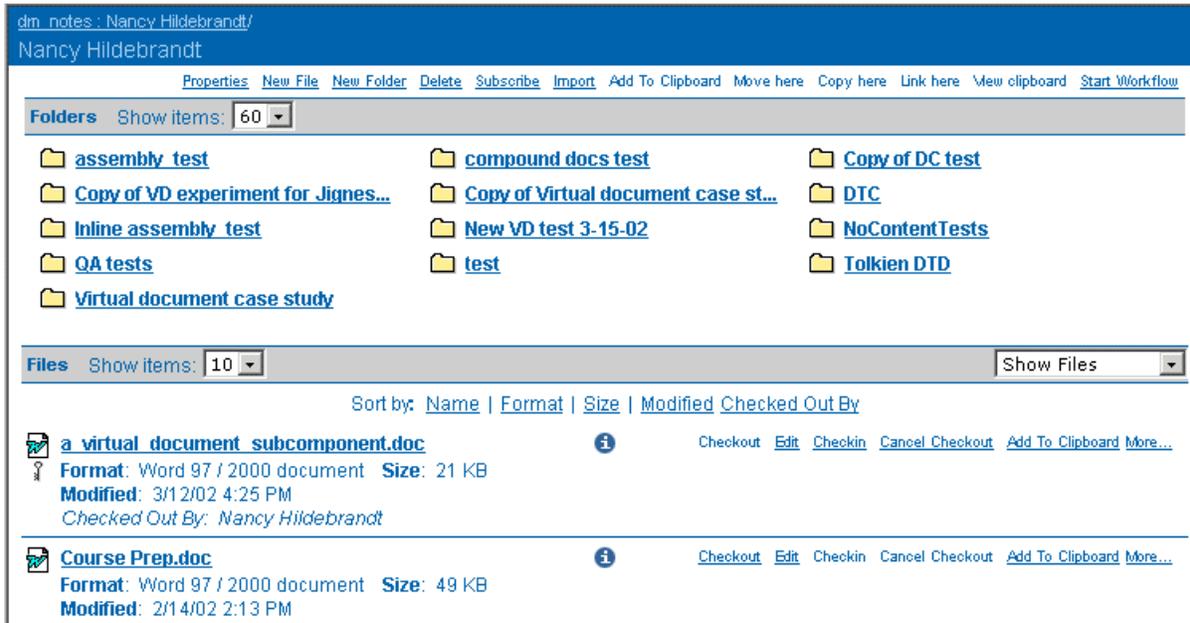


## Tutorial 2: Configuring the Display of Attributes

This tutorial demonstrates how to configure the attributes component, which controls how properties of a Docbase object (such as a document) are displayed in the application.

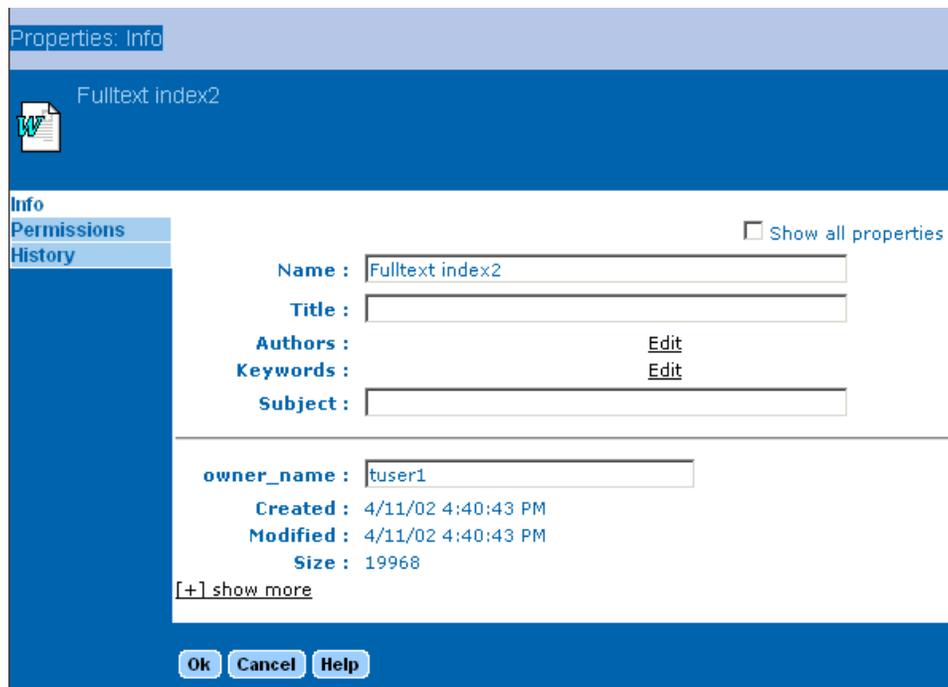
The following figure is typical of a page that displays the contents of a cabinet or a folder:

Figure 10-2. Cabinet or Folder Contents Page



If you click the Info icon (i) for a particular document, you see a short list of its properties (attributes), as in the following figure.

Figure 10-3. Standard dm\_document Properties



The **[+]show more** link at the bottom of the page will display some additional properties. When you click this link and the new properties are displayed, a **[-]hide more** link

appears. The **Show all properties** control at the top of the page will display the entire set of attributes that are stored for documents of type `dm_document`.

Suppose that you want to customize the **Info** page for `dm_document` objects (the default object type for documents) in the following ways:

- Remove the Authors and Keywords properties from the display
- Switch the order of the Title and Subject properties
- Add the existing `owner_permit` (which displays the permissions for the owner of the document) and `world_permit` properties (which displays world permissions on that document) to the short list on the **Info** page, after the `owner_name` property.

## Overview of the Configuration

The customization requires the following tasks:

- Create the appropriate file structure in the custom layer.
- Extend the base component's `dm_document` definition by customizing the appropriate XML definition file.
- Customize a JSP page to change which attributes are displayed and how they are ordered.
- Tell the application where to find the new JSP page.

## Creating the Custom Layer

The custom layer is structured as in [Table 8-1, page 8-3](#). Use the following procedure to create the directory structure and copy the XML definition file and the JSP page to the custom layer.

*To create the custom layer:*

1. Copy the following file:

```
wdk5/webcomponent/config/library/attributes/attributes_
dm_document_component.xml
```

to the following location:

```
wdk5/custom/config
```

This creates the custom layer for the XML definition file for `dm_document` attributes.

2. Copy the following file:

```
wdk5/webcomponent/library/attributes/attributes_dm_document.jsp
```

to the following location, with a new file name:

```
wdk5/custom/attributes
```

(Create the directory structure if it does not already exist.)

3. Rename the new `attributes_dm_document.jsp` in your custom layer to the following:

```
custom_attributes_dm_document.jsp
```

**Note:** You can use any file name for this JSP page, as long as you reference it in the XML definition file.

## Configuring the XML Definition File for Attributes

You customize the XML definition file by extending the definition for `dm_document` attributes and then adding a pointer to your customized JSP page, using the following procedure.

To configure the attributes definition file:

1. Open the following file:

```
wdk5/custom/config/
attributes_dm_document_component.xml
```

2. Change the `<component>` element as follows

```
<component id="attributes" extends="attributes:webcomponent/config/
library/attributes/attributes_dm_document_component.xml">
```

This extends the definition for the attributes component as defined in the default `attributes_dm_document_component.xml`.

3. Change the `<pages><start>` element as follows:

```
<pages>
<start>
/custom/attributes/custom_attributes_dm_document.jsp
</start>
</pages>
```

This points to the new JSP page that you created.

4. Save the file.

This file in the custom layer will override the `<component>` element for `dm_document` that exists in the default directory.

## Testing Access to the Custom Layer

Although you have not yet customized the JSP page, you can test that the custom **Info** page is being accessed correctly in the custom layer.

To test access to the JSP page in the custom layer:

1. Refresh the configuration service by navigating to `http://localhost:8080/wdk5/wdk/refresh.jsp`.

**Tip** Save time navigating by keeping another browser window open at the `refresh.jsp` page. Every time you need to reload the configuration service, you can refresh that browser page by pressing the `F5` function key.

2. Open a browser and go to `http://localhost:8080/wdk5/`.

Make sure the default drilldown page displays, as in [Figure 10-1, page 10-2](#).

**Note:** If you have not logged in yet or your session has timed out, you will be presented with the login page before the drilldown page appears.

3. Navigate to a page displaying a list of documents and click the **Info** icon. You should see a default **Info** page.

**Note:** The initial startup for the application involves initialization and compilation. Subsequent HTTP requests to a URL will load significantly faster. To optimize your application performance, precompile your JSP pages (using a compiler appropriate for Tomcat 4.0).

## Removing Properties from Display

The following procedure shows how to customize the JSP page by removing the Keywords and Authors attributes.

*To remove attributes from the JSP page:*

1. Open the following file:  
`wdk5/custom/attributes/custom_attributes_dm_document.jsp`
2. Remove the following elements from the default attributes section::  

```
<dmfx:docbaseattribute object="obj" attribute="authors" pre=
"<tr><th align=\"right\" valign=\"top\">" coll="</th><th valign=
\"top\">: &nbsp;</th><td>"/>
<dmfx:docbaseattribute object="obj" attribute="keywords" pre=
"<tr><th align=\"right\" valign=\"top\">" coll="</th><th valign=
\"top\">: &nbsp;</th><td>"/>
```
3. Save the file.
4. Refresh the configuration and then view the **Info** page for a document in a browser. The Keywords and Authors properties should no longer be displayed.

## Rearranging Properties in the Display

Use the following procedure to reverse the order of display of the Title and Subject properties on the **Info** page.

*To reorder the display of properties:*

1. Open the following file:  
`wdk5/custom/attributes/custom_attributes_dm_document.jsp`
2. Locate the tag for the Subject property:  

```
<dmfx:docbaseattribute object="obj" attribute="subject" size="48"
coll="</th><th>: &nbsp;</th><td>"/>
```
3. Select all of the text for this tag.

- Cut the text and paste it above the tag for the Title attribute. The markup for the subject and title attributes will now appear as follows:

```
<dmfx:docbaseattribute object="obj" attribute="subject" size="48"
 coll="</th><th>: &nbsp;</th><td>"/>
<dmfx:docbaseattribute object="obj" attribute="title" size="48"
 coll="</th><th>: &nbsp;</th><td>"/>
```

**Note:** The docbaseattribute tag adds the HTML tags <tr><th> to the output

- Save the file.
- Refresh the configuration and then view the **Info** page for a document in a browser. The first three properties should be displayed in the order Name, Subject, Title.

## Adding Existing Docbase Attributes to the Main Info Page

The following procedure shows how to add the existing attributes Owner Permission (owner\_permit) and World Permissions (world\_permit) to the short list displayed on the main **Info** page.

To add an existing attribute to the Info page:

- Open the following file:

```
wdk5/custom/attributes/custom_attributes_dm_document.jsp
```

- Locate the tag for the owner\_name attribute:

```
<dmfx:docbaseattribute object="obj" attribute="owner_name"
 coll="</th><th>: &nbsp;</th><td>"/>
```

- Add the tag for the owner\_permit attribute on the following line:

```
<dmfx:docbaseattribute object="obj" attribute="owner_permit"
 coll="</th><th>: &nbsp;</th><td>"/>
```

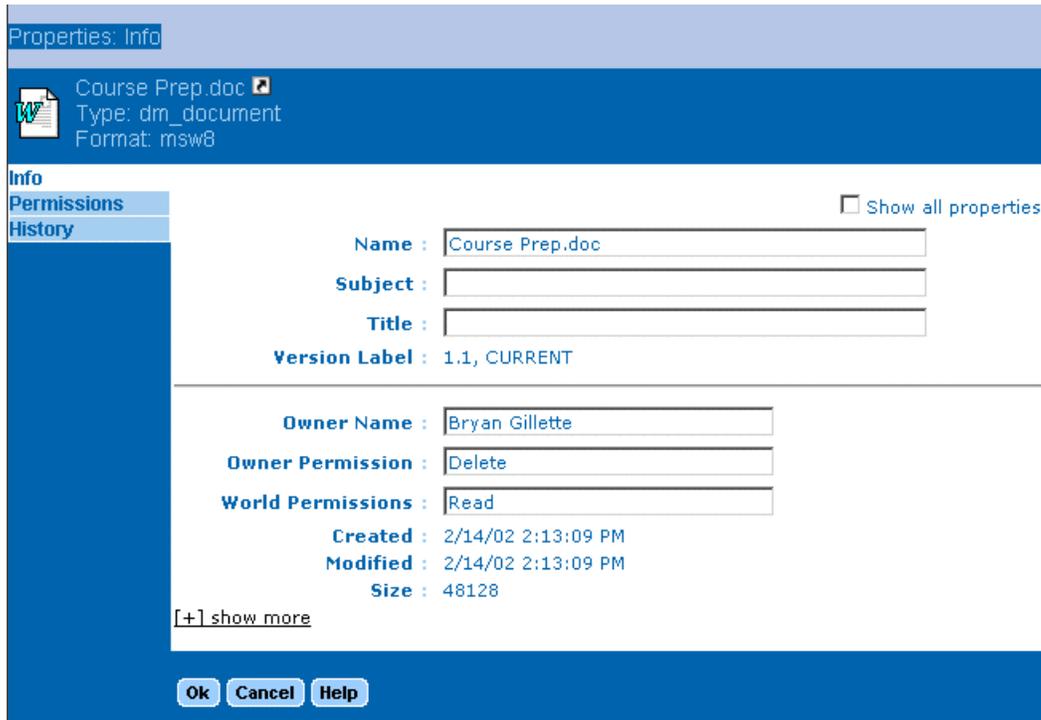
**Tip** Instead of typing, you can copy the previous line and change the name of the attribute from owner\_name to owner\_permit.

- Add the tag for the world\_permit attribute on the following line:

```
<dmfx:docbaseattribute object="obj" attribute="world_permit"
 coll="</th><th>: &nbsp;</th><td>"/>
```

- Save the file.
- Follow the previous procedure to access the **Info** page for a document in a browser. The **Info** page should now appear with the additional properties owner\_name, owner\_permit, and World Permissions, as shown in the following figure.

Figure 10–4. Added Properties



### Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages

Suppose that you want to create a custom object type for SOP documents, with four custom attributes:

- Affected Departments
- Approving Department
- Approval Date
- SOP Number

You want to display these custom attributes on both the **Info** page and the **Checkin** page, and you want those properties to be editable in both places. The following figure shows an example of how the **Info** page should look.

Figure 10-5. Info Page with Custom Attributes

Properties: Info

Test SOP document   
 Type: sop\_document  
 Format: excel5book

Info  Show all properties  
 Permissions  
 History

Name :

Title :

Authors :  [Edit](#)

Keywords :  [Edit](#)

Subject :

r\_version\_label : 1.0, CURRENT

---

Affected Departments :  [Edit](#)

Approving Department :

Approval Date :  /  /   
 :  :

SOP Number :

owner\_name :

Created : 9/12/02 11:25:09 AM  
 Modified : 9/12/02 11:26:44 AM  
 Size : 31744

[+] show more

## Overview of the Customization

This customization requires the following procedures:

- Create a customized object type in your test Docbase.
- Create the proper files in the custom layer.
- Make the file modifications to display the custom attributes on the **Info** page of SOP documents.
- Make the file modifications to display the custom attributes on the **Checkin** page for SOP documents.

## Creating a Custom Object Type

To complete this tutorial, you will need to use a custom document object type in your test Docbase. You may use a custom object type that you have already created, or you can create a custom subtype of `dm_document` called `sop_document`, following the instructions in this section.

To create a custom object type, you must create a Docbase application using Documentum Developer Studio (DDS) and be able to log in to the Docbase with Superuser privileges. For more detailed information on how to build DocApps using DDS, see the manual ***Developing DocApps***.

To create the sop custom object type:

1. In DDS, create an application named sop.
2. Create a type named sop\_document as a subtype of dm\_document, with the label Standard Operating Procedure.
3. Add four attributes to this type, as shown in the following table.

Table 10–1. SOP Application Attributes

Attribute name	<b>Attribute value</b>	Type	Values
affected_departments	Affected Departments	String, length 32, repeating	Value Assistance: <ul style="list-style-type: none"> <li>• Consulting</li> <li>• Engineering</li> <li>• Finance</li> <li>• Marketing</li> <li>• Sales</li> <li>• Technical Support</li> <li>• Quality Assurance</li> </ul>
approval_department	Approving Department	String, length 32	Value Assistance: <ul style="list-style-type: none"> <li>• Consulting</li> <li>• Engineering</li> <li>• Finance</li> <li>• Marketing</li> <li>• Sales</li> <li>• Technical Support</li> <li>• Quality Assurance</li> </ul>
effective_date	Approval Date	Time	N/A
sop_number	SOP Number	String, length 32	Input Mask: SOP-#####

**Note:** Value assistance allows the user to select from predefined values for attributes.

4. Check in the application and the sop type you have created

- Restart the Docbase.

**Note:** If you wanted your customizations to apply to all Docbases, you would copy this application to all your Docbases and test as a final step before moving the customization to your production server.

## Creating the Custom Layer Files

Adding custom attributes to the **Info** and **Checkin** pages involves making modifications to the XML definition files for both pages and creating new JSP pages specifically for the sop\_document custom object type. The following procedure sets up these files in the custom layer.

To create the custom layer:

- Copy the following file:

`wdk5/webcomponent/config/library/checkin/checkin_component.xml`

to the following location:

`wdk5/custom/config`

- Copy the following file:

`wdk5/webcomponent/library/attributes/attributes_dm_document.jsp`

to the following location, with a new file name:

`wdk5/custom/attributes/attributes_sop_document.jsp`

(Create the directory structure if it does not already exist.)

- Copy the following file:

`wdk5/webcomponent/library/checkin/checkin.jsp`

to the following location, with a new file name:

`wdk5/custom/checkin/sop_checkin.jsp`

(Create the directory structure if it does not already exist.)

## Modifying the Info Page

In the previous tutorial, you created an XML definition file that extended the definition of attributes for object type dm\_document. You will now modify the file to include documents of type sop. You will then create a new JSP page for the sop object type.

To add the sop type to the attributes definition file:

- Open **wdk5**/custom/config/attributes\_dm\_document\_component.xml.

**Note:** If you did not complete the previous tutorial, copy attributes\_dm\_document\_component.xml from **wdk5**/webcomponent/config/library/attributes to **wdk5**/custom/config.

- Copy the existing <scope> element and rename the copy to type="sop\_document", or type in the following bold text:

```

<config>
...
<scope type="dm_document">
  <component id="attributes" extends=
    "attributes:webcomponent/config/library/attributes/
    attributes_dm_document_component.xml">
...
  </component>
</scope>
<scope type='sop_document'>
  <component id="attributes" extends=
    "attributes:webcomponent/config/library/attributes/
    attributes_dm_document_component.xml">

    <!-- Description (not NLS'd) -->
    <desc>
      Attribute component: Provides UI and behaviour for
      showing and editing a document's attributes.
    </desc>

    <!-- Component Layouts -->
    <pages>
      <start>/custom/attributes/
      attributes_sop_document.jsp</start>
    </pages>

    <helpcontextid>attributesdmdocument</helpcontextid>
  </component>
</scope>
</config>

```

(The first <scope> element in the file is the customization from the previous tutorial.)

3. Save and close the file.

To modify the JSP file for the Info page:

1. Open **wdk5**/custom/attributes/attributes\_sop\_document.jsp.
2. Locate the following line:

```

<dmfx:docbaseattribute object="obj" attribute="owner_name"
  coll="</th><th>:&nbsp;</th><td>"/>

```

This is the line that displays the owner\_name attribute.

3. Directly above this line, add the following four lines (identical to the owner\_name line except for the name of the attribute):

```

<dmfx:docbaseattribute object="obj" attribute=
  "affected_departments" coll="</th><th>:&nbsp;</th><td>"/>
<dmfx:docbaseattribute object="obj" attribute=
  "approval_department" coll="</th><th>:&nbsp;</th><td>"/>
<dmfx:docbaseattribute object="obj" attribute=
  "effective_date" coll="</th><th>:&nbsp;</th><td>"/>
<dmfx:docbaseattribute object="obj" attribute=
  "sop_number" coll="</th><th>:&nbsp;</th><td>"/>

```

4. Save and close the file.
5. Refresh the configuration service by navigating to <http://localhost:8080/wdk5/wdk/refresh.jsp>.
6. Navigate to the **Info** page of a document of type sop\_document. (If you have not yet created a document of this custom type, do so now.) You should now see the four

custom attributes displayed in addition to the regular attributes that are inherited from the `dm_document` supertype, as in [Figure 10-5, page 10-9](#).

You can experiment with editing the properties for this document. Because Affected Departments and Approving Department were set up with value assistance, they have an **Edit** link, which opens another window of value choices. However, their behavior is slightly different. Since Affected Departments is a repeating attribute, you can select one or more of the pre-defined choices. For Approving Department, you can select one of the predefined values or define your own new value.

**Note:** For the SOP Number attribute, you may receive the following error message:

```
"sop_number" requires an input mask: SOP-LLLLL
```

To resolve this, follow the prescribed input format in entering an SOP number (for example, SOP-00019).

To make sure that this display only applies to documents of type `sop_document`, view the **Info** page for a document of type `dm_document` and verify that it does not display these custom properties.

## Modifying the Checkin Page

This section shows how to modify the **Checkin** page for the `sop_document` custom object type, by following these procedures:

- Extend the checkin component for documents of type `sop_document` by [modifying the XML definition file](#) in the custom layer.
- [Add the custom attributes to the custom checkin JSP](#) defined for SOP documents.

To modify the checkin XML definition file:

1. Open `wdk5/custom/config/checkin_component.xml`.
2. Change the `<scope type="dm_sysobject">` tag as follows:
 

```
<scope type="sop_document">
```
3. Change the `<component>` element to extend the checkin component, as follows:
 

```
<component id="checkin" extends="checkin:webcomponent/
  config/library/checkin/checkin_component.xml">
```
4. Change the `<pages><ui>` element as follows:
 

```
<pages>
  <ui>/custom/checkin/sop_checkin.jsp/ui>
</pages>
```
5. Save the file.

To add custom attributes to the `sop_checkin.jsp` page:

1. Open `wdk5/custom/checkin/sop_checkin.jsp`.
2. Locate the following lines:
 

```
<tr>
  <td><dmfx:docbaseattribute object="object" attribute=
    "a_content_type" readonly="true"/></td>
```

```
</tr>
```

**Note:** When attributes are defined as editable in the Docbase but you want to display them as read-only, you can set the `readonly="true"` attribute, as it appears in this line.

3. Add the following attributes in table rows, after the table row you located in the previous step:

```
<tr>
  <td><dmfx:docbaseattribute object="object" attribute=
    "affected_departments"/></td>
</tr>
<tr>
  <td><dmfx:docbaseattribute object="object" attribute=
    "approval_department"/></td>
</tr>
<tr>
  <td><dmfx:docbaseattribute object="object" attribute=
    "effective_date"/></td>
</tr>

<tr>
  <td><dmfx:docbaseattribute object="object" attribute=
    "sop_number"/></td>
</tr>
```

4. Save and close the file.
5. Refresh the configuration service by navigating to <http://localhost:8080/wdk5/wdk/refresh.jsp>.
6. Navigate to the sop document. Check it out and check it back in again. On the **Checkin** page, you should see the additional four custom properties, and they should be editable, as in the following figure.

Figure 10–6. Checkin Page with Custom Attributes

Check In Test SOP document

**Test SOP document**

**Version** 1.1, CURRENT

**Type** sop\_document

**Format** excel5book

**Affected Departments** [Edit](#)

**Approving Department**

**Approval Date**  /  /   :  
 :

**SOP Number**

**Save as:**  1.1 (same version)  
 1.2 (minor version)  
 2.0 (major version)

**Version label:**

**Description:**

**Format:** Excel workbook 5.0 (MacOS), 5.0-7.0 (Windows)

**Full Text Indexed:**

[\[+\] Show options](#)

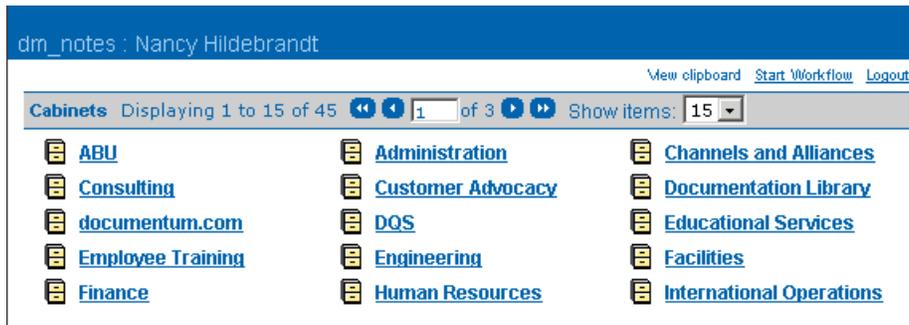
**Ok Cancel Help**

For comparison, check out and check in a document of type dm\_document and check to make sure these custom properties do not appear on the **Checkin** page.

## Tutorial 4: Adding a Logout Link

The following tutorial shows you how to add a Logout link to the drilldown page. When you are finished, the Logout link should appear as in the following figure.

Figure 10-7. Drilldown Page with Logout Link



This tutorial involves the following procedures:

- Creating the files in the custom layer
- Modifying the drilldown component XML definition file
- Modifying the JSP pages that handle drilldown
- Extending the properties file that contains the resource strings for the drilldown component.

## Creating the Custom Layer Files

To create the files for the customization:

1. Copy the following file:

```
wdk5/webcomponent/config/navigation/drilldown/  
drilldown_component.xml
```

to the following location:

```
wdk5/custom/config
```

2. Copy the following file:

```
wdk5/webcomponent/navigation/drilldown/drilldown.jsp
```

to the following location, with a new file name:

```
wdk5/custom/navigation/drilldown/custom_drilldown.jsp
```

3. Copy the following file:

```
wdk5/webcomponent/navigation/drilldown/drilldown_body.jsp
```

to the following location, with a new file name:

```
wdk5/custom/navigation/drilldown/custom_drilldown_body.jsp
```

4. Create the following directory structure:

```
wdk5/custom/strings/com/documentum/custom
```

## Modifying the XML Definition File

To modify the XML definition file:

1. Open the following file:

```
wdk5/custom/config/drilldown_component.xml
```

2. Change the <component> element as follows:

```
<component id="drilldown" extends="drilldown:webcomponent/
  config/navigation/drilldown/drilldown_component.xml">
```

This will extend the drilldown component that is defined in the default definition file.

3. Change the <pages><start> element as follows:

```
<pages>
  <start>
    /custom/navigation/drilldown/custom_drilldown.jsp
  </start>
</pages>
```

This points to one of the new JSP files that you created.

4. Change the <nlsbundle> element as follows:

```
<nlsbundle>com.documentum.custom.MyDrillDownNlsProp</nlsbundle>
```

You have already created the directory structure (**wdk5**/custom/strings/com/documentum/custom), and you will create the custom file (MyDrillDownNlsProp.properties) later in this tutorial. This custom resource file is necessary to define the external resource string for the Logout link. When the <nlsbundle> element is processed at runtime, the **wdk5**/custom/strings directory is searched first, followed by the **wdk5**/webcomponent/strings directory. The resource file that you replaced in the <nlsbundle> element, in this case DrillDownNlsProp, will be included in MyDrillDownNlsProp.

5. Save the file and close it.

## Modifying the Drilldown JSP

There are two JSP pages involved with the drilldown that is displayed after login: drilldown.jsp calls drilldown\_body.jsp. The following procedure changes custom\_drilldown.jsp to point to custom\_drilldown\_body.jsp.

To modify custom\_drilldown.jsp:

1. Open the following file

```
wdk5/custom/navigation/drilldown/custom_drilldown.jsp
```

2. Scroll down to the <dmf:form> tag near the end of the file and change the name of the JSP file so it appears as follows:

```
<dmf:form>
  <!-- include the body of the component -->
  <%@ include file='/custom/navigation/drilldown/
    custom_drilldown_body.jsp' %>
```

```
</dmf:form>
```

3. Save the file and close it.

### *Modifying custom\_drilldown\_body.jsp*

The `drilldown_body.jsp` file contains the action links such as View Clipboard and Start Workflow Template, which appear under the breadcrumb on the default drilldown page. Action links are a type of action control. Action controls invoke the action service and are different from standard controls in that they are automatically hidden or disabled if the associated action is not resolved or executable. The Logout link does not require hiding or disabling, and so we will use a non-input control.

To modify `custom_drilldown_body.jsp`:

1. Open the following file

```
wdk5/custom/navigation/drilldown/custom_drilldown_body.jsp
```

2. Locate the following lines:

```
<dmfx:actionlink cssclass='actions' name='startworkflownotemplate'  
nlsid='MSG_START_WORKFLOW_FULL' action='startworkflownotemplate'>  
  <dmf:argument name='startworkflowId' value=  
    'startworkflowdrilldown' />  
</dmfx:actionlink>
```

3. Immediately below `</dmfx:actionlink>`, add the following line:

```
<dmfx:actionlink cssclass='actions' name='logout' nlsid='MSG_LOGOUT'  
  action="logout"/>
```

The `dmfx:` prefix specifies that this is an action control, and the type is `actionlink`. The attribute `cssclass` specifies the name of the CSS class that will be applied to this action. The attribute `name` sets the name for the control. Named controls are cached on the server. The `'MSG_'` prefix for the `nlsid` attribute value indicates a resource string, which comes from the properties file that is used for this page. When you modified the XML definition file, you changed the `<nlsbundle>` component to point to a custom `.properties` file, which you will create in the next procedure in order to define this string. The `action` attribute is the event that will be triggered. For more information about the `actionlink` control, see the ***Documentum Web Development Kit™ Reference***.

4. Save the file and close it.

### *Creating a Custom Resource File*

The `<nlsbundle>` element in the default `drilldown_component.xml` definition file points to `com.documentum.webcomponent.navigation.drilldown.DrillDownNlsProp`, which means the `DrillDownNlsProp.properties` file in a subdirectory of ***wdk5/webcomponent/strings***. The resource string you specified in the `actionlink` control, namely `MSG_LOGOUT`, is defined in some resource files, but not in `DrillDownNlsProp`. This means that you must create a custom properties file for this page that includes the

DrillDownNlsProp.properties file. (You have already pointed to the location of this custom resource file in the XML definition file.)

To create a custom resource file:

1. Navigate to the following directory:  
`wdk5/custom/strings/com/documentum/custom`
2. Create a new text file and name it `MyDrillDownNlsProp.properties`.
3. Add the following two lines:

```
NLS_INCLUDES=com.documentum.webcomponent.navigation.  
  drilldown.DrillDownNlsProp  
MSG_LOGOUT=Logout
```

These lines include the `DrillDownNlsProp.properties` file in the default location and defines one additional resource string, the one referenced in the `nlid` attribute for the `actionlink` control that you added to `custom_drilldown_body.jsp` for the Logout link.

## Testing the Logout Link

1. Refresh the configuration service by navigating to `http://localhost:8080/wdk5/wdk/refresh.jsp`.
2. After you log in, the drilldown page should appear, with the new **Logout** link in the upper right-hand corner.

**Note:** If you make changes to a JSP page that are not reflected in the display, you may need to clear the cached files in `tomcat_home/work/localhost/wdk5`.



## *Customization Tutorial*

This chapter provides information on setting up an IDE for customizations and provides a customization tutorial. It contains the following sections:

- [Setting up the WDK Project in Sun ONE Studio, page 11- 1](#)
- [Tutorial 5: Converting from a Dialog to a Wizard Display, page 11- 2](#)

### *Setting up the WDK Project in Sun ONE Studio*

When you create a custom class, you compile it using an ID such as Sun One Studio (Forte for Java) or an external compiler. This section contains information on creating a project in Sun One Studio.

This manual uses Sun One Studio only for the customization tutorials, but you can use it for the configuration tutorials as well, if you prefer.

### *Creating a Project for WDK*

*To create a new project in Sun ONE Studio:*

1. Open Sun ONE Studio (Forte for Java).  
**Note:** The first time you run Sun ONE Studio , you are prompted to select a directory where the individual files will be stored (projects, samples, and IDE settings). This directory can be anywhere on your file system and does not need to be the same as your Web application folder.
2. Choose Project→Project Manager.
3. Click New.
4. Create a name for your project, such as wdk5, and click OK.

### *Mounting the WDK Directories*

Mounting the WDK Java archives effectively sets the classpath for your project.

To mount the WDK directories:

1. In the Explorer window in Forte for Java , right-click Filesystems and choose **Mount**→**Local Directory**.
2. Navigate to the Tomcat webapps directory (for example, C:/Program Files/Apache Tomcat 4.0/webapps), highlight the directory in which you installed WDK (for example, wdk5), then click **Finish**.

**Note:** If you see a message that an alternate view is available, you can safely ignore it.

## Stopping the Internal Tomcat Server

If you are running Forte for Java , stop the internal Tomcat server instance before starting an external Tomcat 4 server, since both are configured by default to run at port localhost:8080. Use the following procedure.

To stop the Forte for Java IDE server instance:

1. In Forte for Java Explorer, click the Runtime tab in the Explorer frame.
2. Expand the tree to Server Registry→Installed Servers→Tomcat 4.0.
3. Right-click on Internal and select **Stop Server** from the context menu.



**Caution** Do not start an internal or external Tomcat server from within the IDE using the default command arguments. The default start script will not use the environment settings required by WDK.

You can install an external Tomcat 4 server in the Forte for Java server registry. Make sure you do not select the Full IDE integration mode. Then modify the the default External Execution Process so that it uses the start script that was modified by the WDK installer. To do this, right-click on the external Tomcat 4 server and select Properties. Click on External Execution Process and enter the path to the batch file. For example:

```
C:\Program Files\Apache Tomcat 4.0\bin\startup.bat
```

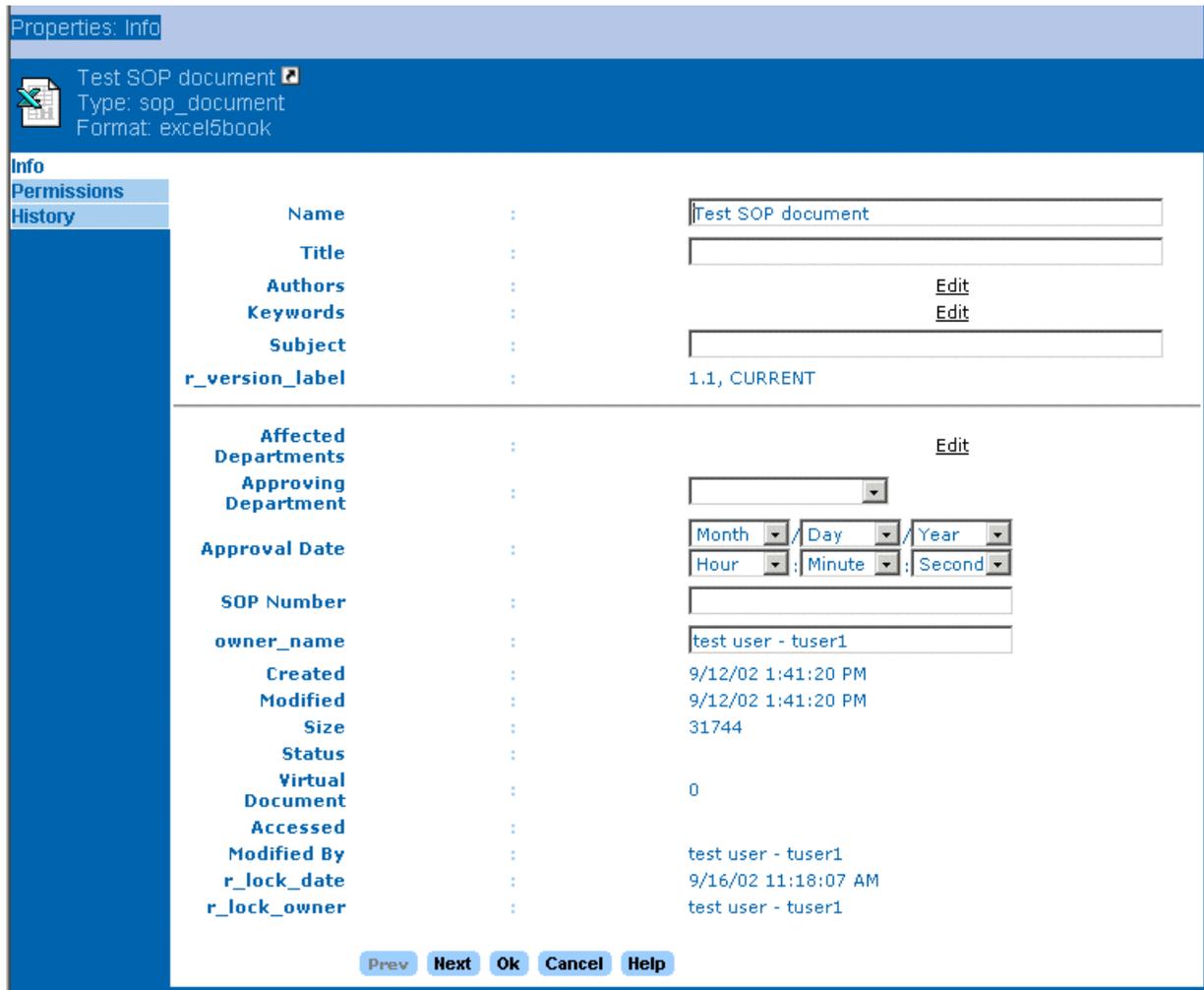
## Tutorial 5: Converting from a Dialog to a Wizard Display

In [Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages, page 10– 8](#) , you added custom attributes to the **Info** page displayed for documents of type sop\_document. In this tutorial, you will perform a customization and configuration that will change two aspects of the display:

- The **Show all properties** checkbox and **[+]show more** link will not be available.
- At the bottom of the page, **Previous** and **Next** buttons will be added to the **OK**, **Cancel**, and **Help** buttons.

As a result, the **Info** page will look similar to the one shown in the following figure (compare with [Figure 10–5, page 10– 9](#)).

Figure 11-1. Customized Info Page



The properties component is a container that embeds the attributes, permissions, and history components on the same page (see the previous figure, which shows the **Info**, **Permissions**, and **History** tabs). The properties component, defined in `properties_component.xml`, extends `property-sheet-container_component.xml`, which in turn extends `wizard-container.xml` and points to `properties.jsp`, which contains OK, Cancel, and Help buttons. You will modify this JSP file to add the Previous and Next buttons.

In addition, the display of the Previous and Next buttons is controlled by the `onPrevPage()` and `onNextPage()` eventhandlers, inherited from the `wizard-container` component class, where they are assigned the value "false", which causes the buttons not to be displayed. To display these buttons, we will extend the attributes component class, adding the methods `hasPrev()`, `hasNext()`, `onPrev()`, and `onNext()`. This will result in the buttons being visible but enabled or disabled based on the logic in the `hasPrev()`, and `hasNext()` methods.

The following procedures are required to make these changes:

- Modify `attributes_dm_document_component.xml` to point to the custom class file you will create.
- Modify `properties_component.xml` to point to a custom JSP page (`sop_properties.jsp`), which will add the **Previous** and **Next** button to the **Info** page.
- Remove the **Show all properties** checkbox and the **[+] show more** link from the **Info** page by modifying `attributes_sop_document.jsp`.
- Extend the `Attributes` class by defining event handlers for the **Previous** and **Next** buttons.

**Note:** This tutorial assumes that you have completed the configuration work in [Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages, page 10– 8](#).

## Creating the Custom Layer Files

Several of the XML and JSP files needed for this tutorial have already been created in [Tutorial 3: Adding Custom Attributes to the Info and Checkin Pages, page 10– 8](#).

To create the files for the customization:

1. Copy the following file:

```
wdk5/webcomponent/config/library/properties/properties_component.xml
```

to the following location:

```
wdk5/custom/config
```

2. Copy the following file:

```
wdk5/webcomponent/library/properties/properties.jsp
```

to the following location, with a new file name:

```
wdk5/custom/properties/sop_properties.jsp
```

## Modifying the Attributes Component Definition

This section shows how to modify the XML definition files for the attributes and the properties components.

To modify the attributes XML definition file:

1. Open the following file:

```
wdk5/custom/config/attributes_dm_document_component.xml
```

2. Locate the following lines:

```
<pages>  
  <start>/custom/attributes/attributes_sop_document.jsp</start>  
</pages>
```

These lines are nested under the `<component>` element for `type="sop_document"`.

3. Immediately under the `</pages>` close tag, add the following line:

```
<!-- Component Behavior -->  
<class>com.documentum.custom.SopAttributes</class>
```

This points to the class file that you are going to create.

4. Save and close the file.

To modify the properties XML definition file:

1. Open the following file:

```
wdk5/custom/config/properties_component.xml
```

2. Change the properties component start tag to the following:

```
<component id="properties" extends="properties:webcomponent/config/
  library/properties/properties_component.xml">
```

3. Change the <page> element as follows, to point to a new JSP page that you will modify:

```
<pages>
  <start>/custom/properties/sop_properties.jsp</start>
</pages>
```

4. Save and close the file.

## Modifying the JSP Pages

You must modify the attributes JSP to remove the **Show all properties** checkbox and the **{+} show more** link from the display of attributes on the **Info** page. All of the attributes that are normally displayed on the second page when you click the **{+} show more** link will now be displayed on the main **Info** page. You must modify the properties JSP file to add the **Previous** and **Next** buttons.

To modify the JSP attributes page:

1. Open the following file:

```
wdk5/custom/attributes/attributes_sop_document.jsp
```

2. Find and remove the following lines:

```
<!-- show all attributes checkbox -->
<tr><td colspan="3" align=right>
  <dmf:checkbox name='show_all' onclick='onShowAllClicked' nlsid=
  'MSG_SHOW_ALL_PROPERTIES' />
</td></tr>
```

This removes the **Show all properties** checkbox.

3. Remove the following lines:

```
<!-- secondary attributes -->
<tr><td colspan="4"><dmf:link name="moreAttributesLink" nlsid=
  "MSG_SHOW_MORE" onclick="onShowHideMoreClicked"/></td></tr>
<dmf:panel name="moreAttributesPanel" visible="false">
```

This removes the the **{+} show more** link.

4. Skip down a few lines a remove the following close tag:

```
</dmf:panel>
```

Now all of the attributes are grouped together for display on a single page.

5. Save and close the file.

To modify the JSP container for properties:

1. Open the following file:

```
wdk5/custom/properties/sop_properties.jsp
```

2. Locate the following lines:

```
<td>
  <dmf:button name='ok' cssclass="buttonLink"
    nlsid='MSG_OK' onclick='onOk'
    height='16' imagefolder='images/dialogbutton' />
</td>
```

3. Immediately above these lines, insert the following lines:

```
<td>
  <dmf:button name='prev' cssclass="buttonLink" nlsid='MSG_PREV'
    onclick='onPrev' height='16' imagefolder='images/dialogbutton' />
</td>
<td>
  <dmf:button name='next' cssclass='buttonLink' nlsid='MSG_NEXT'
    onclick='onNext' height='16' imagefolder='images/dialogbutton' />
</td>
```

This adds the **Previous** and **Next** buttons.

4. Save and close the file.

## Extending the Attribute Class

The following procedure extends the attributes class to assign values to the event handlers for the Previous and Next buttons that will cause them to be displayed when appropriate.

To create the component class:

1. In Sun One Studio (Forte for Java), create a new Java class by right-clicking on the directory WEB-INF/Classes/com/documentum/custom and selecting New -> Classes -> Empty. (If you did not complete the previous customization tutorial, you will need to create this directory structure.)
2. Name the object SopAttributes and click Finish..
3. In the Source Editor window, add the following lines:

```
package com.documentum.custom;
//import original Attributes class so methods will be inherited
import com.documentum.webcomponent.library.attributes.Attributes;
//import required classes needed for layout tags and connection
// class java.lang.sql
import com.documentum.web.common.ArgumentList;
//Define the Java class
public class SopAttributes extends Attributes {
    public boolean hasNextPage() {
        //Call event handler to determine if there is a next page
        return (getComponentPage().equals("all") == false);
    }
    public boolean hasPrevPage() {
```

```

        //Call event handler to determine if there is a previous page
        return (getComponentPage().equals("start") == false);
    }
    public boolean onPrevPage() {
        //Call an event handler when user switches to previous page
        if ( getComponentPage().equals("start") == false ) {
            setComponentPage("start");
            return true;
        }
        else {
            return false;
        }
    }
    public boolean onNextPage() {
        //Call an event handler when user switches to next page
        if ( getComponentPage().equals("all") == false ) {
            setComponentPage("all");
            return true;
        }
        else {
            return false;
        }
    }
}

```

4. Save and compile the new class by right-clicking on the class name in the Sun One Studio (Forte) tree.

**Note:** If you are running Sun One Studio, stop the internal Tomcat server instance before starting an external Tomcat 4 server. For more information, see [Stopping the Internal Tomcat Server, page 11- 2](#).

## Testing the Customization

1. Refresh the configuration service by navigating to <http://localhost:8080/wdk5/wdk/refresh.jsp>.
2. Log in and navigate to a folder that contains a document of type `sop_document`, and click the icon to view its properties. The **Info** page should appear with a **Previous** and **Next** button. The attributes displayed on the first page should be the ones defined in `attributes_sop_document.jsp`. If you click the **Next** button, you will see the entire set of attributes (previously viewed by clicking the **Show All Attributes** checkbox). You can also verify that the **Info** page still displays in the default fashion for documents that are not of type `sop_document`.



## *Troubleshooting*

If your customization does not appear as you intended, you can try some of the procedures below to troubleshoot:

- [Did you precompile your Java class?, page 12- 1](#)
- [Did you refresh the Configuration Service?, page 12- 1](#)
- [Did you remove generated files?, page 12- 1](#)
- [Did you clear the browser cache?, page 12- 2](#)
- [Did you check the name and location of the XML definition file?, page 12- 2](#)

### *Did you precompile your Java class?*

If you extend a Java class or edit it, you should remember to compile it in Forte for Java. On the other hand, you should allow Tomcat 4 to compile your JSP pages automatically, because Forte for Java supports Tomcat 3.

### *Did you refresh the Configuration Service?*

If you make changes to an XML definition file or a JSP page, you need to restart the configuration service, either by navigating to <http://localhost:8080/wdk5/wdk/refresh.jsp> or by stopping and restarting the application server.

### *Did you remove generated files?*

If you edit existing files and the changes are not reflected in the display, you may need to remove the generated files from the Tomcat work area.

*To remove generated files from Tomcat*

1. Navigate to **tomcat\_home\_directory**/work/localhost/**wdk5**/custom.
2. Navigate to the feature you are working with (in the current tutorial, this is the viewcontainer folder).

3. Delete the .class and .java files in the folder.
4. Test the customization again.

### *Did you clear the browser cache?*

If you are having problems that involve JavaScript, you should try clearing your browser cache, as described in the following procedure.

*To clear the browser cache in Internet Explorer:*

1. In Internet Explorer, choose Tools→Internet Options.
2. In the Temporary Internet files pane, click Delete Files.
3. Click OK.

### *Did you check the name and location of the XML definition file?*

Your custom XML definition file should have the same name as an existing XML definition file, and all customized XML definition files should be located in the **tomcat\_home\_directory**/webapps/**wdk5**/custom/config directory, where **tomcat\_home\_directory** is the directory in which Tomcat is installed, and **wdk5** is the name of the virtual directory that was provided when WDK was installed.

One way to test whether this file is being recognized is to make a copy of it in the same directory, then refresh the configuration service by navigating to <http://localhost:8080/wdk5/wdk/refresh.jsp>. You should get an error message that there is a duplicate element, and the name of this copied file will be mentioned near the end of the error message. This shows that your XML definition file is being read. Delete the copy and continue troubleshooting.

If your XML definition file is being read, then you can assume that you will get a 404 error if the browser does not find the JSP file in the specified location, and a Java error if it does not find the class file in the specified location.