



Community Experience Distilled

Alfresco 3

Business Solutions

Practical implementation techniques and guidance for
delivering business solutions with Alfresco



Martin Bergljung

[PACKT] open source*
PUBLISHING community experience distilled

Alfresco 3 Business Solutions

Practical implementation techniques and guidance for
delivering business solutions with Alfresco

Martin Bergljung



BIRMINGHAM - MUMBAI

Alfresco 3 Business Solutions

Copyright © 2011 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: February 2011

Production Reference: 1030211

Published by Packt Publishing Ltd.
32 Lincoln Road
Olton
Birmingham, B27 6PA, UK.

ISBN 978-1-849513-34-0

www.packtpub.com

Cover Image by Ed Maclean (edmaclean@gmail.com)

Credits

Author

Martin Bergljung

Reviewers

Johnny Gee

Sivasundaram Umapathy

Adrián Efrén Jiménez Vega

Acquisition Editor

Steven Wilding

Development Editor

Maitreya Bhakal

Technical Editors

Arun Nadar

Namita Sahni

Aditi Suvarna

Copy Editor

Laxmi Subramanian

Editorial Team Leader

Aditya Belpathak

Project Team Leader

Lata Basantani

Project Coordinator

Leena Purkait

Proofreader

Mario Cecere

Indexers

Tejal Daruwale

Hemangini Bari

Production Coordinator

Aparna Bhagat

Cover Work

Aparna Bhagat

About the Author

Martin Bergljung is a Principal ECM Architect at Ixxus, a UK platinum Alfresco partner. He has over 20 years of experience in the IT sector, where he has worked with the Java platform since 1997.

Martin began working with Alfresco in 2007 developing an e-mail management extension for Alfresco called OpsMailmanager. In 2009 he started doing Alfresco consulting projects and has worked with customers such as Virgin Money, ITF, Unibet, and BNP Paribas.

I would like to thank Steven Wilding at Packt Publishing for suggesting the project and getting it on track.

A thanks goes also to Leena Purkait, my Project Coordinator, who was always pushing me to deliver the next chapter. My Development Editor Maitreya Bhakal was also very helpful the last couple of months by pushing me to get the chapters finished in time and in line with the final size of the book. Thank you also to the entire Packt Publishing team for working so diligently to help bring out a high quality product.

Thanks to all the book reviewers who gave me invaluable feedback during the whole project. I must also thank the talented team of developers who created the Alfresco open source product. It opens up a new way for everyone that wants to build any kind of ECM business solution.

I would like to thank Paul Samuel at Ixxus for supporting my book project.

Finally, I would like to give a special thanks to Robin Bramley for contributing source code for the chapter about mobile applications, and to Michael Walton and Oliver Bradley who let me do a lot of work and research for the book when I was working for Opsera.

About the Reviewers

Johnny Gee is the Chief Technology Officer at Beach Street Consulting, Inc. In that role, he is responsible for architecting solutions for multiple clients across various industries and building Content Enabled Vertical Applications (CEVAs) on the Documentum platform. He has over 13 years of experience in ECM system design and implementation, with a proven record of successful ECM project implementations.

In addition to earning his undergraduate degree in Aerospace Engineering from University of Maryland, Johnny achieved two graduate degrees—one in Aerospace Engineering from Georgia Institute of Technology, and another in Information Systems Technology from George Washington University.

Johnny is an EMC Proven Professional Specialist in Application Development in Content Management and has helped co-author the EMC Documentum Server Programming certification exam. He holds the position of top contributor to the EMC Support Forums and is one of the twenty EMC Community Experts worldwide. He has been invited on multiple occasions to the EMC Software Developer Conference and has spoken at EMC World. He also has a blog dedicated to designing Documentum solutions.

Johnny was the technical reviewer for Pawan Kumar's revision to "Documentum Content Management Foundations: EMC Proven Professional Certification Exam E20-120 Study Guide". He was also a technical reviewer for Munwar Shariff's book on "Alfresco 3 Web Content Management."

Sivasundaram Umapathy is currently working as a Technical Architect with Sella Servizi Bancari, the IT division of Gruppo Banca Sella, Italy where he is leading the organization's transition to Alfresco and Liferay technologies. He has a "Post Graduate program in Software Enterprise Management" (PGSEM) from IIM, Bangalore and MS in Software Systems from BITS, Pilani. He has an array of certifications ranging from CGEIT, TOGAF 8, PMP, SCEA, OCA, SCBCD, SCWCD, SCMD, to SCJP. He has co-authored "SCMD Exam Guide"(ISBN-9780070077881) and been a technical reviewer of "Head First EJB" (ISBN-9780596005719). His current interests are Enterprise Architecture, IT Governance, IT-Business mismatch, Tech Startups and Entrepreneurship. He can be reached at siva@sivasundaram.com or via his LinkedIn profile at <http://bit.ly/sivasundaram>

Adrián Efrén Jiménez Vega works at the Center of Information Technologies (CTI) of the University of the Balearic Islands, in Mallorca (Spain). For four years, he has built and deployed various applications based on Alfresco.

Since registering in the Alfresco Spanish forum approximately two years ago, he has dedicated time and openly shared his experience posting more than 600 messages, and contributed many practical solutions and useful hints for members of the Community. The 'mini-guides' he developed are now widely used and referenced among developers in Spain and Spanish speaking countries. He obtained the "Alfresco Chumby Awards for Community Achievement" in November 2008.

He won the "Web Script Developer Challenge" with a Web Script solution to limit the space for users, including e-mail notification.

He collaborated as technical reviewer for the book "Alfresco 3 Enterprise Content Management Implementation" (Packt Publishing) in 2009 and recently he has reviewed the book "Alfresco 3 Web Services" (Packt Publishing) in 2010.

I would like to thank all the people who made possible my participation in this project. In particular I would thank my parents (despite the distance), my sister, and my friends at CTI.

www.PacktPub.com

Support files, eBooks, discount offers, and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.



<http://PacktLib.PacktPub.com>

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

This book is dedicated to my wife, Veronika, for always believing in me and accepting that her husband spent most weekends this year in front of the computer, and to my parents Sven-Erik and Irene, without you nothing would have been possible.

Table of Contents

Preface	1
Chapter 1: The Alfresco Platform	7
Platform overview	8
Repository concepts and definitions	10
Repository	10
Stores	12
The Content Store	12
The AVM Store	14
Store reference	15
Nodes	16
Root node	17
Node reference	17
Node properties	18
Node property sheets	19
Node associations	19
QName	19
Permissions	20
User groups	20
Roles	21
Permission groups	22
Owner authority	23
Permission example	23
Multi-Tenant	23
Core platform	23
Open source libraries	24
Services and components	25
Content rules	26
Event model	27
Metadata extraction	35
Content transformation	36
Alfresco Management Beans (JMX)	39

Application Programming Interfaces (APIs)	39
Subsystems	40
Bootstrap	41
Patches	41
Importers	41
Extension modules	42
Third-party extension modules	42
User interface clients	43
Alfresco Explorer	43
Alfresco Share	43
Alfresco SharePoint	43
Alfresco Mobile	45
Alfresco CIFS	45
The Alfresco installation directory structure	46
The alf_data directory	46
The contentstore directory	47
The contentstore.deleted directory	47
The audit.contentstore directory	48
The lucene-indexes and backup-lucene-indexes directories	48
The mysql directory	48
The oouser directory	48
The amps directories	48
The tomcat directory	49
Getting the Alfresco source code	50
The Alfresco database	50
DB schema	51
Significant tables	51
ALF_NODE	51
ALF_NODE_PROPERTIES	52
ALF_NODE_ASPECTS	52
ALF_QNAME	53
ALF_APPLIED_PATCH	53
Example queries and update statements	53
Querying for number of nodes of a certain type	53
Querying for number of nodes stored in a particular month	54
Running a patch again	54
Summary	54
Chapter 2: The Alfresco APIs	57
Application Programming Interfaces (APIs)	59
In-process APIs	59
The Java Foundation Services API	59
Event management API	77
Metadata Extraction API	81
Content Transformation API	86

The JavaScript API	88
Client-server APIs	94
CMIS API	94
Repository API	95
Custom APIs	95
Bootstrap APIs	98
Patches	98
Importers	99
Summary	100
Chapter 3: Setting Up a Development Environment and a Release Process	103
Setting up a development environment	105
Alfresco Extension projects	105
Alfresco Explorer and repository extensions	105
Alfresco Share UI extensions	107
Project directory structure	109
_alfresco/config	110
_alfresco/source	114
_share/config	115
Building and deploying	118
The Build file	118
Using the build file to deploy extensions	120
Debugging extensions	122
Setting up a continuous integration solution	124
Setting up a release process	130
Release notes template	133
Updating Change Log	135
Training	135
Summary	136
Chapter 4: Authentication and Synchronization Solutions	137
Authentication and synchronization concepts	138
Basic authentication	138
NTLM authentication	139
Alfresco CIFS and NTLM authentication	141
Alfresco NTLM passthru authentication	141
Kerberos authentication	142
User and service login via KDC AS	144
Accessing Alfresco via KDC TGS	145
LDAP authentication	147
Checking what SASL mechanisms the LDAP server supports	149
LDAP synchronization	150
Alfresco authentication and synchronization subsystems	150
Alfresco authentication and application zones	151

Setting up authentication and synchronization with Remote Directory servers	152
Configuring authentication and synchronization against OpenLDAP	153
Configuring user authentication with OpenLDAP	153
Configuring user and group synchronizing with OpenLDAP	155
Configuring authentication and synchronization against Microsoft Active Directory	159
Configuring multiple LDAP authentication subsystems	159
Moving OpenLDAP subsystem configuration to its own directory	160
Configuring authentication and synchronization with Active Directory	161
Customizing group imports	163
Accessing via the CIFS interface	163
Implementing a custom authenticator for CIFS authentication against an LDAP server	165
Making authentication more secure and using SSO	172
Troubleshooting NTLM authentication and SSO	173
Using directory servers in a Development Environment	174
Summary	175
Chapter 5: File System Access Solutions	177
File access concepts	178
CIFS protocol overview	178
CIFS Transport—NetBIOS over TCP/IP (NBT)	179
Naming service	180
Communication services	183
CIFS transport—TCP/IP (Native SMB)	185
CIFS dialect negotiation	187
CIFS authentication and security	189
Next generation CIFS—SMB2	190
Alfresco CIFS server	191
Alfresco CIFS server on Windows	192
Alfresco CIFS server on Linux	193
Alfresco CIFS server configuration	194
Alfresco file server subsystem	195
Windows Vista server, Windows 7, and XP clients configuration	195
Windows 2003 Server and Windows 7 client configuration	200
Windows 2008 Server, Active Directory, and Windows 7 client configuration	202
Linux server and Windows 7 client configuration	208
Alfresco WebDAV	210
WebDAV clients	212
Windows built-in WebDAV clients	214
Web Folders (XP only)	214
WebDAV Mini Redirector (XP, Vista, and Win7)	214

Troubleshooting Alfresco CIFS	215
General	215
Nothing happens in Alfresco when trying to log in via CIFS	215
Server says NTLMv2 is not valid for authentication	216
SMBException: invalid parameter and access denied	216
NetBIOS DLL is not accessible	217
Turning on debug logging for SMB	217
Checking ports from server	217
Checking ports from client	218
Checking that CIFS server NetBIOS name is ok	218
Checking that CIFS server NetBIOS name is resolvable from client	219
Does any debug logging show up during connection attempts?	219
Does the client use the correct authentication method?	219
Are you running in a Citrix environment?	220
Summary	220
Chapter 6: Document and Records Management Solutions	223
Out of the box folder hierarchy	225
The Data Dictionary top folder	226
Designing document management solutions	228
Document Folder Template	229
Folder name	229
Folder title	231
Folder permissions	231
Rules	232
Metadata	233
Document versioning	235
Processes	235
Designing the Best Money document management solution	236
Meetings and Press folder hierarchy	236
The Meetings folder hierarchy	236
The Press folder hierarchy	240
Meeting folder/space hierarchy template	242
Implementing the Best Money document management solutions	246
Setting up users and groups	246
Using a script to set up users and groups	246
Setting up the folder hierarchy	249
Using CIFS to set up folders	250
Using the Alfresco user interfaces to set up folders	250
Using scripts to set up folders	251
Setting up folder permissions	257
Setting up business rules for folders	258
Setting up space templates	269
Configuring details list view for folder and file display	271
Configuring Google-Like search	271
Setting up document review periods	272
Adding the reviewable aspect	272

Setting a review period for a folder	274
Creating script to check folder review periods	275
Setting up a scheduler that runs review folder content script	278
Exporting and importing folders, users, and groups	280
Copying folder hierarchies between Alfresco boxes	280
Copying users and groups between Alfresco boxes	281
Introduction to Records Management	281
Alfresco records management	282
Summary	284
Chapter 7: Content Model Definition Solutions	285
Meta Model XML schema	287
model	287
model.imports	289
model.namespaces	290
model.data-types	291
model.constraints	292
model.types	294
model.types.type.properties	295
model.types.type.associations	298
Type definition examples	301
model.aspects	304
Modeling tips and tricks	306
Not changing the out of the box models	306
Starting small	306
Performance	306
Manageability	307
Changeability	307
Defining a new custom type for a domain	308
When to use a type and when to use an aspect	309
Design patterns	310
Domain document root type	310
Problem	310
Solution	310
Diagram	310
Definition example	311
Composite type	311
Problem	311
Solution	312
Definition example	312
Multiple types inheritance	314
Problem	314
Solution	315
Definition example	315

Configuration object	315
Problem	315
Solution	316
Definition example	316
Code example	316
Defining a new custom content model	320
The model definition	320
Registering the model with the repository	325
Configuring property sheets for UI display	326
Alfresco Explorer	326
Alfresco share	330
Summary	344
Chapter 8: Document Migration Solutions	345
Document migration strategies	346
General migration strategies	346
Document staging area	346
Preserving Modified Date on imported documents	346
Post migration processing scripts	348
Importing documents via CIFS	353
Pros and cons with CIFS import	355
Importing documents via external tool	355
Pros and cons with tool import	356
Importing documents via ACP file	357
Common steps during document migration	358
Planning document migration	358
Implementing document migration	359
Using Alfresco bulk filesystem import tool	359
Running Alfresco bulk import tool	360
Running Alfresco bulk import tool and applying extra metadata	362
Using an ACP Generator tool	364
Summary	367
Chapter 9: Business Process Design Solutions	369
Designing business processes with Swimlane diagrams	370
Introduction to Swimlane diagrams	370
Subprocesses	373
Task metadata	375
Process phases	376
Task naming convention	377
Designing the material production process	378
Job process Swimlane diagram	378
Sign-off process Swimlane diagram	379

Studio process Swimlane diagram	380
Work process Swimlane diagram	381
Summary	382
Chapter 10: Business Process Implementation Solutions: Part 1	383
Implementing the marketing production workflow	385
Implementing the Work subprocess	385
Work process—workflow definition (jPDL)	386
Work process—workflow content model	394
Work process—property files for UI labels	401
Work process—using dynamic descriptions and setting task due date	404
Work process—defining the job data	405
Work process—task property sheets	406
Work process—bootstrapping UI property files and property sheets configuration	411
Work process—testing it	412
Running the work process from the Alfresco Share UI	418
Summary	426
Chapter 11: Business Process Implementation Solutions: Part 2	427
Completing the implementation of the marketing production workflow	427
Implementing the Studio subprocess	428
Studio process—workflow definition (jPDL)	428
Studio process—workflow content model	434
Studio process—property files for UI labels	435
Studio process—task property sheets	435
Studio process—bootstrapping UI property files and property sheets configuration	438
Studio process—testing it	439
Implementing the Sign-off subprocess	443
Sign-off process—workflow definition (jPDL)	443
Sign-off process—workflow content model	448
Sign-off process—create and bootstrap the e-mail template	449
Sign-off process—property files for UI labels	451
Sign-off process—task property sheets	451
Sign-off—bootstrapping UI property files and property sheets configuration	453
Sign-off process—testing it	454
Implementing the Job process	455
Job process—workflow definition (jPDL)	455
Job process—workflow content model	467
Job process—property file, property sheets, and bootstrapping	468
Job process—testing it	469
Extending the workflow solution	469
Adding e-mail notification	470
Using customized task dashlets	471

Management dashlets	471
All assigned tasks for all jobs dashlet	472
All job workflows dashlet	473
Exporting the task summary list in an Excel spreadsheet	475
Material folder link	476
Summary	478
Chapter 12: Enterprise Application Integration (EAI) Solutions	479
Introducing portlets	480
Portlet standards	480
Portlet lifecycle	480
Portlet modes and window states	481
Portlet implementation and deployment	482
Implementing portlets that display Alfresco content	483
Portal architecture	483
Alfresco portlet implementation approaches	484
Implementing the "recently added documents" portlet	486
Implementing the "recently added documents" web script	486
Implementing a Java-based "recently added documents" portlet	490
Implementing a GWT/GXT-based "recently added documents" portlet	497
Summary	506
Chapter 13: Types of E-mail Integration Solutions	507
E-mail integration solutions	508
E-mail client talking directly to Alfresco via the IMAP protocol	508
E-mail client talking to Alfresco through custom built plugin and Web Scripts	510
E-mail server talking to Alfresco through custom module and Web Scripts	512
Implementing e-mail management solutions	514
Implementing e-mail management solutions with Alfresco IMAP	514
Configure Alfresco to enable the IMAP server	514
Setting up an IMAP account in Outlook 2007	515
Drag-and-drop e-mail into Alfresco folder in Outlook 2007	517
Viewing the e-mail from Alfresco Explorer	518
E-mail attachment extraction	520
Viewing document metadata from the e-mail client	520
Dragging-and-dropping e-mails into Alfresco Share site	522
How to use Mount Points	523
Summary	525

Chapter 14: Mobile Phone Access Solutions	527
Alfresco mobile web application for iPhone	528
Installing the Alfresco mobile web application	528
Accessing the Alfresco mobile web application	528
A custom mobile application solution for smartphones	533
Mobile application architecture overview	534
Mobile application feature overview	535
User authentication	535
Folder and document browsing	536
Document search	537
Setting up the mobile Grails application	538
Configuring the mobile Grails application	539
Implementing the CMIS service	540
Fetching the folder root node reference from the CMIS service document	541
Authenticating the user with Alfresco	543
Fetching child content for a folder via CMIS	544
Searching the Alfresco repository via CMIS	545
Implementing the helper methods for the CMIS service	547
Implementing UI controllers	551
Implementing the Groovy Server Pages (GSP)	555
Implementing an authentication filter	558
Running the mobile application	559
Content creation with MobileX	559
Using the Apache chemistry API	559
Summary	560
Index	561

Preface

Alfresco is a renowned and multiple award-winning open source Enterprise Content Management System that allows you to build, design, and implement your very own ECM solutions. It offers much more advanced and cutting-edge features than its commercial counterparts with its modularity and scalability. If you are looking for quick and effective ways to use Alfresco to design and implement effective and world class business solutions that meet your organizational needs, your search ends with this book.

Welcome to *Alfresco 3 Business Solutions*: Your practical and easy-to-use guide, which instead of teaching you just how to *use* Alfresco, teaches you how to *live* Alfresco. It will guide you through implementing real-world solutions through real-world scenarios. Each ECM problem is treated as a separate case study and has its own chapter, enabling you to uncover the practical aspects of an ECM implementation. You want more than just the theoretical details – you want practical insights to building, designing, and implementing nothing less than world class business solutions with Alfresco – and *Alfresco 3 Business Solutions* is your solution.

This practical companion cuts short the preamble and you dive right into the world of business solutions with Alfresco.

- Learn all techniques, basic and advanced, required to design and implement different solutions with Alfresco in easy and efficient ways
- Learn all you need to know about document management
- Connect Alfresco with directory servers
- Learn how to use CIFS and troubleshoot all types of problems
- Migrate data when you have an existing network drive with documents and want to merge them into Alfresco
- Implement Business Process Design Solutions with Swimlane diagrams

- Easily extract content from Alfresco and build mashups in a portal such as Liferay
- Gain insights into mobile access and e-mail integration

This book will teach you to implement all that and more, in real-world environments.

What this book covers

Chapter 1, The Alfresco Platform, introduces the architecture behind the repository and goes through important concepts such as store, node, and association. It describes the major features that are available such as rules, events, metadata extractors, transformers, subsystems, patches, and so on. It also explains the directory structure and database schema of an Alfresco installation.

Chapter 2, The Alfresco APIs, presents the remote and embedded APIs that are available out of the box. It focuses on the embedded Foundation API and the JavaScript API and shows how to create, update, delete, and search for content.

Chapter 3, Setting Up a Development Environment and a Release Process, shows you how to set up a development environment so you can build both Alfresco Explorer extensions and Alfresco Share extensions. It also covers how to manage a release process.

Chapter 4, Authentication and Synchronization Solutions, describes how the authentication subsystem is working and how to configure it for the LDAP and the Microsoft Active Directory. It also covers how to synchronize user and group data with these directories.

Chapter 5, File System Access Solutions, teaches you what CIFS is, how the underlying technology works, and how to troubleshoot it. It will take you through different configurations on different platforms. It also covers WebDAV and how to use that instead of CIFS.

Chapter 6, Document and Records Management Solutions, covers how to design folder hierarchies with permissions, rules, and custom metadata using a folder template. It shows a lot of examples on how to use the JavaScript API when implementing DM solutions. It also introduces the Alfresco RM module.

Chapter 7, Content Model Definition Solutions, explains the XSD Schema/meta model that describes the Alfresco content modeling language. It shows a lot of examples on how to design custom content models on top of the out-of-the-box Alfresco content models and how to display the custom data in Alfresco Explorer and Alfresco Share. It also shows you a couple of design patterns that can be used for content modeling.

Chapter 8, Document Migration Solutions, teaches you strategies for implementing a document migration solution. It explains advantages and disadvantages between different import tools such as ACP file, CIFS, and the Alfresco Bulk File system Import Tool.

Chapter 9, Business Process Design Solutions, shows you how Swimlane diagrams can be used to design business processes. It explains how a task-naming convention can be useful to distinguish between tasks and how to design using phases and sub-processes.

Chapter 10, Business Process Implementation Solutions: Part 1, introduces the JBoss jBPM workflow engine that is used by Alfresco. It takes you through implementing a simple workflow and introduces jPDL concepts such as task node and decision node.

Chapter 11, Business Process Implementation Solutions: Part 2, digs deeper into implementing workflows with JBoss jBPM and introduces concepts such as fork node, join node, phases, and sub-processes.

Chapter 12, Enterprise Application Integration (EAI) Solutions, shows you how to build portlets that fetch content from Alfresco via remote Web Script calls.

Chapter 13, Types of E-mail Integration Solutions, describes different types of e-mail management solutions and goes through how to configure Alfresco's built-in IMAP service.

Chapter 14, Mobile Phone Access Solutions, takes you through building a small mobile client with Groovy and Grails. The CMIS interface is used to fetch content from Alfresco.

What you need for this book

To build the examples in this book you will need JDK 6, Apache Ant, and Alfresco SDK 3.x. To run the examples, an Alfresco 3.x (including MySQL) installation is needed. Some examples (Chapter 4) require a directory server such as OpenLDAP or the Apache Directory Server. For the portal integration example (Chapter 12), you will need to install Liferay Portal 6 and download GWT 1.7 and GXT 2.2. For the mobile application example (Chapter 14), you will need to install Grails 1.3.x.

Who this book is for

This book is designed for system administrators and business owners who want to learn and implement Alfresco Business Solutions in their teams or business organizations. General familiarity with Java and Alfresco is required.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "Next, we'll add a function to the `CModel` class that will allow us to set a given effect to any given mesh part."

A block of code is set as follows:

```
private LdapTemplate m_ldapTemplate;  
private String m_userBase;
```

When we wish to draw your attention to a particular part of a code block, the relevant lines or items are set in bold:

```
<types>  
  <type name="{namespace:typeName}">  
    <title>{description of file domain type}</title>  
    <parent>cm:cmobject</parent>  
    ...  
  </type>  
</types>
```

Any command-line input or output is written as follows:

```
18:28:39,842 INFO [management.subsystems.  
ChildApplicationContextFactory] Starting 'Authentication' subsystem, ID:  
[Authentication, managed, bestmoneyLDAP]
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: " This Swimlane diagram shows a subprocess called **Work Process** that is called from a parent process called **Studio Process**".

 Warnings or important notes appear in a box like this.

 Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book – what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.



Downloading the example code for the book

You can download the example code files for all Packt books you have purchased from your account at <http://www.packtpub.com>. If you purchased this book elsewhere, you can visit <http://www.packtpub.com/support> and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books – maybe a mistake in the text or the code – we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting <http://www.packtpub.com/support>, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from <http://www.packtpub.com/support>.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1

The Alfresco Platform

Before we dive into implementing ECM solutions, we are going to have a look at the Alfresco platform. There are some key concepts and features that are important for us to know about before implementing anything on top of Alfresco.

It helps to think about Alfresco as a big toolbox for building **Content Management Systems (CMS)**. Alfresco, out of the box, can obviously be used straightaway but usually you want to configure it and customize it for the organization that is going to use it.

This is important to think about, as otherwise you are missing the full potential of Alfresco. It enables organizations to tweak Alfresco, so that it works with their business processes and business rules. It does not impose a special way of working that the organization has to adopt. Instead, *Alfresco adapts to the organization*.

In a lot of cases, organizations buy proprietary turnkey solutions that look really good out of the box with predefined content models, domain process definitions, business rules, and so on. However, after a while they usually realize that things are not working exactly as they want them to. Then they realize that to customize the solution will cost way more than if they would have started creating it from scratch, or it might not even be possible to customize functionality in the proprietary solution.

In this chapter, you will learn:

- Important repository concepts
- How to use content rules
- What a metadata extractor is
- Why content transformers are used
- How to trigger custom code from events
- What a Servlet Command is

- What a subsystem is
- How the system can be bootstrapped in different ways
- What user interfaces are available
- About the directory structure created by the installation
- How to access content information directly from the database

Throughout the book, we will be working with "Best Money" – a fictive financial institution that offers financial products such as credit cards, loans, and insurances. Best Money wants to complete its range of financial products by offering personal banking products. It is therefore under pressure to improve efficiency by automating business processes, structuring document storage, classifying documents, implementing document lifecycles, improving the level of auditing, managing e-mail content, and many other challenges in a complex business environment subject to heavy regulatory oversight.

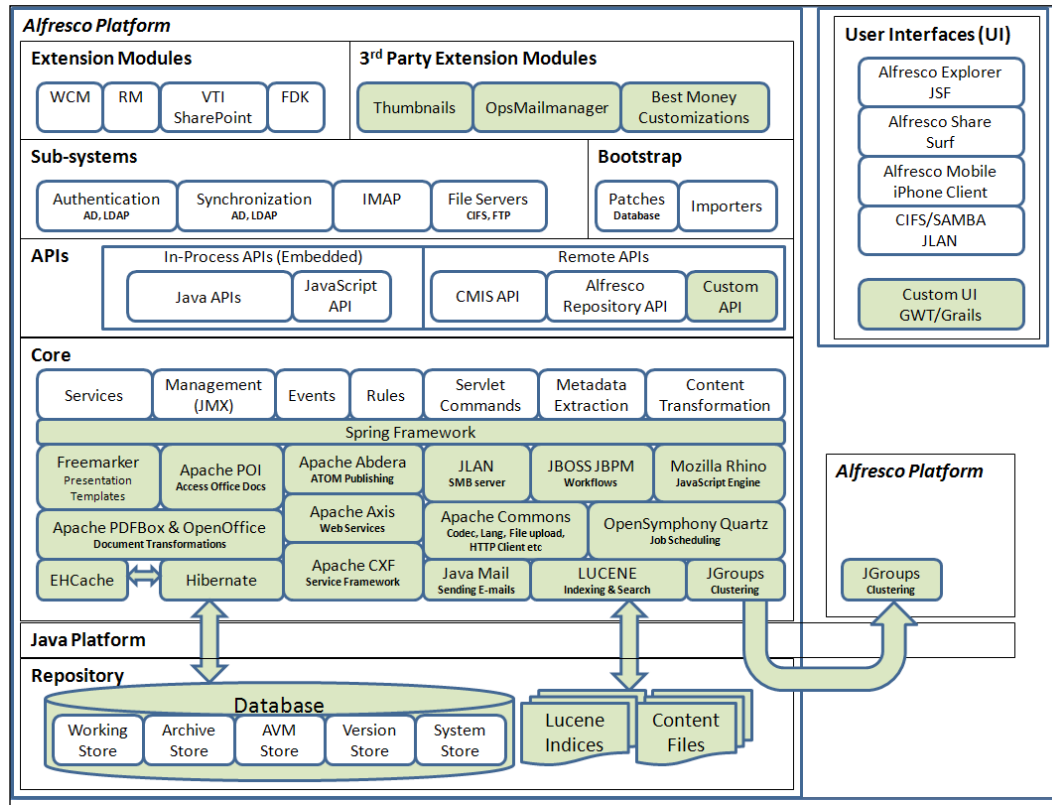
Best Money realizes that to do all of this it needs to put in place an Enterprise Content Management solution and it has selected Alfresco.

Platform overview

Alfresco is an open source content management system written entirely in Java that can be run in a standard Servlet container, such as Apache Tomcat or a JEE server, such as JBoss. The Alfresco platform is built using many third-party open source Java libraries and it's good to know about these libraries as we will use many of them when building extensions and solutions.

The platform has many **Application Programming Interfaces (APIs)** and configuration techniques that we can use to build custom solutions on top of Alfresco.

The following figure gives an overview of the platform:



The Alfresco-specific components, modules, and user interfaces are depicted in a lighter color and the third-party libraries are depicted in a darker color. The Alfresco platform is presented in a layered approach as follows:

- **Repository:** The bottom layer comprises the database, the search indices, and the content files.
- **Java Platform:** Everything runs on Java, so it is independent of hardware, operating systems, and also of databases as they are accessed via Hibernate.
- **Core:** This layer contains all of the modules and libraries used by Alfresco to implement the CMS functionality.

- **APIs:** The interface layer contains a variety of application programming interfaces that can be used to communicate with Alfresco both in-process and remotely.
- **Sub-systems:** This layer consists of self-contained components that extend the CMS system with important functionality that often need to be configured during installation. Sub-systems can be started and stopped while the server is running.
- **Bootstrap:** System integrators can use bootstrap extensions to perform a variety of tasks, for example, to import content or patch the content with some custom metadata.
- **Modules:** The modules usually extend the Alfresco system with some major extra functionality such as web content management or records management. We will use a module for all new custom functionality we implement for the Best Money ECM system.
- **User Interfaces (UI):** Alfresco comes with a number of user interfaces that can be used to upload and manage content.

Now, let's have a detailed look at each layer starting with the **Repository**.

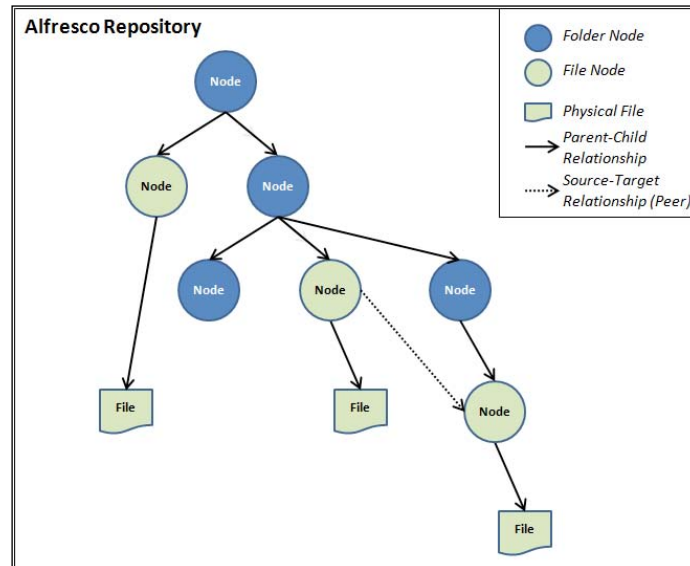
Repository concepts and definitions

Before doing any custom coding for Alfresco, it is important to get to know the concepts and definitions around the repository. We need to get familiar with things such as **Store** and **Association**. And what does it mean when we talk about a **Node** in the repository.

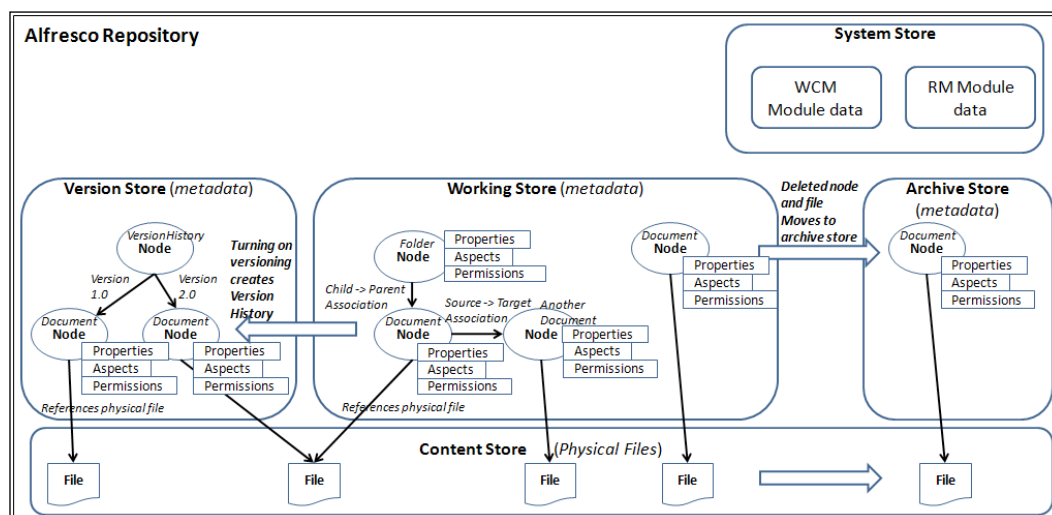
Repository

When we talk about Alfresco, we often refer to the Alfresco Repository. So what is the repository more specifically? The **repository** is an abstract term for where everything is stored in Alfresco. It is also often called just "repo" and one of the main packages in the source code is also called `repo`.

The following figure gives you an overview of the Alfresco Repository:



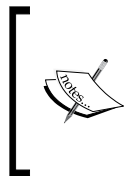
The repository consists of a hierarchy of different types of nodes. This can be for example, folder nodes or leaf nodes representing a file. Each node is associated with a parent node via a parent-child relationship, except the top root node. Nodes can also be related to each other via source-target associations (that is, peer associations). If the node represents a file, then it is also associated with a file in the filesystem. This is a somewhat simplified view of the repository as each node actually resides in a store as in the following figure:



The repository is built up by a number of stores and each one of them contains a node hierarchy. The nodes make up the metadata for the content and are stored in the database. The actual content itself, such as document files, is stored in the filesystem.

Stores

There are a couple of stores that you will come in contact with, if you work with Alfresco for a while. First, we have the **Working Store**, which is the main store where metadata for all live content is contained; this store is often referred to as just **DM (Document Management)**. This is the content that you can access from all the different user interface clients. The default behavior when something is deleted from the Working Store via any of the user interfaces is that both the content file and the content metadata ends up in a store called the **Archive Store**.



Content is not permanently removed from the disk when it is in the Archive Store. To physically remove the content from the disk, you need to manage content via the **Admin** user profile screen or configure a content store cleaner (http://wiki.alfresco.com/wiki/Content_Store_Configuration).

If you turn on "versioning" for a document, then you will see some activity in the **Version Store** where all the previous versions of a piece of content will be stored. This is called the "version history" and there is one Node created per version history.



The complete file for a previous version is stored and the system does not store the delta between versions.

Whenever you install a new application module such as Records Management or Web Content Management, the data about this module is stored in the **System Store**. The data that is stored is, for example, module version number.

Finally, we have the **Content Store** that contains all the *physical files* and it lives on the disk compared to the other stores that live in the database. It is called Content Store even though the behavior is not the same as for the stores that live in the database. It is more of an abstract term for where the physical content files are located.

The Content Store

So why are the physical files stored in the filesystem and not in the database as **Binary Large Objects (BLOBs)**? It would be easier to back up the whole system and also to set up replication if everything was in the database. And system administrators would not have to manage both database space and filesystem space.

There are several reasons why content files are stored in the filesystem:

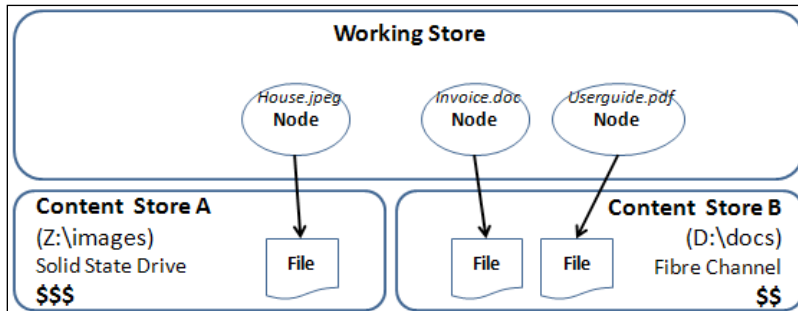
- **Random access to files:** One of the big advantages with Alfresco is that users can keep working the way they are used to by accessing Alfresco as a shared drive (that is, via the CIFS interface). However, this would not be possible if Alfresco was not storing the files in the filesystem, so they can be randomly accessed (sometimes also referred to as direct access). To support frequent updating and reading of database BLOBs would slow down performance of the CIFS interface to an unacceptable level.
- **Real-time streaming:** It is a lot easier to stream large content such as video and audio using files. A content file can now be streamed directly back to the browser as the file input stream is copied directly to the HTTP Response Output stream. If BLOBs were used, you would first have to read the BLOB then create a temporary file to stream from. Also, when writing BLOBs to the database, a lot of databases require you to know the stream size when inserting the record, so a temp file needs to be created. Further on, some databases such as MySQL have problems sending very large binaries from the JDBC driver to the database.
- **Standard database access:** Most database systems support BLOBs with custom access methods and custom objects. These usually perform much better than the JDBC BLOB objects and access methods. So it would be difficult to use Hibernate to access BLOBs in a standard way for all databases. For example, if you wanted to manage BLOBs with Oracle, you would have the best performance using their BLOB object. Also, the caching of BLOBs in databases is known to slow down the rest of the metadata access.
- **Faster access:** It is much faster to access content that is stored as files, which means that the user experience is much better and this leads to happier customers.

Content Store policies

Content Store policies let us decide what media we will store the selected content to. Quite often, content will have a lifetime during which it is relevant and then it will become obsolete; content store policies help with a solution for this. We do not want to get rid of the content files but store them on a cheaper, slower-access disk.

So we might use a very fast tier 1, **Solid-State Drives (SSD)**, for our most important content files, and based on business policies that we control, gradually move the data to cheaper tier 2 drives such as **Fiber Channel (FC)** drives or Serial ATA drives as it becomes less important. In this way, we can manage the storage of content more cost-effectively.

So you could have one part of the repository store files on one disk and another part of the store files on another disk. The following figure illustrates:



In the preceding figure, we can see that the system has been configured to store images on one disk and documents on another disk. This sort of content store configuration can be done with **Content Store Selectors**.

The AVM Store

One store that has not been mentioned so far is the special store introduced for the Alfresco WCM module. It is called the **Advanced Versioning Manager (AVM)** Store and it is modeled after Apache Subversion to be able to support extra features such as:

- File-level version control
- File-level branching
- Directory-level version control
- Directory-level branching
- Store-level version control (snapshots)
- Store-level branching

These extra features are needed to be able to create user and staging sandboxes, so that web content can be created and previewed independently between the users. A staging environment is also supported where different snapshots of the website can be managed and deployed to production servers.

There are some major differences between the Working Store (DM) and the AVM Store (WCM) that are good to know about when we are planning an ECM project. The following list explains some of the differences:

- Permissions can be set on object level in DM but only on Web Project level in WCM
- Types are defined with XML Schema files in WCM but with XML files in DM
- AVM folders do not support rules as in DM
- In WCM, we can search only one Web Project at a time, whereas in DM the complete repository is searchable
- E-mailing with SMTP or IMAP is not supported in WCM, but it is in DM

Content can be cross-copied between the DM store and the AVM store and vice versa.

There are things happening now and in the near future to update Alfresco WCM to be able to use the normal Alfresco DM Working Store, so that web content can reside along with all other content and be searchable in the same way.

Store reference

When you work with the application interfaces, you often have to pass a so-called store reference into a method call. This store reference tells Alfresco what store you are working with. A store reference is constructed from two pieces – the protocol and an identifier.

The *protocol* basically specifies what store you are interested in. For example, two of the protocols are *workspace* and *archive*. You also need an identifier to create a store reference and it tells us what kind of store it is, for example, does it contain spaces or is it keeping version information. Most of the time we are accessing a store with folders (that is, spaces) and the *identifier* is then called *SpacesStore*.

So if you wanted to access the Working Store from the previous figures, you would create the following store reference: `workspace://SpacesStore`. And this is the store that you will use most of the time.

The following is a complete list of store references:

- `workspace://SpacesStore`: Contains the live content; this is the store reference that will be used in most situations
- `workspace://lightWeightVersionStore`: Version history for content
- `workspace://Version2Store`: Next-generation version history for content

- `archive://SpacesStore`: Archived files (that is, deleted files)
- `user://alfrescoUserStore`: User management
- `system://system`: Installed modules information
- `avm://sitestore`: Alfresco WCM content

Nodes

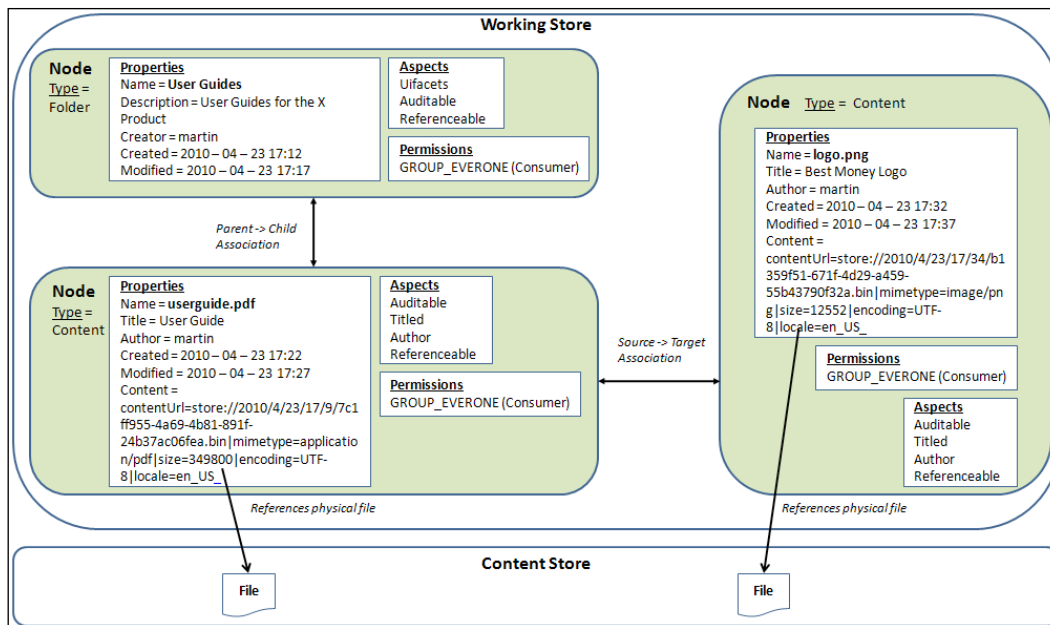
Each store in the repository contains nodes and every piece of content that is saved in the repository is represented by a node. This can be a document, a folder, a forum, an e-mail, an image, a person, and so on. Everything in a store is a node. A node is stored in the database and contains the following metadata for a piece of content:

- **Type**: A node can be of one type.
- **Aspects**: A node can have many aspects.
- **Properties**: A node can have properties defined in the type or the aspects. One of the properties points to the actual physical content file.
- **Permissions**: Permissions for this node.
- **Associations**: Associations to other nodes.

Each node is of a certain **Type** such as `Folder`, `Content`, `VersionHistory`, `Forum`, and so on. Each type can have one or more properties associated with it. A node can only be of one type. So a `Folder` cannot also be a `VersionHistory`, pretty obvious, but it is good to mention this anyway so that there are no misunderstandings when we start creating custom types.

So what if we wanted to have properties from two different types associated with a node, how would we do that? We would use something called an **aspect**. A node can be associated with more than one aspect. An aspect is for example, `Versionable`, `Emailed`, `Auditable`, and so on. So, this means that a MS Word document could be of type `Content` and be `Versionable` and `Emailed`.

The following figure shows a folder node called **User Guides** that contains one document called `userguide.pdf`, which in turn is associated with an image node called `logo.png`:



Some nodes such as folder nodes are not associated with any content; they just contain metadata, permission settings, and an association to the parent folder.

Root node

All nodes have to have a parent node and the top-level node in the store is called the **store root** as it does not have a parent node. The root node has an aspect applied to it called `sys:aspect_root`. It might look like a good idea to search for this aspect to get to the root node in a store, but it does not work as there are other nodes such as the root node for categories that also have this aspect set.

An easy way to get to the root node in any of the stores is to do a Lucene search with `PATH: "/"` or if we are using the Java Foundation Service API, we can use the Node Service to get the root node for a particular Store Reference.

Node reference

So, we have heard about all these nodes and seen how they can have properties, and so on. But how can one uniquely identify one node in the repository? This is where node references come into the picture. They are used to identify a specific node in one of the stores in the repository. You construct a node reference by combining a **Store Reference** such as `workspace://SpacesStore` with an **identifier**.

The identifier is a **Universally Unique Identifier (UUID)** and it is generated automatically when a node is created. A UUID looks something like this: 986570b5-4a1b-11dd-823c-f5095e006c11 and it represents a 128-bit value. A complete Node Reference looks like `workspace://SpacesStore/986570b5-4a1b-11dd-823c-f5095e006c11`.

The node reference is one of the most important concepts when developing custom behavior for Alfresco as it is required by a lot of the application interface methods.

Node properties

Properties contain all the information about the Node and are often referred to as metadata. When you create a new node of a certain type, such as `Content`, there are certain default number of properties that are set. These are properties such as `Name`, `Created Date`, `Author`, and so on.

What properties are set, and if they are set automatically, depends on the MIME type of the content that is being added to the repository.



Multipurpose Internet Mail Extensions (MIME) is an Internet standard that extends the format of e-mail to support message bodies with multiple parts, text in character sets other than ASCII, non-text attachments, header information in non-ASCII character sets, and so on.

The MIME type is these days referred to as a content type after the header name `Content-Type`. But in the Alfresco environment they are called MIME types.

Some MIME types such as the one for a MS Word document have so-called **Metadata Extractors** available that automatically extract properties from the content when it is added. So when we add a MS Word document to the repository, we will see that some properties have been filled in automatically via the automatic metadata extraction.

Properties can be defined either as part of a type or as part of an aspect. When you list the properties in the UI, it does not show what type or aspect they belong to. You have to programmatically query the system to find out what properties belong to an aspect or type.

What if we wanted to add a property called `Name` but it is already defined for type `Content`, what do we do then? Every property is scoped within a so called **namespace**, so we can have a property called `Name` in namespace A and in namespace B without any problem.

Node property sheets

So we got all these properties for a node in the repository, how do we display them in the UI? They are displayed through so-called **property sheet** definitions. The default property sheet displays a number of properties depending on the type of the node and what aspects have been applied to it.

When we add custom types and aspects to a node, we also have to define appropriate property sheets, so that these custom properties are displayed every time we look at this node.

Property sheets are used to display custom properties both in the Alfresco Explorer UI and in the Alfresco Share UI.

Node associations

Most of the nodes in the repository are linked or associated with other nodes. For example, document nodes are associated with folder nodes and e-mail attachment nodes are associated with e-mail nodes.

There are two kinds of associations: a *parent -> child* association that you, for example, have between a folder and document, and the *source -> target* association (that is, peer association) that you have between, for example, a marketing brief and produced marketing materials.

If we delete a parent node in a *parent -> child* association, an automatic cascading delete will be executed deleting all child nodes. This means that if we delete a folder, all the contained documents and subfolders will also be deleted. Deleting any node in a *source -> target* association does not automatically delete the associated node.

Node associations can also be configured to be displayed in property sheets in the user interface.

QName

When we define a property, aspect, type, and so on for a node in the repository, we will come in contact with something called `QName`, which is the fully qualified name of, for example, a property including the local name and the namespace it has been defined in.

Here are some examples of QName:

- `{http://www.alfresco.org/model/content/1.0}description`: Defines the description (that is, description is the local name) property that is part of the standard Alfresco content namespace. This property is part of the `cm:titled` aspect defined in the same namespace.
- `{http://www.bestmoney.com/model/content/1.0}job`: Defines the job aspect that is part of the "Best Money" content namespace.
- `{http://www.mycompany.com/model/content/1.0}language_options`: Defines a constraint of languages in some company's namespace.

The QName does not actually tell us if it defines a property, type, constraint, aspect, and so on, it does not know anything about that. The local name of a QName has to be unique within a namespace. So there can only be one QName with local name `description` within the Alfresco namespace `http://www.alfresco.org/model/content/1.0`. If we wanted to have a property description for another aspect such as the person aspect, we would have to call it something different such as `persondescription`. We could also use a different namespace `{http://www.mycompany.com/model/person/1.0}description`.

The namespace is written with quite a long string but can be shortened by using the prefix that has been set for it. The `http://www.alfresco.org/model/content/1.0` namespace has the prefix `cm`. So we can use `cm:description` to refer to the `description` property of the `titled` aspect in the Alfresco content namespace.

Permissions

We can set individual permissions for every node in the repository. For example, when we create a new folder node, it can be set up to be accessible by only one particular person or one specific group of people. We can also just inherit permissions from the parent node; this is the default behavior if permissions are not manually specified when a node is created.

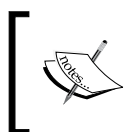
User groups

Groups of users can be created to better handle permission settings. Usually it is a good idea to set permissions for folders and content via a group. Then we can just add and remove members from the group without having to worry about setting or removing permissions for individual users. And when permissions are changed for a group they are immediately propagated to all users.

Users and groups can be synchronized with an external directory server and then we can manage the group membership in a more centralized way.

There are a couple of groups created automatically when Alfresco is installed and they all have special meanings as follow:

- **EVERYONE:** This is a group that implicitly has all users as members. This group is not viewable when managing groups via the user interface, but you can use this group when setting up permissions.
- **ALFRESCO_ADMINISTRATORS:** Any member of this group has administrator rights, meaning full access to the complete repository. This group has one member as default, which is the admin user.
- **E-MAIL_CONTRIBUTORS:** One of the Alfresco features is to be able to receive e-mails via a built-in SMTP service. Any e-mails sent into Alfresco cannot be stored unless the user sending the e-mail has been authenticated. The authentication is done via the sender's e-mail address and the user matching the e-mail address must be a member of this group. If the user is not a member of this group, then the e-mail will not be stored.



Groups can also be used for other things than permission management, for example, sending e-mails to members of a particular group or assigning workflow tasks to members of a group.

Roles

The permission system is role-based where each role groups together one or more individual permissions such as a Write or Read permission. To configure permissions for a node, we first choose what person or group – and this is referred to as the authority – should have access to the node, and then we specify what kind of access the authority should have to the node by associating it with a role.

There are five roles available out of the box and they are:

- **Consumer:** Gives the authority Read permission to the node
- **Contributors:** Gives the authority Read and Create permission to the node
- **Collaborator:** Gives the authority Read, Create, and Update permission to the node
- **Editor:** Gives the authority Read and Update permission to the node
- **Coordinator:** Gives the authority full access to the node

The permissions specified for each role in the previous list are a generalization of the permissions for each role. It is good to think about the roles like that when discussing with clients and explaining the permissions for each role. For a complete detailed list of the permissions for each role, see the following page: <http://www.alfresco.com/help/webclient/concepts/cuh-user-roles-permissions.html>.

For Alfresco WCM, see the following page: http://wiki.alfresco.com/wiki/WCM_roles.

Permission groups

Permission groups are used to group together one or more related permissions. We need to worry about permission groups when we, for example, want to define a new role. The low-level permissions in Alfresco have been grouped as follows:

- **Read:** Includes permissions groups: `ReadProperties`, `ReadChildren`, and `ReadContent`
- **Write:** `WriteProperties`, `WriteContent`
- **Delete:** `DeleteNode`, `DeleteChildren`
- **AddChildren:** `CreateChildren`, `LinkChildren`
- **Execute:** `ExecuteContent`

All low-level permissions have also been defined as groups containing only the individual permission. The individual permission name is the same as the group name but with an underscore at the beginning (for example, `_ReadProperties`).

Then there are higher level permission groups for the roles such as:

- **Consumer:** Includes permission group `Read`
- **Contributor:** `Consumer`, `AddChildren`, `ReadPermissions`
- **Editor:** `Consumer`, `Write`, `CheckOut`, `ReadPermissions`
- **Collaborator:** `Editor`, `Contributor`
- **Coordinator:** Full control (can do anything)

If you want to have a look at the complete permission model, it is available in the `/tomcat/webapps/alfresco/WEB-INF/classes/alfresco/model/permissionDefinitions.xml` file.

Owner authority

There is a special authority called **Owner**, which is basically what we call a user that created a piece of content. The Owner of content always has full access (that is, same as the Coordinator role) to it no matter what permissions have been set up. Someone can take ownership of content if they have Coordinator or Administrator rights. An extra aspect `cm:ownable` is then applied to the content with a property `cm:owner` that specifies the username for the new owner.

Permission example

For a complete permission example, see the following page: http://wiki.alfresco.com/wiki/Security_and_Authentication#A_Simple_Permissions_Example.

Multi-Tenant

Alfresco is usually installed and used locally at a company or an organization. However, sometimes it might be useful to be able to divide the repository, so that content belonging to one group of users is kept separate from another group.

Maybe we are service providers who want to offer content management solutions as a service and we want to do this from a single Alfresco installation. This can be done by using the **Multi-Tenant (MT)** feature of Alfresco.

The MT feature enables Alfresco to be configured as a single-instance multi-tenant environment, which enables multiple independent tenants such as companies, organizations, or groups to be hosted on a single instance. This instance can be installed either on a single server or across a cluster of servers.

The MT feature is not enabled by default in Alfresco and has to be turned on by configuration in XML files. There is a special **Tenant Administration Console** that can be used to create new tenants, show tenants, enable and disable tenants.

Core platform

The core Alfresco platform is built on Java, which makes it deployable on any platform with a JVM. Typically, Alfresco is deployed to Apache Tomcat, but it is also possible to deploy Alfresco to **Java Enterprise Edition (JEE)** servers, such as JBoss, WebSphere, and WebLogic. The platform makes use of many of today's best open source projects to build a first class content management solution.

Open source libraries

One of the reasons Alfresco could build such a powerful content management solution in just a few years is that instead of reinventing the wheel for every needed feature, they looked at what open source projects were available that could provide the functionality that was needed. By using the best open source projects, Alfresco can develop functionality really fast and the solutions are much more stable than if everything was created from scratch.

The following is a list of the major open source projects that are used to implement the core platform:

- **Acegi Security:** Used for authentication and method-level authorization
- **Apache Axis:** Web Service container
- **Apache Abdera:** ATOM syndication format and publishing protocol
- **Apache Chemistry:** CMIS AtomPub binding and Abdera CMIS Extension for Web Services binding
- **Apache CXF:** Service Framework for CMIS Web Services
- **Apache Commons:** A lot of the commons libraries are used such as Codec, Lang, File Upload, HTTP Client
- **Apache PDFBox:** Document transformations
- **Apache POI:** Access Microsoft Office documents
- **Chiba:** WCM Forms Engine
- **EHCache:** Level 2 caching
- **FreeMarker:** Presentation templates
- **Greenmail:** IMAP support
- **Hibernate:** Database access via Object-relational mapping
- **iBATIS:** SQL mapping layer, replaces Hibernate in some places in Alfresco 3.3, but will replace Hibernate completely in the future versions of Alfresco – for scalability and reliability issues
- **Java Mail:** Sending e-mails
- **JBOSS jBPM:** Workflow engine
- **JGroups:** Clustering support via multicast protocol
- **Lucene:** Indexing and searching
- **OpenOffice:** Transforming office documents to text
- **Spring:** Dependency Injection (DI) component container
- **Mozilla Rhino:** Server-side JavaScript engine
- **OpenSymphony Quartz:** Job scheduler

Services and components

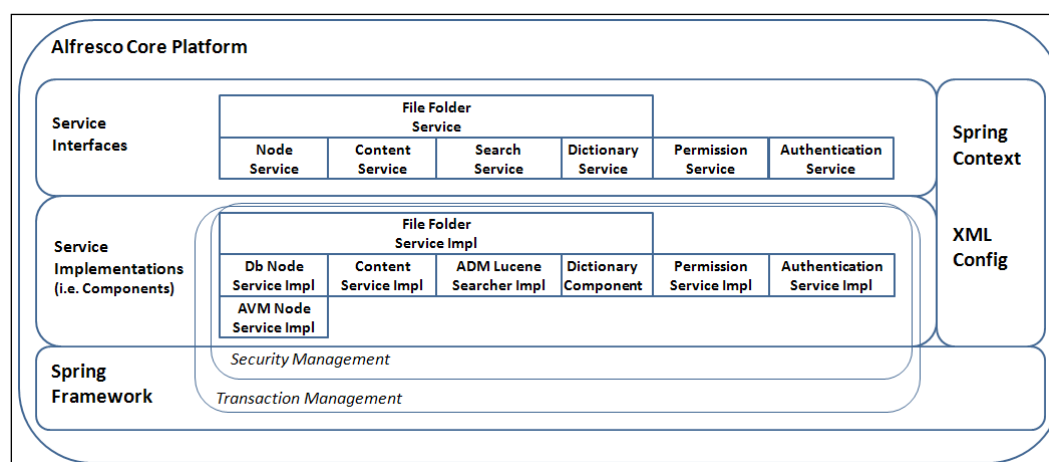
The Alfresco platform is built up around many so-called *services* such as the Node Service – that is used to manipulate the nodes in the repository, or the Content Service – that is used to manipulate the content that the nodes are pointing to. You can think of a service as the interface to some piece of CMS functionality. Each service has one or more implementations that are sometimes referred to as **components**.

The core platform uses the Spring Framework as its component container, so anyone familiar with Spring would feel at home looking at the source code and the configuration files. Spring also makes the platform very flexible and easy to customize.

Every service is available as a Spring Bean, so we can easily customize them by overriding with our own implementations, if so, only change a minor feature and use the rest as is.

The platform also uses the aspect-oriented features of Spring to support transactions and security in a transparent way. Using an aspect-oriented approach also minimizes code duplication and intrusiveness in the service implementations.

The following figure illustrates:



These service interfaces are often referred to as the Alfresco Foundation Services and they are the lowest level of public interfaces giving access to the Alfresco content management functionality. A Spring client, in the same process as the Alfresco repository, can inject these services into custom code.

The **File Folder Service** is a little bit different from the rest of the services, as it uses the other services to implement its functionality. So you could say that it is not at the lowest level but abstracts some of the other services.

An important point is that the Foundation Services is where the transaction and security policies are enforced. These policies are declaratively specified and enforced for each service. The service interfaces are also stateless in the design, so all the data (that is, state) needed for a method operation has to be passed in to the service call.

Anyone familiar with the *Strategy Design Pattern* will see several uses of this pattern in the Alfresco platform. For example, when you configure an authentication method, you select a particular authentication strategy such as LDAP Authentication.

If the authentication strategy you are looking for is not available, such as LDAP server authentication when using CIFS, you can create your own authenticator component and just plug it into the platform. (*Chapter 4, Authentication and Synchronization Solutions* shows you a solution on how to do just this.)

Content rules

An important part of any content management platform is to be able to automate business rules. In Alfresco, you can set up content rules to enforce business rules. Rules are applied to folders and they consist of the following parts:

- **General Description:** Information about what the rule does
- **Condition:** Consists of two parts:
 - When to execute the rule (one or more can be selected):
 - *Inbound* (when documents are created or added to a folder)
 - *Outbound* (when documents are deleted or moved from a folder)
 - *Update* (when documents are updated)
 - **Selection criteria** (one or more criteria can be specified):
 - *Property Values* (for example, if document author is martin, if document name contains the text security, and so on)
 - *Has Aspect* (for example, if document has aspect e-mailed applied)
 - *Document of Type or Subtype* (for example, if document is of type meeting)
 - *All documents* (default)
 - And more...

- **Action:** What to do when the rule is executed (one or more actions can be selected). A content rule action can be things, such as move a document to another folder, transform an MS Word document into a PDF, apply versioning, send an e-mail, run a script, and so on.

We normally set up these content rules from the Alfresco user interface but they can also be imported into the repository in a bootstrap procedure.

Content Rules are defined per folder and if a rule should apply to a lot of different folders, such as an "Apply Versioning" rule, it does not have to be defined for all folders. It is enough to define the rule for one folder and then link to that rule from all other folders that should have the same rule applied.

If more than one rule is defined, it is possible to specify in what order they should be executed. For example, let's say, we have the following rules:

- Transform MS Word documents to PDF
- Apply versioning to MS Word documents

If we run the rules on a new MS Word document, we will end up with a PDF version of the document without versioning applied, and versioning applied to the MS Word document. However, if we change the order of the rules so that the versioning is applied first, then the PDF version of the document will also have versioning applied. So it is very useful to be able to set the order of execution for the rules.

Rules can be executed asynchronously, which is good, as then operations such as adding a document to a folder will not be affected by how long it takes to execute all the enabled rules. The document will just be stored immediately and then at a later time the rules will be executed. What then happens if there is an error executing the rules later on? Can something be done to notify the users or send an e-mail? Yes, a script can be associated with a rule to run if an error occurs.

Event model

Sometimes, using rules might not be enough for what we want to do. Let's say that we wanted to execute a business rule just before a node is deleted and a business rule after the node has been deleted. This cannot be done with rules, as they allow us to execute business rules only when nodes are created, deleted, or updated. Other examples are executing business logic after version changes for a document or when an association is deleted.

Also, in some cases the rule will not behave correctly if you add content via CIFS and then you might have to resort to using the event model.



If we, for example, add a rule that enforces a specific document naming convention for a particular folder hierarchy, then this rule will work okay from the Alfresco Explorer and the Alfresco Share user interfaces where it will prohibit the user from adding the document by throwing an exception. However, via CIFS the document will still be added even though an exception was thrown.

In these cases, we can use the Alfresco event model that enables us to execute Java or JavaScript code when an event happens in the system. In Alfresco, these events are called behavior policies.

The events that we can listen to are associated with the service that triggers them. The Content Service events can be found in the `org.alfresco.repo.content.ContentServicePolicies` class and look like this:

Event (Behavior policy)	Description
<code>onContentUpdate</code>	Called when one or more node properties are updated. If you need to know what properties were updated, and the before and after value of properties during an update, then use the <code>onContentPropertyUpdate</code> method instead.
<code>onContentPropertyUpdate</code>	Called once for each node property that has been updated. The before update and after update value for the property is available.
<code>onContentRead</code>	Called when a content reader is requested for a node that has content. Could, for example, be used to keep statistics for when a document was last accessed, how many times it has been accessed, and so on.

The Copy Service events can be found in the `org.alfresco.repo.copy.CopyServicePolicies` class:

Event (Behavior policy)	Description
<code>getCopyCallback</code>	Called just before the copying of the node to find out what should be copied. By default, all aspects, the type, properties, and associations will be copied. However, sometimes it is necessary to be able to customize what should be copied and not copied. For example, if some properties should not be copied, then implement this method and exclude these properties from being copied. When implementing this method all aspects, associations, and properties that should be copied need to be specified.

Event (Behavior policy)	Description
<code>beforeCopy</code>	Called once it has been decided which properties and aspects will be copied, but before the copy occurs. This allows us to remove cached data based on the destination node, before it is overwritten. You are unable to make changes to what gets copied though, that must be done earlier via a <code>getCopyCallback</code> .
<code>onCopyComplete</code>	Called when the copying has completed (including any cascading).

The Asynchronous Action Execution Queue events can be found in the `org.alfresco.repo.action.AsynchronousActionExecutionQueuePolicies` class:

Event (Behavior policy)	Description
<code>onAsyncActionExecute</code>	Called when an asynchronous action has completed execution and transaction has committed. This event is not linked to a content type or aspect, but to a service implementation.

The Node Service events can be found in the `org.alfresco.repo.node.NodeServicePolicies` class and there are a lot of them. Therefore, we can divide them up into the following groups so that it is easier to overview them:

- Store events
- Node events
- Aspect events
- Association events

The following node events are related to the store:

Event (Behavior policy)	Description
<code>beforeCreateStore</code>	Called before a new store is created.
<code>onCreateStore</code>	Called just after a new store has been created.

The following node events are related to any node and its properties:

<code>beforeCreateNode</code>	Called before a new node and its parent association is created.
<code>onCreateNode</code>	Called just after a new node and its parent association has been created. Note that this event method is called before <code>onCreateChildAssociation</code> and <code>onUpdateProperties</code> .
<code>onMoveNode</code>	Called when a node has been moved. Note that this method is not called if the node is moved to another store such as from Working Store to Archive Store.
<code>beforeUpdateNode</code>	Called before a node is updated in the following situations: • Adding or removing aspects • Adding, removing, or updating properties • Adding or removing associations • Updating type
<code>onUpdateNode</code>	Called when a node is updated in the following situations: • Just before adding aspects • Just after removing aspects • Just after adding, updating, or removing properties • Adding or removing associations • Updating type
<code>onUpdateProperties</code>	Called when a node's properties are updated in these situations: • Just after adding, updating, or removing properties • Just after a node has been created
<code>beforeDeleteNode</code>	Called before a node is deleted. If the node has children, then this method will be called before each child is deleted. This method will also be called before a node is moved and the old source node is about to be deleted.
<code>onDeleteNode</code>	Called after a node has been deleted. If the node has children, then this method will be called after each child is deleted. This method will also be called after a node is moved and the old source node has been deleted.

The following node events are related to manipulation of a node aspect:

<code>beforeAddAspect</code>	Called before an aspect is added to a node.
<code>onAddAspect</code>	Called after an aspect has been added to a node.
<code>beforeRemoveAspect</code>	Called before an aspect is removed from a node.
<code>onRemoveAspect</code>	Called after an aspect has been removed from a node.

The following node events are related to manipulation of a node association:

beforeCreate NodeAssociation	Never called. (Should be called before a <i>parent</i> <-> <i>child</i> or peer association is created).
onCreate NodeAssociation	Never called. (Should be called after a <i>parent</i> <-> <i>child</i> or peer association has been created).
beforeCreate ChildAssociation	Called before a <i>parent</i> <-> <i>child</i> association is created in the following situations: • When a <i>parent</i><-><i>child</i> association is directly created • When a <i>parent</i><-><i>child</i> association is indirectly created as a result of a new node being created • When a <i>parent</i><-><i>child</i> association is indirectly created as a result of a node being moved
onCreate ChildAssociation	Called after a <i>parent</i> <-> <i>child</i> association has been created in the following situations: • When a <i>parent</i><-><i>child</i> association is directly created • When a <i>parent</i><-><i>child</i> association is indirectly created as a result of a new node being created • When a <i>parent</i><-><i>child</i> association is indirectly created as a result of a node being moved
beforeDelete ChildAssociation	Called before a <i>parent</i> <-> <i>child</i> association is deleted in the following situations: • When a <i>parent</i><-><i>child</i> association is directly deleted • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of a node being deleted • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of a node being moved • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of an aspect being removed from a node
onDelete ChildAssociation	Called after a <i>parent</i> <-> <i>child</i> association has been deleted in the following situations: • When a <i>parent</i><-><i>child</i> association is directly deleted • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of a node being deleted • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of a node being moved • When a <i>parent</i><-><i>child</i> association is indirectly deleted as a result of an aspect being removed from a node

onCreate	Called after a peer association has been created.
Association	
onDelete	Called after a peer association has been deleted.
Association	

The Records Management events can be found in the `org.alfresco.module.org_alfresco_module_dod5015.RecordsManagementPolicies` class:

Event (Behavior policy)	Description
beforeRMActionExecution	Called before a records management action executes.
onRMActionExecution	Called when a records management action has been executed.
beforeCreateReference	Called before creation of a custom reference between two components of a record, such as between an e-mail and its attachment.
onCreateReference	Called after the creation of a custom reference between two components of a record, such as between an e-mail and its attachment.
beforeRemoveReference	Called before removal of a custom reference between two components of a record, such as between an e-mail and its attachment.
onRemoveReference	Called after a custom reference has been removed between two components of a record, such as between an e-mail and its attachment.

The Version Service events can be found in the `org.alfresco.repo.version.VersionServicePolicies` class:

Event (Behavior policy)	Description
beforeCreateVersion	Called before a new version is created for a document. Also called before the version history is checked.
onCreateVersion	Called immediately before the new version node is created. Use it to determine what the versioning policy for a particular type may be.
afterCreateVersion	Called after the version has been created and after any associations to, for example, root version has been created.
calculateVersionLabel	Called when the version label should be calculated. Implement this method to do version numbering and labeling in a custom way.

Defining custom business logic to be executed when any of these events occur also requires knowledge of the content model that is implemented. When an event handler is registered with the system, the following things need to be specified:

- **Event method name:** QName for the event method (for example, {http://www.alfresco.org}onCreateNode)
- **Content model class:** QName for type or aspect that should be affected (for example, {http://www.bestmoney.com/content/model/1.0}meeting)
- **Java method name:** Name of the method that implements the business logic that should be executed when the event happens

This makes it possible to pinpoint exactly what content should be affected by the business logic when the event occurs. Besides being able to bind the business logic to a particular content model class (that is, type or aspect), it can also be bound to either a content model association or a content model property.

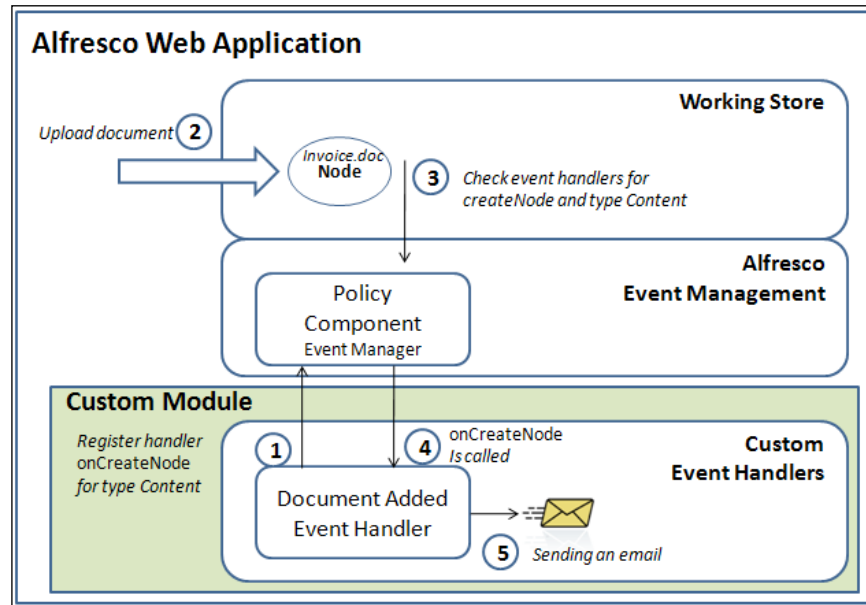
When an operation such as adding a document is executed, it is done in a transaction. During this, any registered event handlers are called in the same transaction and in the order they were registered. Because of this, a faulty event handler could prohibit the system from working properly. Let's say, we install an event handler that triggers when content is added to the system (that is, onCreateNode) and we have made a mistake when coding it.

If this coding mistake results in an exception being thrown, then that rolls back the transaction. This would effectively block anybody from adding content to the system. So if unexpected errors happen after installing a lot of event handlers, it might be a good idea to remove them and see if they are the cause of the problem.

It is possible to do even more fine-tuning of where in the transaction event handlers should be called (that is, compared to just before or after an operation). There are three different stages (Alfresco calls it notification frequencies) where the custom handler could be configured to be invoked:

- **EVERY_EVENT:** This is the default if the notification frequency is not specified. The event handler is then just executed wherever it is being invoked in the code. The name of this notification frequency implies that the event handler will be called multiple times, but that is not the case.
- **TRANSACTION_COMMIT:** The event handler is queued and invoked at the end of the transaction after it has been committed. A proxy around the event handler manages the queuing.
- **FIRST_EVENT:** The event handler is invoked just after the transaction is started. A proxy around the event handler manages this.

The following figure shows an example of how an event handler works:



Here, we have added an event handler that will be called whenever a new document is added to the repository. When that happens, we send an e-mail. The sequence is as follows:

1. The event handler, `onCreateNode`, is registered with the Alfresco Event Manager to be triggered for any documents of type `cm:content`.
2. Someone uploads a document.
3. This triggers the **Policy Component** to check if there are any registered event handlers for the `onCreateNode` event.
4. The **Policy Component** finds one event handler and calls the `onCreateNode` method in our custom code.
5. Our custom code sends an e-mail from the `onCreateNode` method.



When using the event manager it is important to know that it manages synchronous events. So whatever code we implement in the event handler will be executed in the same transaction as the main Alfresco code that triggered the event. This means that the code in the event handler will impact the performance of general Alfresco operations such as adding a document. So we need to be careful about this and use this event system only when it is really necessary.

Metadata extraction

The idea with metadata extraction is to automatically do some classification of content when it is added to the Alfresco repository. The extracted properties are stored together with the physical content as metadata. The more metadata that is stored with a piece of content, the more will be the search possibilities that exist.

The whole idea with a content management system is to be able to manage content in an easier way and metadata plays a big role in that. Metadata helps with:

- **Search:** As it gives the possibility to search on individual properties or combinations of properties.
- **Faster access:** Metadata is usually indexed, which means that the time it takes to search for content is reduced.
- **Sorting:** In Alfresco, documents can be tagged to belong to a certain type of content. For example, you could tag all documents that have to do with, for example, running with the word "running". It is then very easy to find and sort all documents that have to do with running.
- **Management:** If, for example, there is a piece of custom code that manages meeting documents in some way, then it would be very easy for that code to find relevant documents, if they have proper meeting metadata.
- **Rights:** Content rights management could be handled via metadata.

Metadata extractors are used to automatically extract properties from different content formats. Alfresco comes with a number of metadata extractors that are used by default without us having to do anything:

- **PDF:** Extracts the author (as `cm:author`), title (`cm:title`), subject (`cm:description`), and created (`cm:created`) from PDF files using the Apache PDFBox library.
- **MS Office:** Extracts the author (as `cm:author`), title (`cm:title`), subject (`cm:description`), and `createdDateTime` (`cm:created`), and `lastSaveDateTime` (`cm:modified`) from Microsoft Office documents (97-2003, 2007) using the Apache POI library. There are more MS Office document properties that can be extracted and saved as metadata, but they are not at the moment. The following properties could also be saved as metadata: `comments`, `editTime`, `format`, `keywords`, `lastAuthor`, `lastPrinted`, `osVersion`, `thumbnail`, `pageCount`, and `wordCount`.
- **MSG Email:** Extracts the `sentDate` (as `cm:sentDate`), `originator` (`cm:originator`), `addressee` (`cm:addressee`), `addressees` (`cm:addressees`), and `subjectLine` (`cm:subjectline`) from an Outlook e-mail (that is, .msg) using Apache POI.

- **HTML:** Extracts the author (as `cm:author`), title (`cm:title`), and description (`cm:description`) from HTML files using the Apache PDFBox library.
- **OpenOffice:** Extracts the creator (as `cm:author`), title (`cm:title`), description (`cm:description`), and `creationDate` (`cm:created`), from OpenOffice.org documents using the Apache Tika library. There are more Open Office document properties that can be extracted and saved as metadata, but they are not at the moment. The following properties could also be saved as metadata: `date`, `generator`, `initialCreator`, `keyword`, `language`, `printDate`, `printedBy`, and `subject`.
- **MIME Email:** Extracts the `messageFrom` (as `imap:messageFrom`), `messageTo` (`imap:messageTo`), `messageCc` (`imap:messageCc`), `messageSubject` (`imap:messageSubject`), `messageSent` (`imap:dateSent`), `Thread-Index` (`imap:threadIndex`), `Message-ID` (`imap:messageId`), and `date` (`imap:dateReceived`) from a MIME e-mail (that is, .eml in RFC822 format) using "JavaMail".
- **StarOffice** (Oracle Open Office): Extracts the author (as `cm:author`), title (`cm:title`), and description (`cm:description`) from Oracle Open Office documents.
- **DWG:** Extracts the author (as `cm:author`), title (`cm:title`), and description (`cm:description`) from drawings (that is, .dwg) produced by several CAD packages such as AutoCAD, IntelliCAD, and Caddie. The Apache Tika library is used for this. There are also a few other drawing properties that are available to set as metadata: `keywords`, `comments`, and `lastauthor`.

There are also experimental metadata extractors such as the MP3 extractor that you would have to manually turn on to test out.

Those metadata extractors that have the possibility to extract a few more properties – than are mapped to metadata – can be configured to set these extra properties to metadata as well. When there is no metadata extractor available for a file type, a custom extractor can be written and plugged into the system. For more information, see http://wiki.alfresco.com/wiki/Metadata_Extraction.

Content transformation

Content transformers are an important part of the Alfresco content management system as they enable content to be indexed by Lucene. All content that should be indexed, first needs to be converted to text files, which is done with transformers.

Content transformers are also very useful when you want to create new content formats for publishing, or the like. For example, when a Word document has been approved, we might want to automatically create a PDF version of it. Another useful feature of transformers is that they can be used to convert images into different formats and sizes.

The Alfresco system comes with a number of content transformers out of the box:

- **Any text to plain text:** Converts any textual format to plain text. For example, `text/xml` to `text/plain` or `application/x-javascript` to `text/plain` (used primarily for indexing).
- **PDF to plain text:** Converts PDF files to plain text files using the Apache PDFBox library (used primarily for indexing).
- **Excel to plain text:** Converts Microsoft Excel (version 97-2003, 2007) files to plain text files using the Apache POI library (used primarily for indexing).
- **Word to plain text:** Converts Microsoft Word (version 97-2003, 2007) files to plain text files using the TextMining library (used primarily for indexing).
- **HTML to plain text:** Converts HTML documents to plain text files using the HTML Parser library (used primarily for indexing).
- **E-mail (Outlook) to text:** Converts Microsoft Outlook e-mails (that is, `.msg`) to plain text files using the Apache POI library (used primarily for indexing).
- **E-mail (MIME) to text:** Converts RFC822 MIME e-mails (that is, `.eml`) to plain text files using the Java Mail library (used primarily for indexing).
- **MediaWiki markup to HTML:** Converts MediaWiki Markup pages to HTML documents using the Java Wikipedia API library.
- **PDF to image:** Converts a PDF file to a PNG image using the Sun PDF Renderer library or the Apache PDFBox library. Converts a PDF file to a JPEG or GIF Image using the ImageMagick tool.

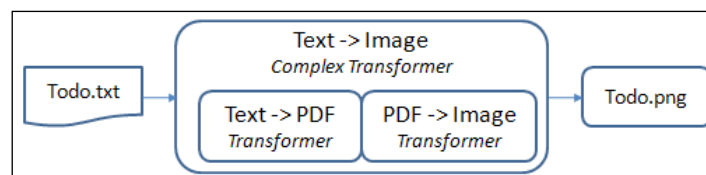
This transformer can actually transform to a lot of different image formats supported by the ImageMagick tool (over 100) including DPX, EXR, GIF, JPEG, JPEG-2000, PhotoCD, and TIFF (used for thumbnails).

- **Image to image:** Converts an image to a different size or format via the ImageMagick tool.
- **Open Office to PDF:** Converts OpenDocument, PDF, RTF, Word, Excel, PowerPoint, and Flash files into PDF files using the JODConverter and OpenOffice.org.
- **Open Office to image:** Converts Open Office documents into images using the Open Office to PDF transformer and the PDF to image transformer (used for thumbnails).

- **Plain Text to PDF:** Converts a plain text file into a PDF file using the Apache PDFBox library.
- **Plain Text to image:** Converts a plain text file into an image using the Plain Text to PDF transformer and the PDF to Image transformer.

When transformers are configured and set up, this can be done in a couple of ways to reach the end transformation goal. As we have seen in the previous list, most transformers convert from one format to the other such as from PDF to plain text.

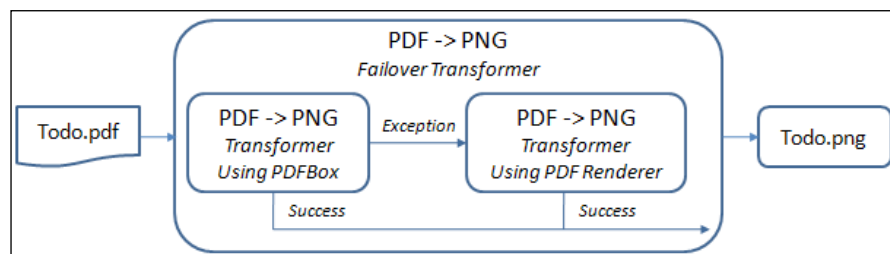
However, transformers can also be combined to solve a more complex transformation such as a plain text to Image transformation where two or more transformers work together. These transformers are called **complex transformers** and here is an example of how they work:



In this example, a text document is transformed into a PNG image by first being transformed to a PDF file.

If we are not sure whether one particular transformer can always successfully transform one format to the other, we can chain together the so-called **failover transformers**. Only one of these transformers will do the transformation. When the first transformer gets the job to transform, it will either respond with successful transformation or an exception. If a transformer responds with an exception, then the turn goes to the next transformer in the chain to see if it can do the transformation successfully.

The following example shows two transformers in a chain that can do PDF to PNG transformations:



In some cases, the transformation we would like to do is not possible via a Java Library but only via a command-line executable file. We can then use a special transformer called a **runtime executable content transformer** and it can be used to run a tool such as ImageMagick or OpenOffice.org from the command line.

For more information, see http://wiki.alfresco.com/wiki/Content_Transformations.

Alfresco Management Beans (JMX)

The core platform contains management beans (JMX) that can be used to configure Alfresco when it is running and also to inspect the health of the running system. By using a standard JMX Console such as JConsole that supports JSR 160 (that is, JMX Remoting) you can:

- Manage and control the subsystems
- Change the log level to, for example, debug for some part of the system
- Turn on and off file servers such as CIFS
- Set the server in read-only mode
- Set the server in single user mode
- Prevent further logins
- View user session tickets

Some of these features are really useful such as the possibility to turn on debug logging for a specific component without having to stop the server. It might not even be possible to stop the server whenever we want in a production system. Setting up the system in *read-only* mode is also very useful if, for example, you need to take a snapshot backup of the system's current state for offline debugging.

Application Programming Interfaces (APIs)

There are several APIs that you can use when extending the platform such as the low-level Java Foundation Services API or the JavaScript API. When you implement Java extensions delivered in the form of an **Application Module Package (AMP)**, you would mostly use the so-called Foundation Service API, but when you implement Web Scripts and Business Rules it is more convenient to use the JavaScript API.

If you access the repository from a remote application using a client-server approach then there are several REST-based APIs such as the Repository API and the CMIS API that can be used. The higher-level APIs use the Foundation Service API that has transaction management and security built in.

See the next chapter for more information on how to use these APIs.

Subsystems

Subsystems are configurable modules responsible for a piece of functionality in the Alfresco content management system. The functionality is often optional such as the IMAP server or can have several different implementations, such as authentication.

A subsystem can be thought of as a mini-server that runs embedded within the main Alfresco server. A subsystem has the following characteristics:

- It can be started, stopped, and configured independent of the Alfresco server
- It has its own isolated Spring application context and configuration

Subsystems are independent processes that can be brought up and down. This design lets an administrator of the system change a single configuration without having to bring down the entire Alfresco system. The advantages are reliability and availability.

Examples of Alfresco subsystems include:

- **Audit:** Configuration of audit parameters
- **Authentication:** Contains different authentication subsystems such as LDAP
- **E-mail:** SMTP support for sending e-mails
- **File servers:** CIFS, FTP, and NFS servers
- **IMAP:** Internal IMAP server
- **Open Office transformations:** Helps converting office documents to text
- **Synchronization:** LDAP synchronization settings
- **Sys admin:** It allows real-time control across some general repository parameters
- **Third-party:** Owns the SWFTools and ImageMagick content transformers
- **WCM development receiver:** A built-in WCM deployment target for local AVM to DM content deployments

Bootstrap

There are several ways to bootstrap the system with custom functionality or new content.

Patches

In some situations, we want to add something to the system just once during installation and then never do it again. And we want to do this in a controlled way, specifying from what version of the system it is applicable. This is called **patching** the system or **bootstrapping** the system. Alfresco uses patches to handle different things such as:

- Database upgrades
- Template installations
- Folder creation
- Permission updates
- Group imports, and so on

Every time we do a new Alfresco installation, or an upgrade, the logs will show what patches were executed. Every patch execution is logged in the database with information about whether it was successful or not. If an error occurred, then the database will contain an error message about it. If a patch did not succeed, then Alfresco will try and execute it every time you start the system until it is successful.

We can also set in what order patches should be executed, which is important, as many a times one patch depends on another patch's updates. Patches are implemented as Java classes and it is possible to create custom patches (we will have a look at that in the next chapter).

Importers

An **importer** component is also used to import data into the repository in the same way as a patch is. However, it is different from a patch—in that the outcome of executing it is not logged in the database and to control the execution order we have to use the Spring depends-on configuration. Importer components are also not usually written in Java. An importer component can be used to import the following things into the repository via XML files:

- Users
- Groups
- Scripts

- Presentation templates
- Folder hierarchies
- Documents

Data that should be imported can first be exported via the export command-line tool provided by Alfresco, which produces XML files that can be loaded into the repository via an importer component. So it might make sense to first create what you want to import via the user interface and then export it. Next chapter shows an example of how to configure an importer component.

Extension modules

Extension modules are used to extend Alfresco with significant new functionality, such as records management. Extension Modules are delivered in so-called **Application Module Packages (AMP)** files.

The following extension modules are available from Alfresco:

- Web Content Management (WCM) AMP
- Records Management (RM) AMP
- SharePoint Protocol Support (VTI) AMP
- Alfresco Forms Development Kit (FDK) AMP
- Lotus Quickr Integration AMP
- MediaWiki Integration AMP

Third-party extension modules

A lot of companies develop extensions for Alfresco and deliver them in AMP modules. The following is a list of some available modules:

- **Thumbnails AMP:** Adds thumbnails to Alfresco Explorer
- **Enterprise Reporting AMP:** Enables running reports from the Alfresco environment
- **OpsMailmanager AMP:** Enterprise e-mail management
- **Alfresco Bulk Filesystem Import:** Loads content from local filesystem and can preserve modified date during import (see *Chapter 8, Document Migration Solutions* for more information)

User interface clients

There are a couple of different user interfaces that you can use to access Alfresco either from a browser or from the filesystem.

Alfresco Explorer

Alfresco Explorer is the traditional Document Management client that most people use when they access the repository via a web interface. It has access to the complete repository and also offers administration functionality to handle, for example, users, groups, import, and export.

This client is successively being replaced by the Alfresco Share client, which has a much nicer and richer user interface.

However, there are situations when the Alfresco Share client does not yet support all functionality that Alfresco Explorer provides such as when using advanced workflows or creating and managing web content. In these cases, we have to still use the Alfresco Explorer client.

Likewise, the Alfresco Explorer platform does not support the collaboration and sharing features that are available in Alfresco Share.

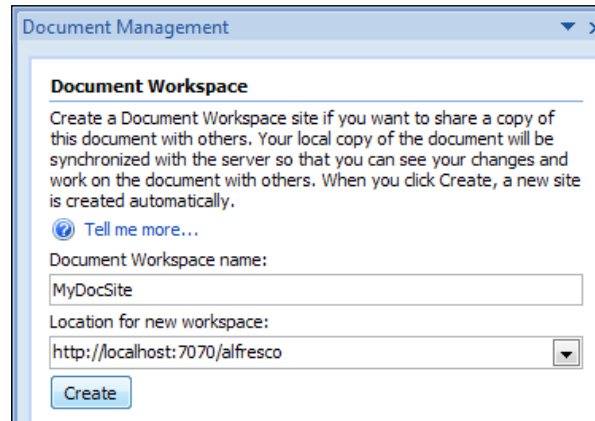
Alfresco Share

The Alfresco Share client is the new client for Alfresco that started out life as a pure collaboration and sharing platform. After a while, it became very popular and people wanted to use it for more than just collaboration and sharing features. Alfresco responded by adding more functionality to be able to access the document repository via Alfresco Share. Now it also has a lot of the administration features previously only available in Alfresco Explorer.

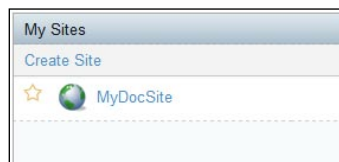
Alfresco SharePoint

This is not really a standalone client, but more an integration with Microsoft SharePoint functionality. When the SharePoint protocol is enabled in Alfresco, a Microsoft Office program can connect to Alfresco thinking it is connecting to SharePoint.

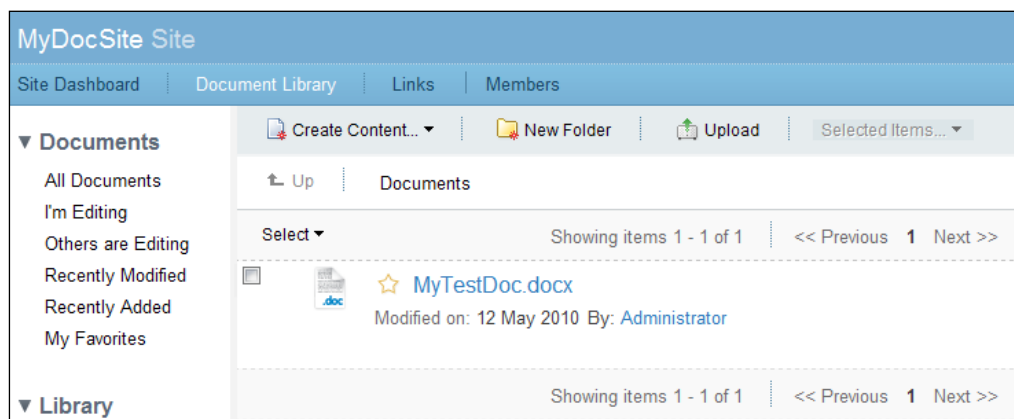
We can connect to Alfresco from, for example, MS Word as follows:



Here, we are creating a new Document Workspace that will be created as an Alfresco Share site called **MyDocSite**. We can then save documents directly into Alfresco without having to go through any other Alfresco user interface. This is how it looks in Alfresco Share after creating the workspace and saving a Word 2007 test document called `MyTestDoc.docx`:



Clicking on this site and then navigating to the document library shows the following:





Unfortunately we cannot save documents anywhere we like in the Repository, only to an Alfresco Share site.

Alfresco Mobile

Alfresco comes with a special web application to support the iPhone. It presents a smaller interface adjusted for the iPhone to be able to access content in an Alfresco Share site.

The interface gives you access to Alfresco Share sites and the possibility to search, view, and edit documents in the Document Library within sites. Users can also view tasks and activities.

For more information about mobile application solutions see *Chapter 14, Mobile Phone Access Solutions*.

Alfresco CIFS

The possibility to access files in Alfresco via a shared drive is one of the major benefits with Alfresco, as most people are used to working with a shared drive already. So the transition to a content management system becomes less cumbersome when users can work in the way they have always done.

The shared drive access is probably the interface that most people use on a day-to-day basis. However, there are things that cannot easily be done when using the CIFS interface:

- Searching for documents based on metadata
- Enter metadata for added documents
- Display error messages from rules, some rules will not even work when documents are added via this interface
- Workflow task management

For more information about the CIFS interface see *Chapter 5, File System Access Solutions*.

The Alfresco installation directory structure

After you have installed Alfresco, you will have a directory structure that looks something like this:

```
| - Alfresco
|
| - /alf_data
| - /amps
| - /amps-share
| - /bin
| - /extras
| - /ImageMagick
| - /install_extension
| - /licenses
| - /mysql
| - /OpenOffice.org
| - /tomcat
| - /virtual-tomcat
| - alf_start.bat      (Starts Apache Tomcat and Alfresco)
| - alf_stop.bat       (Stops Apache Tomcat and Alfresco)
| - alfresco.log       (Default location for Alfresco log file)
| - virtual_start.bat  (Starts another Tomcat for WCM preview)
| - virtual_stop.bat   (Stops Apache Tomcat for WCM preview)
```

It is important to get to know the directory structure of an Alfresco installation. So you know where to go and look for certain information or to be up to date when you get questions from clients.

The alf_data directory

The Alfresco data directory is the most important directory as it contains all the content files and all the content indexes of the repository, for both live content and deleted content. The location of this directory is specified in the `alfresco-global.properties` file located in the `tomcat/shared/classes` directory with the property `dir.root`.

This is the `alf_data` directory structure:

```
| - Alfresco
|   | - /alf_data
|     | - /audit.contentstore
|     | - /backup-lucene-indexes
```

```
| - /contentstore  
| - /contentstore.deleted  
| - /lucene-indexes  
| - /mysql  
| - /oouser
```

The contentstore directory

The `contentstore` directory contains all the live content files. You will notice, as you start clicking down in the directory hierarchy, that you will not recognize any of the filenames.

For example, if you upload a file called `mytext.doc` to a folder, you will not find any file by that name in the `contentstore` directory. It has been stored under a different name that looks more like a reference number.

To find a specific document you have to first go into **Alfresco Node Browser** and look up the item and its `cm:content` field. This field looks something like this:

```
contentUrl=store://2010/2/12/18/1/62781911-635d-4366-80dc-  
13d4a3e5e4fe.bin|mimetype=application/pdf|size=375530|encoding=utf-  
8|locale=en_GB_
```

And in this case the PDF file would be found under: `alfresco/alf_data/contentstore/2010/2/12/18/1`.

And it would be called: `62781911-635d-4366-80dc-13d4a3e5e4fe.bin` (that is, `UUID.bin`). If you change the extension to `.pdf` you will be able to open it.

File versioning

Every version is stored in its entirety. No delta data is kept. You can look up the content store location for the master version, such as:

```
contentUrl=store://2010/2/16/11/18/d3278bd6-2ad0-4d85-b627-  
0ce757b985a8.bin
```

And next to it you will see the `.bin` files for the other versions.

The contentstore.deleted directory

Whenever you delete a file from the Alfresco repository via any of the user interfaces, it is not physically deleted from disk, instead it is moved to an archive store. This store can be found on disk under the `contentstore.deleted` directory.

If you do not see any files in the `contentstore.deleted` directory, then your Alfresco installation might have a custom Content Cleanup Listener configuration.

The audit.contentstore directory

Alfresco can be configured to audit all changes to metadata and content and the audit trail is stored in this directory. This directory will usually be empty in a standard installation as audit logging is not turned on by default.

The lucene-indexes and backup-lucene-indexes directories

All content in the repository is indexed via the Lucene index engine (unless you have turned off indexing for some types of content). The index files are kept in the `lucene-indexes` directory.

If the index ever gets corrupted or deleted, it is possible to do a full re-indexing by setting the `index.recovery.mode` property to `FULL` and restart the system. This property can be found in the `repository.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco` directory. To do a full re-indexing every time you start Alfresco, even if it has been re-deployed, put this property in the `alfresco-global.properties` file.

Lucene indexes are backed up to the `backup-lucene-indexes` directory every night at 3 a.m. by a scheduled job.

The mysql directory

If Alfresco is installed from the "Full Installation" file, then MySQL can be installed at the same time. In this case, the data files for MySQL are stored in this directory.

The oouser directory

If Alfresco is installed from the "Full Installation" file, then OpenOffice.org can be installed at the same time. In this case, the user who executes the document conversions uses this directory.

The amps directories

The application modules (that is, AMPs) that are used to extend Alfresco can be found in the following two directories:

- | - `Alfresco`
- | - `/amps`
- | - `/amps-share`

The `amps` directory is used for modules that extend the Alfresco Explorer web application (that is, `alfresco.war`) and the `amps-share` are used for modules that extend the Alfresco Share web application (that is, `share.war`).

After putting the AMP files in these directories, you can use the `alfresco/apply_amps.bat` file to install them into the WAR file.

After a full installation of Alfresco 3.3, the following AMPs can be found in the directories:

```
| - Alfresco3.3
  | - /amps
    |
    | - alfresco-dod5015.amp      (Alfresco RM for Explorer UI)
    | - alfresco-quickr-unsupported.amp (Lotus Integration)
    | - vti-module.amp           (MS SharePoint Protocol)
  | - /amps-share
    | - alfresco-dod5015-share.amp (Alfresco RM for Share UI)
```

The tomcat directory

This is the main application server directory. When you install Alfresco from the "Full installation" file, it includes an Apache Tomcat installation that ends up in this directory.

The `tomcat` directory structure looks as follows:

```
| - Alfresco
  | - /tomcat
    | - /bin      (binaries to start Tomcat)
    | - /conf     (Tomcat configuration, configure SSL for example)
    | - /endorsed
    | - /lib       (Tomcat libraries)
    | - /logs     (Tomcat log files, except Alfresco log)
    | - /shared   (Config files that lives over deployments)
    | - /temp     (Temporary files such as for EHCACHE)
    | - /webapps  (Alfresco webapps go here)
    | - /work     (JSP compiled into Servlets)
```

The two most important directories here are the `shared` and the `webapps` directories. The `shared` directory is where you will do all Alfresco configuration that should live over deployments and Alfresco upgrades. Basically, try and always do the configuration here because then you know it is not going to be overwritten when somebody installs a new AMP or does an upgrade.

The `webapps` directory is where you will find the web application WAR files for Alfresco Explorer (that is, `alfresco.war`), Alfresco Share (that is, `share.war`), and Alfresco Mobile (that is, `mobile.war`).

You will also use files in the `bin` directory when you want to install Alfresco as a Windows service.

Getting the Alfresco source code

If you do not yet have the Alfresco source code downloaded, it is time to do so now.

You can access the trunk from `svn://svn.alfresco.com/alfresco/HEAD` and it is a good idea to update it every week to see what new stuff is being added. More information about the Alfresco development environment can be found here http://wiki.alfresco.com/wiki/Alfresco_SVN_Development_Environment.

The Alfresco database

Normally, we do not have to bother about the database, but there are situations when it is necessary to be familiar with it.

In general, one should not do any CRUD operations directly against the database bypassing the foundation services when building a custom solution on top of Alfresco. This will cause the code to break in the future if the database design is ever changed. Alfresco is required to keep older APIs available for backward compatibility (if they ever change), so it is better to always use the published service APIs.

Query the database directly only when:

- The customization built with available APIs is not providing acceptable performance and you need to come up with a solution that works satisfyingly
- Reporting is necessary
- Information is needed during development for debugging purposes
- For bootstrapping tweaking, such as when you want to run a patch again

Almost any RDBMS database can be used as the platform that uses Hibernate as the database access layer. For Level 2 caching the EHCACHE library is used, which speeds up the performance.

DB schema

The Alfresco database schema cannot be found in any .sql file, as the whole database is created via Hibernate the first time Alfresco is started after installation.

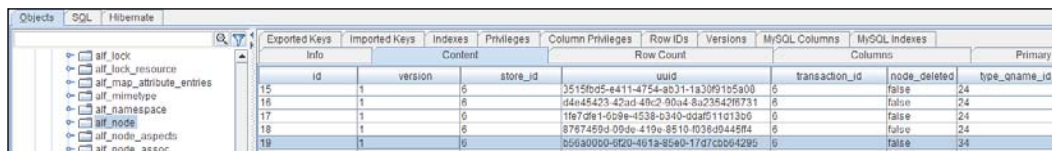
To access the tables use a tool like the **Squirrel SQL Client** or just use the **mysql command-line utility**. The database is called `alfresco` by default and to access it via JDBC an URL that looks like `jdbc:mysql://localhost:3306/alfresco` can be used. Default *username/password* is *alfresco/alfresco*.

Significant tables

Let's take a look at some of the tables that we might come in contact with or need to query for information.

ALF_NODE

This is the parent table for node metadata and many other tables refer to it with a foreign key. Listing a couple of rows from it looks like this:



The screenshot shows the Squirrel SQL Client interface. On the left, a tree view lists database objects, including 'alf_node'. The main pane displays the 'Info' tab for the 'alf_node' table. The table has columns: id, version, store_id, uuid, transaction_id, node_deleted, and type_name_id. The 'Row Count' is 6. The 'Columns' tab shows the same columns. The 'Primary Key' tab shows that 'id' is the primary key. Below the tabs, a list of rows is shown, with the first row having id=15, version=1, store_id=6, uuid=3515b025-e411-4754-8031-1a309105a00, transaction_id=6, node_deleted=false, and type_name_id=24.

id	version	store_id	uuid	transaction_id	node_deleted	type_name_id
15	1	6	3515b025-e411-4754-8031-1a309105a00	6	false	24
16	1	6	d4e48423-42ad-48c2-90a4-8a29542f6731	6	false	24
17	1	6	1fe7dfe1-6b9e-4538-b340-00af511013b0	6	false	24
18	1	6	8767459d-09de-419e-8510-f036d0445ff4	6	false	24
19	1	6	b56a0000-9f20-461a-8580-17d7cb664295	6	false	24

What we get is the node UUID, version, what store it is saved in, type QName, and so on. The ID of the rows is used to look up related rows in other tables, such as associated properties.

To select a particular node after looking up its node UUID (that is, from the `{http://www.alfresco.org/model/system/1.0}node-uuid` property) via the Alfresco Explorer Node Browser, we can execute the following query:

```
select * from alf_node where uuid='456822c7-8f3e-4129-9804-dfaaab54f47a'
```

This will give us the node ID to use for further queries.

ALF_NODE_PROPERTIES

This table contains all the properties that have been set as metadata for a particular node.

When we have the node ID, we can query for the associated properties as follows:

```
select * from alf_node_properties where node_id=1016
```

This table contains all the properties that have been set as metadata for a particular node:

node_id	boolean_value	long_value	float_value	double_value	string_value	qname_id
19	false	0	0	0	invite-email.ftl	27
19	false	0	0	0	invite-email.ftl	28
19	false	0	0	0	Email template used to generate the invite email for Alfresco ...	29
19	false	4	0	0	<null>	34
19	false	0	0	0	<null>	35
19	true	0	0	0	<null>	36

We can still see only the values for the properties not what their names are. The name of the property can be looked up via the qname_id.

Notice also that some properties like the default Created Date, Creator, Modifier, and Modified date are not listed. This is because they are part of the ALF_NODE row.

ALF_NODE_ASPECTS

This table contains all aspects that are associated with a node. When we query this table, we get the following result:

node_id	qname_id
19	25
19	31
19	35
19	37

This is not very helpful as we cannot see the names of the aspects, just their QNames. We would have to link up with the ALF_QNAME table to see the names.

ALF_QNAME

This table contains all the QName definitions and it is referred to from lots of the other tables. For example, here is how to use it together with the ALF_NODE_ASPECT table:

```
select a.*, q.local_name from ALF_NODE_ASPECTS a, ALF_QNAME q where NODE_ID=19 and a.qname_id=q.id
```

Rows 4: select a.*, q.local_name from ALF_NODE_ASPECTS a, ALF_QNAME q where NODE_ID=19 and a.qname_id=q.id

node_id	qname_id	local_name
19	25	auditable
19	31	titled
19	35	author
19	37	inlineeditable

In this way, we can clearly see what aspects are associated with a node.

ALF_APPLIED_PATCH

This table contains information about all executed patches. It keeps information about if they were successful or not and any error messages:

```
select id, description, applied_on_date, was_executed, succeeded from alf_applied_patch
```

Limited to 100 rows: select id, description, applied_on_date, was_executed, succeeded from alf_applied_patch

id	description	applied_on_date	was_execut..	succeed...
patch.actionRuleDecouplingPatch	Migrate existing rules to the updated model where rules are d...	2010-12-02 15:14:59.0	false	true
patch.actions.scheduledFolder	Creates the scheduled actions folder in the Data Dictionary.	2010-12-02 15:15:07.0	false	true
patch.authorityDefaultZonesPatch	Adds groups and people to the appropriate zones for wcm, sh...	2010-12-02 15:15:05.0	false	true
patch.authorityMigration	Copies any old authorities from the user store to the spaces s...	2010-12-02 15:15:05.0	false	true
patch.avmFormPropertyIdentifier	Reindex wca:webform to make wca:formname an identifier	2010-12-02 15:15:02.0	false	true
patch.AVMGuidPatch	Set GUIDs on AVM nodes.	2010-12-02 15:15:00.0	false	true
patch.AVMLayeredSnapshot	Set indirectionVersion on Layered Nodes.	2010-12-02 15:15:00.0	false	true
patch.avmStoreAsIdentifier	Reindex wca:webfolder to make wca:avmstore an identifier	2010-12-02 15:15:02.0	false	true
patch.webReindexPermissions	Reindex wca:webpermissions to make wca:webpermissions an identifier	2010-12-02 15:15:02.0	false	true

Example queries and update statements

The queries are some examples used in real content management deployments.

Querying for number of nodes of a certain type

Let's say you wanted to find out how many e-mails have been stored in the repository so far. Then you could do that with the following query:

```
SELECT count(*) from ALF_NODE n, ALF_NODE_ASPECTS a, ALF_QNAME q where
n.ID = a.NODE_ID and a.QNAME_ID=q.ID and q.LOCAL_NAME='imapemail';
```

This query will search for all content nodes that have the aspect `imapemail` applied and count them.

Querying for number of nodes stored in a particular month

If you wanted to build on the previous query and find out how many e-mail nodes have been stored in a particular month, you could do that as follows:

```
SELECT count(*) from ALF_NODE n, ALF_NODE_ASPECTS a, ALF_QNAME q where
n.ID = a.NODE_ID and a.QNAME_ID=q.ID and q.LOCAL_NAME='imapemail' and
n.audit_created like '2010-01%';
```

This will query for the number of e-mail nodes that have been stored in January 2010.

Running a patch again

If you, for some reason, wanted to run a patch again, you can do that as follows:

```
UPDATE ALF_APPLIED_PATCH SET WAS_EXECUTED=0, SUCCEEDED=0
WHERE ID='patch.applyImapMailboxAspect';
```

This will set up the patch to status not successfully run last time and the next time we start Alfresco, it will execute the patch again.

Summary

This chapter has taken us through most of the features of the Alfresco platform and by now we should be familiar with repository concepts such as stores, nodes, associations, aspects, and types. Everything in the Alfresco repository is represented as a node. If we have business rules that should apply to nodes in the repository, then these can be enforced by implementing the so-called content rules.

Document properties can be automatically extracted with Metadata extractors while the document is being added to the repository. When a document is uploaded to the repository, content transformers can be configured to convert between different formats, such as from an MS Word document to a PDF. Transformers are also used to convert any non-text format to text, so that content can be easily indexed by the Lucene search engine.

If we wanted to send an e-mail every time a document was added anywhere in the repository, then we could use the internal event management system and define a policy for the "add content" event.

Alfresco comes with quite a few user interfaces out of the box, where the traditional JSF based one is called Alfresco Explorer and offers management screens for all features available in Alfresco. This user interface will be replaced by a newer one called Alfresco Share, which is currently the most developed UI. Alfresco CIFS is used to emulate a shared drive for easy migration of users into using Alfresco.

Alfresco uses subsystems for functionality that require many different implantations, such as authentication, or for optional functionality like support for IMAP. The Alfresco system can be bootstrapped with either patches or importers. Patches are used extensively by Alfresco to do, for example, database upgrades.

We also had a look at the directory structure of an Alfresco installation and one of the most important folders is the `alf_data` folder that contains the content and the Lucene index. The Alfresco web application is contained under the `alfresco/tomcat/webapps/alfresco` folder in an installation.

Finally, some of the more important database tables were examined and a couple of SQL query examples were shown, and this should give us more confidence in how things work together and we can also create reports directly against the database if necessary.

Now you are probably eager to get going and do some coding. The next chapter introduces the application programming interfaces that can be used to access the repository. For each API, there will be a number of code examples that you can dig into.

2

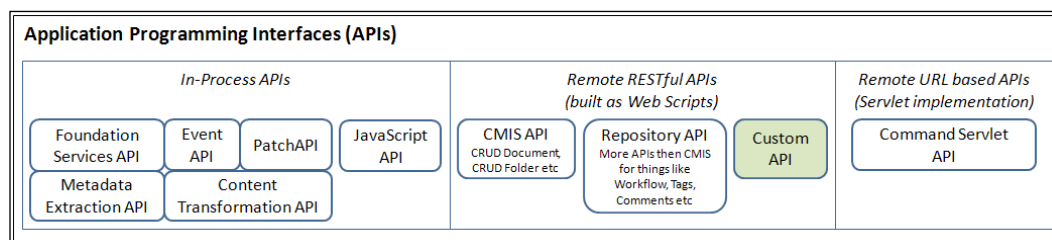
The Alfresco APIs

The Alfresco platform has many different **Application Programming Interfaces (APIs)** that we can use to access content in the repository or extend the content management functionality. To utilize the full potential of Alfresco, it is important to know how each API fits into the system and how we can use it.

Alfresco runs on a Java platform, so most of the APIs that we have available are naturally going to be Java-based APIs. However, one API that is used a lot is the **JavaScript API** as it hides much of the complexities found with other APIs.

Also, when we use any of the REST-based APIs, we could use any language we wanted on the client side, which makes this approach really flexible and scalable.

The following figure gives an overview of the available APIs:



The following list gives a short description of each API:

- **Foundation Services API:** A low-level Java-based API that most other APIs use
- **Event API:** A Java-based API used when we want to trigger custom code based on events
- **Patch API:** A Java-based API that can be used to bootstrap the system in a controlled way

- **Metadata Extractor API:** A Java-based API that is useful when we want to extend the system with more metadata extractors
- **Content Transformation API:** A Java-based API used to build new content transformations
- **JavaScript API:** A server-side JavaScript used by Web Scripts and Rules
- **CMIS API:** A standard non-lock REST-based API
- **Repository API:** Alfresco's proprietary API that contains extensions to CMIS to deal with tags, comments, workflow, and other Alfresco specifics
- **Custom API:** Custom remote APIs that we build
- **Command Servlet API:** A Servlet-based API based on the command and strategy design patterns

There are two other APIs that are not depicted in the figure, but we might come across them when searching for Alfresco information:

- **JCR API:** This is the Content Repository for Java Technology API as specified in JSR 170 and 283. It now looks like CMIS is used more and more instead of this API.
- **Web Service API:** Alfresco also provides a web service interface but the REST-based interfaces seem to be used more.

We will not cover the JCR and Web Service APIs in this book; however, the Alfresco 3 Web Service book covers the Web Services API.

In this chapter, you will learn:

- About the In-process and Remote APIs
- Using the Java Foundation Services API
- Using JavaScript to manage content
- Implementing event handlers
- Creating a bootstrap patch
- Configuring a bootstrap importer component
- What a Web Script is and how to create one
- What CMIS is

Application Programming Interfaces (APIs)

There are several APIs that you can use when extending the platform such as the low-level Java Foundation Services API or the JavaScript API. When you implement Java extensions delivered in the form of an AMP, you would mostly use the so-called Foundation Service API, but when you implement business rules it is more convenient to use the JavaScript API.

If you access the repository from a remote application using a client-server approach, then there are several REST-based APIs that can be used such as the Repository API and the CMIS API. The higher level APIs use the Foundation Service API, which has transaction management and security built in.

In-process APIs

These are application programming interfaces that require the client to be in the same process context as the Alfresco server. Typically, the client code is delivered in an **Alfresco Module Package (AMP)**.

The Java Foundation Services API

Knowing the Foundation Service API inside out will help you a lot when developing Alfresco customizations as most other APIs and modules are built on top of these low-level services.

The API is designed as a group of services and we will take a look at some of the most popular ones. It is used only when you develop extensions that are merged with the Alfresco web application archive (WAR) via an AMP.

Configuration and Transaction Management

To use one of the Foundation Services, such as the Node Service, it is injected via Spring into the custom bean that wants to use it. There are two ways of doing this — one way is to inject the so-called *public* version of the service and the second is to inject the *internal* version of the service.

The difference between these versions is that the public version comes with audit management, transaction management, and security management interceptors configured. So, whenever we call a method inside a public service, a transaction is automatically created for us and permissions checked plus any auditing that has been configured is done.

So why would we want to use anything other than the public service despite the fact that it comes with everything pre-configured for us? Some of the reasons are:

- To control the transaction boundaries
- To check permissions ourselves when necessary
- To make service calls as fast as possible

The first reason is probably the most important one as it can cause problems if not handled properly. Let's say you want to store a file and set an aspect for it with the Node Service. This can be done in three steps:

1. Store the metadata.
2. Store the physical content.
3. Apply the aspect.

Now, let's say that there is an error in the system between any of these steps. This would lead to inconsistent data as the transaction boundaries are between each call. So when building a new module or customization on top of the foundation services it is best to manage the transaction boundaries ourselves.

In the following example, we inject the internal version of the Node Service into our custom bean called `myCmsService`:

```
<?xml version='1.0' encoding='UTF-8'?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">
  <bean id="myCmsService"
    class="org.springframework.transaction.interceptor.
      TransactionProxyFactoryBean">

    <property name="proxyInterfaces">
      <value>com.bestmoney.test.MyCmsService</value>
    </property>

    <property name="target">
      <bean class="com.bestmoney.test.MyCmsServiceImpl">
        <property name="nodeService">
          <ref bean="nodeService"/>
        </property>
      </bean>
    </property>
  </bean>
</beans>
```

```

    <property name="transactionAttributeSource">
        <bean class="org.springframework.transaction.annotation.
            AnnotationTransactionAttributeSource"/>
    </property>
</bean>
</beans>

```

 To inject the public version of the service, we would just change the first letter to uppercase as follows: `<ref bean="NodeService"/>`.

Any other foundation service is injected in the same way. To control the transaction boundaries, we configured the bean to use the Spring Transaction Service by using the Transaction Proxy and the transaction annotations. The `com.bestmoney.cms.MyCmsService` interface looks as follows with transaction annotations for a "write content" method and a "read content" method:

```

@Transactional(readonly = true)
public interface MyCmsService {

    public void readSomething();

    @Transactional(readonly = false,
        propagation = Propagation.REQUIRED)
    public void writeSomething();
}

```

 When we use a declarative transaction management and put the transaction annotation at the interface level (that is, `@Transactional(readonly = true)`), it becomes the default transactional semantics for all methods unless overridden via explicit method level annotation (that is, `@Transactional(readonly = false, Propagation.REQUIRED)`).

The implementation class for this interface looks like this:

```

public class MyCmsServiceImpl implements MyCmsService
{
    private NodeService m_nodeService;

    public void setNodeService(NodeService nodeService)
    {
        m_nodeService = nodeService;
    }
}

```

```
public void readSomething() {
    // Calls that only read data go here and
    // they are executed inside a read-only transaction.
}

public void writeSomething() {
    // Here we can use methods that require a
    // read/write transaction.
}
}
```

When we use the foundation service API, this is the way in which we always define the custom beans. So, when the different methods of the foundation services are demonstrated in this section, we will assume that we have this custom bean set up.

Using the Node Service and the Content Service

The Node Service and the Content Service are probably the most used foundation services as they are used to manipulate the repository nodes and their content in all the possible ways.

Here is how to create a new file node:

```
public void writeSomething()
{
    String filename = "helloworld.txt";
    NodeRef parentFolderNodeRef = // get parent node ref
    ChildAssociationRef parentChildAssocRef = null;
    QName associationType = ContentModel.ASSOC_CONTAINS;
    QName associationQName = QName.createQName(
        NamespaceService.CONTENT_MODEL_1_0_URI,
        QName.createValidLocalName(filename));
    QName nodeType = ContentModel.TYPE_CONTENT;

    Map<QName, Serializable> nodeProperties =
        new HashMap<QName, Serializable>();
    nodeProperties.put(ContentModel.PROP_NAME, filename);

    parentChildAssocRef = m_nodeService.createNode(
        parentFolderNodeRef, associationType,
        associationQName, nodeType, nodeProperties);
}
```

This creates metadata for a file `helloworld.txt` that will be uploaded or created sometime in the future.

 If we wanted to create a node that is not visible from any of the user interfaces, we could create it with the type set to `ContentModel.TYPE_CMOBJECT` instead of `ContentModel.TYPE_CONTENT`.

The `org.alfresco.model.ContentModel` interface contains the `QName` definitions for all types, aspects, properties, associations, and so on that are available in the standard Alfresco content model. So when we do not have our own content model, this one can be used.

 A new node can be created only as a subnode to a parent node, so we have to have a node reference to a parent folder node to be able to add a new node. We could get this by searching for it with Lucene for example, but this is not shown in this particular example. This example also assumes that we have write permission to the parent folder node.

We can get the file node reference from the *parent->child* association:

```
NodeRef newFileNodeRef = parentChildAssocRef.getChildRef();
```

As we said, the new file node is not yet associated with any physical content, so let's fix that:

```
Boolean updateContentPropertyAutomatically = true;
ContentWriter writer = m_contentService.getWriter(newFileNodeRef,
    ContentModel.PROP_CONTENT, updateContentPropertyAutomatically);
writer.setMimetype(MimetypeMap.MIMETYPE_TEXT_PLAIN);
writer.setEncoding("UTF-8 ");
String fileContent = "Hello World!";
writer.putContent(fileContent);
```

This will automatically set the node's `cm:content` property to point to the `helloworld.txt` file.

Here, we are just adding a piece of text as content and it automatically creates the physical file in the Content Store. If we already have the file and an input stream for it, then we could add it as content for the node like this:



```
File file = new File("helloworld.text");
InputStream fis = new FileInputStream(file);
writer.setMimetype(MimetypeMap.MIMETYPE_TEXT_PLAIN);
writer.setEncoding("UTF-8");
writer.putContent(fis);
```

The input stream `fis` is closed automatically, so no need to close it after the file is stored in the repository. We could also just skip the input stream step and call the `putContent(file)`.

If we want to add a title and description property to the new file node, we can do that by adding the `Titled` aspect:

```
Map<QName, Serializable> aspectProperties =
    new HashMap<QName, Serializable>();
aspectProperties.put(ContentModel.PROP_TITLE, "Hello World!");
aspectProperties.put(ContentModel.PROP_DESCRIPTION,
    "This is the traditional Hello World example.");
m_nodeService.addAspect(newFileNodeRef,
    ContentModel.ASPECT_TITLED, aspectProperties);
```

To create a folder node, we can do the following:

```
String folderName = "MyFolder";
associationType = ContentModel.ASSOC_CONTAINS;
associationQName = QName.createQName(
    NamespaceService.CONTENT_MODEL_1_0_URI,
    QName.createValidLocalName(folderName));
nodeType = ContentModel.TYPE_FOLDER;

nodeProperties = new HashMap<QName, Serializable>();
nodeProperties.put(ContentModel.PROP_NAME, folderName);

ChildAssociationRef childAssocRef =
    m_nodeService.createNode(parentFolderNodeRef, associationType,
        associationQName, nodeType, nodeProperties);
```

As we can see, there is no big difference between creating a file node and creating a folder node. The main difference is the type you give the node. It is when we add content that it differs, as only a file node has content.

If we wanted to get all file and subfolder nodes for a node, and then for each child node check what type and properties they have, we could use the following code:

```
public void readSomething() {
    List<ChildAssociationRef> childAssocRefs =
        m_nodeService.getChildAssocs(parentFolderNodeRef);

    if (childAssocRefs.isEmpty())
    {
        return;
    }

    for (ChildAssociationRef childAssocRef : childAssocRefs)
    {
        NodeRef childNodeRef = childAssocRef.getChildRef();
        QName nodeTypeQName = m_nodeService.getType(childNodeRef);
        Map<QName, Serializable> nodeProperties =
            m_nodeService.getProperties(childNodeRef);

        if (nodeTypeQName.equals(ContentModel.TYPE_CONTENT))
        {
            // Do something with the file metadata or content
            String filename = (String)nodeProperties.get(
                ContentModel.PROP_NAME);
        }
        else if (nodeTypeQName.equals(ContentModel.TYPE_FOLDER)) {
            // Do something with folder metadata
            Date createDate = (Date)nodeProperties.get(
                ContentModel.PROP_CREATED);
        }
    }
}
```

If we have the node reference for a folder or a file and want to get to the parent folder containing it, we can use the following code:

```
public void readSomething() {
    NodeRef fileOrFolderNodeRef = // get this from somewhere
    NodeRef parentFolderNodeRef =
        m_nodeService.getPrimaryParent(fileOrFolderNodeRef).getParentRef();
}
```

To check if a node has a particular aspect applied to it, do this:

```
public void readSomething() {
    NodeRef fileOrFolderNodeRef = // get this from somewhere
    Boolean isVersioned = m_nodeService.hasAspect(
        fileOrFolderNodeRef, ContentModel.ASPECT_VERSIONABLE);
}
```

In this case, we are checking if a node has versioning turned on. To read the physical content file associated with a node, use the following code:

```
public void readSomething() {
    NodeRef fileNodeRef = // get this from somewhere
    ContentReader reader = m_contentService.getReader(
        fileNodeRef, ContentModel.PROP_CONTENT);

    if (reader == null) {
        // Maybe it was a folder after all
        return;
    }

    InputStream is = reader.getContentInputStream();

    try {
        // Read from the input stream
        String contentText = IOUtils.toString(is, "UTF-8");
    } catch (IOException ioe) {
        logger.error(ioe);
        throw new RuntimeException(ioe);
    } finally {
        if (is != null) {
            try {
                is.close();
            } catch (Throwable e) {
                logger.error(e);
            }
        }
    }
}
```

When reading content for a node, it is very important to close the input stream after we are done, as this is not done automatically. If we forget it, we will soon have eaten up all file descriptors available.

When we want to remove nodes, we do as follows:

```
public void writeSomething() {
    m_nodeService.deleteNode(newFileNodeRef);
    m_nodeService.deleteNode(newFolderNodeRef);
}
```

This removes both the metadata from the database and also moves the physical file from the Working Store to the Archive Store.

Using the File Folder service

The File Folder service is a higher level interface than the Node Service and Content Service interfaces. The implementation of the File Folder Service uses most of the other low-level services such as the Node Service, Content Service, Dictionary Service, Copy Service, and so on.

This service can be good to use if you don't need to manipulate nodes in as much detail as with the Node Service and the Content Service. When using the File Folder Service, we think more in terms of files and folders.

Here is how to create a new file with the File Folder service:

```
public void writeSomething() {
    String filename = "helloworld2.txt";
    NodeRef parentFolderNodeRef = // get parent node ref
    QName nodeType = ContentModel.TYPE_CONTENT;

    FileInfo file = m_fileFolderService.create(
        parentFolderNodeRef, filename, nodeType);
}
```

This creates metadata for a file `helloworld2.txt` that will be uploaded or created sometime in the future. We can see how differently this was done with the Node Service, where we also had to set up what association we wanted with the parent node and what properties should be set up for the new node. Using the File Folder service, we also get a `FileInfo` object as return type instead of an association reference that we got when using the Node Service.

The `FileInfo` object gives us access to things like the node reference for the newly created file node:

```
NodeRef newFileNodeRef = file.getNodeRef();
```

To create some content for the file node with the File Service, do this:

```
ContentWriter writer = m_fileFolderService.getWriter(
    newFileNodeRef);
writer.setMimeType(MimetypeMap.MIMETYPE_TEXT_PLAIN);
writer.setEncoding("UTF-8");
String fileContent = "Hello World!";
writer.putContent(fileContent);
```

This will automatically set the node's `cm:content` property to point to the `helloworld2.txt` file. We can begin to see a pattern—in that the File Folder Service is doing a lot of stuff under the covers, so we get less code to maintain.



The File Folder Service cannot be used to add an aspect to a node. Use the Node Service for this.

To rename a file with the File Folder Service is easy:

```
String newName = "HelloWorld2renamed.txt";
try {
    m_fileFolderService.rename(newFileNodeRef, newName);
}
catch (FileNotFoundException fnfe) {
    logger.info("FileFolderService: Could not rename (
        " + filename + ") to (" + newName + "), file was not found
        [fileNodeRef = " + newFileNodeRef + "] [parentFolderRef=" +
            parentFolderNodeRef + "]);
}
```

To create a folder node, we can do the following:

```
String folderName = "MyFolder";
NodeRef parentFolderNodeRef = // get parent node ref
QName nodeType = ContentModel.TYPE_FOLDER;

FileInfo folder = m_fileFolderService.create(
    parentFolderNodeRef, folderName, nodeType);
```

If we wanted to get all files and subfolders for a parent folder, and then for each child check what type and properties they have, we could use the following code:

```
public void readSomething() {
    List<FileInfo> filesAndFolders = m_fileFolderService.list(
        parentFolderNodeRef);

    if (filesAndFolders.isEmpty()) {
        return;
    }

    for (FileInfo fileOrFolder : filesAndFolders) {
        if (fileOrFolder.isFolder()) {
            // Do something with the folder
            Date createdDate = fileOrFolder.getCreatedDate();
            Date modifiedDate = fileOrFolder.getModifiedDate();
        } else if (fileOrFolder.isLink()) {
            // Do something with a link to a folder or file
        } else {
        }
    }
}
```

```

        // Do something with a file
        ContentData contentInfo = fileOrFolder.getContentData();
        String mimetype = contentInfo.getMimetype();
        long size = contentInfo.getSize();
        String encoding = contentInfo.getEncoding();
    }
}

```

We can see here that the File Folder service is designed to work more with objects than with individual properties (which was the case with the Node Service), so we do not have to cast property values, for example, as there are specific methods to access the most used properties. It also has features to extract what is contained in the `cm:content` property that usually has a value that looks something like this:

```

contentUrl=store://2010/4/26/11/58/5d8df875-37a7-470d-ad79-
fa4001bf3a13.bin|mimetype=text/plain|size=1570|encoding=utf-
8|locale=en_US_

```

By using the `ContentData contentInfo = fileOrFolder.getContentData()` call, we get direct access to each of the values in this property.



It is also possible to list just folders or files in a parent folder with the `listFiles` and `listFolders` calls.

When we want to remove a file or folder, we do as follows:

```

public void writeSomething() {
    m_fileFolderService.delete(newFileNodeRef);
}

```

This removes both the metadata from the database and also moves the physical file from the Working Store to the Archive Store.

To read the physical content file associated with a file you can use the same code as we did with the Node Service, just change it and use the following call to get the reader:

```

ContentReader reader = m_fileFolderService.getReader(fileNodeRef);

```

The File Folder service can also be used to copy and move nodes:

```

public void writeSomething()
{
    try
    {
        String newName = null; // Do not change the name
    }
}

```

```
FileInfo fileCopy = m_fileFolderService.copy(
    fileNodeRef, destFolderNodeRef, newName);
FileInfo fileMoved = m_fileFolderService.move(
    fileCopy.getNodeRef(), destFolderNodeRef2, newName);
} catch (FileExistsException fee) {
    // File already exist in destination folder
} catch (FileNotFoundException fnfe) {
    // Destination folder could not be found
}
```

The File Folder service covers most of the functionality that one might need when working with files and folders in the repository. A recommendation is to use the File Folder service as much as possible and then resort to the more fine-grained services, such as the Node Service or the Copy Service, in those situations where it does not have enough features to perform a specific operation.

Using the Search Service

There are a couple of different ways that can be used to search for content within the repository. We can either use the Search Service with the LUCENE query language or XPATH query language, or we can use the Node Service to search with the XPATH query language.

The recommended approach is to use the Search Service with the LUCENE query language, as an XPATH query via the Node Service is too slow, and the XPATH query feature via the Search Service is not fully implemented.

In this book, we will use the LUCENE query language for all searching. Here is how to search for a user's home folder in the repository:

```
public void readSomething() {
    StoreRef workspaceStore = StoreRef.STORE_REF_WORKSPACE_SPACESSTORE;
    String query =
        "+PATH:\"/app:company_home/app:user_homes/sys:martin\"";
    ResultSet results = null;
    List<NodeRef> matchingNodes = null;

    try {
        results = m_searchService.query(workspaceStore,
            SearchService.LANGUAGE_LUCENE, query);
    } finally {
        if (results != null) {
            matchingNodes = results.getNodeRefs();
        } else {
            matchingNodes = new ArrayList<NodeRef>();
        }
    }
}
```

```

        // Underlying search engine, such as Lucene,
        // might have IO resources open so close them
        results.close();
    }

```

The preceding code is a standard template for how you can use the Search Service with Lucene queries. The `PATH` keyword is used for specifying a folder path in the repository where we want to start the search. This query will return just one node reference referring to the `martin` folder.



The folder path is not specified in a way that we might have expected such as `/Company Home/User Homes/martin`, which we can call the **display path**, but instead in a different format like `/app:company_home/app:user_homes/sys:martin`, which is actually an XPATH expression. So how can we get the XPATH from a display path?

We can use a combination of the File Folder Service, Node Service, and Namespace Service as follows (this also shows a different way to get to the Company Home node reference, instead of searching for it):

```

StoreRef workspaceStore =
StoreRef.STORE_REF_WORKSPACE_SPACESSTORE;
NodeRef storeRootNodeRef =
    m_nodeService.getRootNode(workspaceStore);

// Setup Company Home path (i.e. app:company_home)
QName companyHomePath =
    QName.createQName(m_companyHomeChildname, m_namespaceService);

// Get node reference for Company Home
List<ChildAssociationRef> assocRefs =
    m_nodeService.getChildAssocs(storeRootNodeRef,
        ContentModel.ASSOC_CHILDREN, companyHomePath);
NodeRef companyHomeNodeRef = assocRefs.get(0).getChildRef();

// Setup list of folders excluding the first one
List<String> folderList = new ArrayList<String>();
folderList.add("User Homes");
folderList.add("martin");

// Get the file information for the leaf folder
FileInfo folderInfo = null;
try {

```



```
        folderInfo = m_fileFolderService.resolveNamePath(
            companyHomeNodeRef, displayPathFolderList);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    // Get the XPath formatted path
    String xpath =
        m_nodeService.getPath(folderInfo.getNodeRef()).toPrefixString(
            m_namespaceService);
```

It is now just a question of refining and changing the query to what we want. The preceding query could easily be changed to return everything in the `martin` folder as follows:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin/*\"";
```

The star (*) is used as a wildcard and by adding it just after the forward slash, we can search for all content in the `martin` folder. This searches just one folder level directly under `martin`. To do a deep search, we can use a double forward slash as follows:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin//*\"";
```

This will return all the files and folders under the `martin` folder. Or more specifically, it will return any node under the `martin` folder as we have not specified what node types we are actually looking for.

To narrow down the search to just look for folders, we can change the query as follows:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin//*\"";
query +=
    "+TYPE:\"{http://www.alfresco.org/model/content/1.0}folder\"";
```

This will narrow down the search to include just folders by specifying the type of nodes we are looking for with the keyword `TYPE`. What if we did not want to have system folders included? We can achieve that by excluding that type as follows:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin//*\"";
query +=
    "+TYPE:\"{http://www.alfresco.org/model/content/1.0}folder\"";
query +=
    "-TYPE:\"{http://www.alfresco.org/model/content/1.0} systemfolder\"";
```

It is also possible to narrow down searches based on aspects. Let's adjust the query so we get back only non-system folders with versioning turned on:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin/*\"";
query +=
    "+TYPE:\"{http://www.alfresco.org/model/content/1.0}folder\"";
query +=
    "-TYPE:\"{http://www.alfresco.org/model/content/1.0}systemfolder\"";
query +=
    "+ASPECT:\"{http://www.alfresco.org/model/content/1.0}versionable\"";
```

Here, we used another keyword called ASPECT to specify what aspects a node should have to be included in the search result. Now, let's say we wanted to narrow down the search further and only look for folders with the name Test. To search on the `cm:name` property, add the following to the query:

```
String query =
    "+PATH:\"/app:company_home/app:user_homes/sys:martin/*\"";
query +=
    "+TYPE:\"{http://www.alfresco.org/model/content/1.0}folder\"";
query +=
    "-TYPE:\"{http://www.alfresco.org/model/content/1.0}systemfolder\"";
query +=
    "+ASPECT:\"{http://www.alfresco.org/model/content/1.0}versionable\"";
query += "+@" + LuceneQueryParser.escape(
    "{http://www.alfresco.org/model/content/1.0}name") + ":\\" +
    LuceneQueryParser.escape("Test") + "\"";
```

When searching on properties, we use the `@` character to specify the name of the property and the value of it. Remember to escape the property name and the property value to handle special characters such as `+ - ! ( ) { } [ ] ^ " ~ * ? : \`. For more information about search, see <http://wiki.alfresco.com/wiki/Search>.

Using the Permission Service

When we are using the internal versions of the Foundation Services, it is up to us to make sure that users are not accessing folders they are not supposed to access or create content where they should not, and so on. We can easily check a user's permissions with the Permission Service.

To check if a user has permission to add documents or folders in a parent folder, we can do the following:

```
public void readSomething() {
    NodeRef parentFolderNodeRef = // get node ref
    boolean hasCreatePermission =
        m_permissionService.hasPermission(parentFolderNodeRef,
            PermissionService.ADD_CHILDREN) == AccessStatus.ALLOWED;

    if (hasCreatePermission) {
        // Go ahead and add document or create folder...
    }
}
```

In this case, we are checking if the AddChildren permission group (that is, CreateChildren permission and LinkChildren permission) is allowed for the user in the parent folder with passed in node reference. We can check read, update, and delete permissions in the same way:

```
boolean hasReadPermission =
    m_permissionService.hasPermission(parentFolderNodeRef,
        PermissionService.READ) == AccessStatus.ALLOWED;

if (hasReadPermission) {
    // It is okay to display folder or document for user
}
```

By using the READ permission group, we check if the user can access properties of a node, access child nodes, and access the physical content of a node.

```
boolean hasUpdatePermission =
    m_permissionService.hasPermission(parentFolderNodeRef,
        PermissionService.WRITE) == AccessStatus.ALLOWED;

if (hasUpdatePermission) {
    // It is okay for user to go ahead and update document
}
```

By using the WRITE permission group, we check if the user can update properties of a node, and add physical content for a node.

```
boolean hasDeletePermission =
    m_permissionService.hasPermission(parentFolderNodeRef,
        PermissionService.DELETE) == AccessStatus.ALLOWED;

if (hasDeletePermission) {
    // User can delete folder or document
}
```

By using the DELETE permission group, we check if the user can delete the actual node or any of its child nodes.

In none of these calls to the Permission Service have we passed in a username, so what user is the permission check executed against? The Permission Service will use the user associated with the current thread.

Using the Dictionary Service

The Dictionary Service can be used to extract definitions from the content models that have been deployed in the system. It can be used to inspect the content models as follows:

```
public void readSomething()
{
    Collection<QName> allDeployedModels =
        m_dictionaryService.getAllModels();
    for (QName model : allDeployedModels) {
        ModelDefinition modelDef = m_dictionaryService.getModel(model);
        QName modelName = modelDef.getName();
        Collection<QName> allTypes =
            m_dictionaryService.getTypes(modelName);
        Collection<QName> allAspects =
            m_dictionaryService.getAspects(modelName);
        ...
    }
}
```

However, one of the more used functions of the Dictionary Service is probably the possibility to check if a node reference is of a certain type or a subtype. This can be done as follows:

```
NodeRef someNodeRef = // get node reference
QName nodeTypeQName = m_nodeService.getType(someNodeRef);

Boolean isFolder = m_dictionaryService.
    isSubClass(nodeTypeQName, ContentModel.TYPE_FOLDER);

if (isFolder) {
    // We got some type of folder node
}
```

This would match any folder including system folders, as they are a subtype of folders. If we wanted to check that a node type matches exactly a specific type then we can just use equals, as follows:

```
nodeTypeQName.equals(ContentModel.TYPE_FOLDER);
```

In this case, it will match only if the node type is a normal folder (`cm:folder`) and it will not, for example, match a system folder type (`cm:systemfolder`), even though it extends the normal folder type.

Logging

It is a good idea to include logging in our code. To set up a logging object we add the following to our implementation class:

```
public class MyCmsServiceImpl implements MyCmsService {
    private static Log logger =
        LoggerFactory.getLog(MyCmsServiceImpl.class);

    private NodeService m_nodeService;

    public void setNodeService(NodeService nodeService) {
        m_nodeService = nodeService;
    }

    public void readSomething() {
        // Here we can use methods that are read-only
    }

    public void writeSomething() {
        // Here we can use methods that require
        // a transaction that is not read-only
    }
}
```

Use `DEBUG` logging in most cases as writing `INFO` logs impacts performance. When writing `DEBUG` logs first test if the debug level is enabled or not:

```
if (logger.isDebugEnabled()) {
    logger.debug("A log with " + var1 + " a lot " + var2 + "
        of stuff");
}
```

In this way, you will not have unnecessary string concatenation going on if debug is turned off. When writing logs make sure to include as much information as possible; avoid writing logs such as the following:

```
logger.warn("Access denied");
```

A better log would give detailed information about the problem:

```
logger.warn("Could not create new folder,
Add Children permission denied to user
"+AuthenticationUtil.getFullyAuthenticatedUser()+"
[folderNodeName= "+name+"] [parentFolderName="+parentName+
" [parentFolderNodeRef="+parentRef+"] ");
```

A system administrator is going to see this log, so it should contain as much information as possible to be able to track down the problem and fix it. Warning logs should be such in nature that the system can continue to function after they have happened.

Error logs should only be written when the problem is irrecoverable. When writing an error log, we would in most cases also throw a `RuntimeException` to roll back the ongoing transaction, so the system is kept in a consistent state:

```
public void writeSomething() {
    ...

    parentChildAssocRef = m_nodeService.createNode(
        parentFolderNodeRef, associationType, associationQName,
        nodeType, nodeProperties);

    if (parentChildAssocRef == null)
    {
        String msg = "Could not create file node (" + filename + "),
        createNode resulted in null
        [parentFolderNodeRef=" + parentFolderNodeRef + "
        [username=" + username + "];
        logger.error(msg);
        throw new RuntimeException(msg);
    }
}
```

It is highly likely that error logs will be picked up by a system monitoring tool and will alert the system administrator of a serious problem. So use error logs only if the problem is really serious.

Event management API

The event management system provided by Alfresco is quite useful when we are using the foundation services to code an embedded extension to Alfresco. With the vast number of events that we can listen to, this gives us a lot of possibilities to automate certain business rules that might be dependent on what happens in the system.

To create an event handler, we first start with the event handler implementation itself. In this example, we will listen to a couple of document events and do some logging when they occur:

```
public class DocumentEventHandler
{
    private static Log logger =
        LogFactory.getLog(DocumentEventHandler.class);

    private PolicyComponent m_eventManager;

    public void setPolicyComponent(PolicyComponent p)
    {
        m_eventManager = p;
    }

    public void setNodeService(NodeService nodeService) {
        m_nodeService = nodeService;
    }

    public void onAddDocument(ChildAssociationRef parentChildAssocRef)
    {
        NodeRef parentFolderRef = parentChildAssocRef.getParentRef();
        NodeRef docRef = parentChildAssocRef.getChildRef();

        // Check if node exists, might be moved,
        // or created and deleted in same transaction.
        if (docRef == null || !m_nodeService.exists(docRef)) {
            // Does not exist, nothing to do
            return;
        }

        logger.info("A new document with ref (" + docRef + ") was just
            created in folder (" + parentFolderRef + ")");
    }

    public void onUpdateDocument(NodeRef docNodeRef) {
        // Check if node exists, might be moved,
        // or created and deleted in same transaction.
        if (docNodeRef == null || !m_nodeService.exists(docNodeRef)) {
            // Does not exist, nothing to do
            return;
        }
    }
}
```

```

    NodeRef parentFolderRef =
        m_nodeService.getPrimaryParent(docNodeRef).getParentRef();

    logger.info("A document with ref (" + docNodeRef + ") was just
        updated in folder (" + parentFolderRef + ")");
}

public void onDeleteDocument(ChildAssociationRef
    parentChildAssocRef, boolean isNodeArchived)
{
    NodeRef parentFolderRef = parentChildAssocRef.getParentRef();
    NodeRef docRef = parentChildAssocRef.getChildRef();
    logger.info("A document with ref (" + docRef + ") was just
        deleted in folder (" + parentFolderRef + ")");
}
}

```

This just defines our event handling methods and we need to also register them with the Alfresco Event Manager, so that they get called when these events happens. To do this, let's create another method that will be used to register our event methods with the event manager:

```

public void registerEventHandlers() {
    m_eventManager.bindClassBehaviour(
        NodeServicePolicies.OnCreateNodePolicy.QNAME,
        ContentModel.TYPE_CONTENT,
        new JavaBehaviour(this, "onAddDocument",
            Behaviour.NotificationFrequency.TRANSACTION_COMMIT));
}

```

When you register an event handler, there are three things you need to tell the event manager:

- **Event:** What event method we want to listen to. In this case, we have told the event manager to notify our event handler when the `onCreateNode` event happens.
- **Content Model Class:** We also need to tell the event manager for what type of content it should notify us. In this case, we want to be notified when *any* type of content is created (that is, any content of type `cm:content` or subtype thereof).

We are not restricted to use only types; we can also specify aspects such as anything with versioning turned on (that is, `ContentModel.ASPECT_VERSIONABLE`). If we want the event method to be called for several different aspects and types, then we have to call `bindClassBehaviour` several times.

- **Event Handler:** The last thing we need to do is to tell the event manager what method in our code makes up the actual event handler. In this case, it is the `onAddDocument` method that we want to be called when the `onCreateNode` event happens for a node of type `Content`.

The other two event handlers are registered in the same way:

```
m_eventManager.bindClassBehaviour(  
    NodeServicePolicies.OnUpdateNodePolicy.QNAME,  
    ContentModel.TYPE_CONTENT,  
    new JavaBehaviour(this, "onUpdateDocument",  
        Behaviour.NotificationFrequency.TRANSACTION_COMMIT));  
  
m_eventManager.bindClassBehaviour(  
    NodeServicePolicies.OnDeleteNodePolicy.QNAME,  
    ContentModel.TYPE_CONTENT,  
    new JavaBehaviour(this, "onDeleteDocument",  
        Behaviour.NotificationFrequency.TRANSACTION_COMMIT));
```

We have decided to bind our event handlers to a class event (that is, `Type` or `Aspect`). It is also possible to bind an event handler directly to a property event or an association event by using the `bindPropertyBehavior` or `bindAssociationBehavior` property respectively.



For example, to get notified only when the name (that is, `cm:name`) of a document changes, we could do this registration:

```
m_eventManager.bindPropertyBehaviour(  
    NodeServicePolicies.OnUpdatePropertiesPolicy.QNAME,  
    ContentModel.TYPE_CONTENT, ContentModel.PROP_NAME,  
    new JavaBehaviour(this, "onUpdateDocumentName",  
        Behaviour.NotificationFrequency.TRANSACTION_COMMIT));
```

Note that we have to use a policy method (that is, event) that is on the property level (that is, `onUpdateProperties`).

The last thing we need to do is to add the `DocumentEventHandler` class to the Spring context and make sure that the `registerEventHandlers` method is called:

```
<bean id="documentEventHandler"  
    class = "com.bestmoney.repo.policy.DocumentEventHandler"  
    init-method = "registerEventHandlers">  
    <property name="policyComponent">  
        <ref bean="policyComponent"/>  
    </property>  
</bean>
```

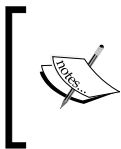
So how do we know what the event handler method signatures should look like? Take the following method signature for example:

```
public void onDeleteDocument(ChildAssociationRef
    parentChildAssocRef, boolean isNodeArchived) {
```

How would we know that the event manager expects to call a method with that signature? We can find out the correct method signature by looking in the `NodeServicePolicies` class:

```
package org.alfresco.repo.node;

public interface NodeServicePolicies {
    ...
    public interface OnDeleteNodePolicy extends ClassPolicy {
        public void onDeleteNode(ChildAssociationRef
            childAssocRef, boolean isNodeArchived);
    }
    ...
}
```



If you implement an extension based on the Foundation Service API and you also use the Event API, this cannot later be changed to use for example, the CMIS API as it does not provide an event model with callbacks, or the like.

Metadata Extraction API

The Metadata Extraction API can be used to extend Alfresco with new metadata extractors. Normally, we would not have to implement any new metadata extractors as the system comes with a lot of different extractors out of the box (as we have seen earlier). If you want to check out the metadata extractors installed in your system, open the `alfresco/tomcat/webapps/alfresco/WEB-INF/classes/alfresco/content-services-context.xml` file.

However, sometimes Alfresco does not come with a metadata extractor for a MIME type. This is the case with, for example, XML documents, as they can come in an infinite number of formats. However, let's say we have a customer with a specific XML format looking like this:

```
<document id="doc_id_2010_02_21_1002">
  <header>
    <date>2010-04-22</date>
    <title>Test text import</title>
    <desc>
```

```
        This is a description of this text...
    </desc>
</header>
<text>Lots of text go here...</text>
</document>
```

And they want the header fields extracted into some content model metadata fields each time an XML file with this format is added to the repository. The mapping from XML field to content model field should be as follows:

- date -> Created Date (that is, cm:created)
- title -> Title (that is, cm:title)
- desc -> Description (that is, cm:description)

To support this, we can implement a new metadata extractor. What we need when implementing a new metadata extractor is some sort of tool that can be used to extract the properties, which make up the metadata, from the document format. When we get that tool, in this case an XML parser, we can go ahead and create the class as follows:

```
public class SampleXMLMetadataExtractor extends
    AbstractMappingMetadataExtractor {

    private static final String KEY_CREATION_DATE = "date";
    private static final String KEY_TITLE = "title";
    private static final String KEY_DESCRIPTION = "desc";

    public static String[] SUPPORTED_MIMETYPES = new String[] {
        MimetypeMap.MIMETYPE_XML };

    public SampleXMLMetadataExtractor ()
    {
        super(new HashSet<String>(Arrays.asList(SUPPORTED_MIMETYPES)));
    }
}
```

The first thing we do is extend the `AbstractMappingMetadataExtractor` base class, so we get a lot of functionality for free, and this base class also implements the `MetadataExtractor` interface for us. Then we tell the abstract base class what MIME types this extractor will support. What we also define is the name of the document properties that we will extract. These names are used as keys in the Document Property Key -> Metadata Property Value map.

The next thing we need to do is implement the `extractRaw` method and this method should extract as many properties as possible from the XML document, even if all of them are not actually going to be mapped to any metadata in the content model:

```
@Override
public Map<String, Serializable> extractRaw(
    ContentReader reader) throws Throwable
{
    Map<String, Serializable> rawProperties = newRawMap();

    InputStream is = null;
    try
    {
        is = reader.getContentInputStream();

        // Use some magical tool to parse and extract
        // properties from XML file
        Date dateValue = // extract property
        String titleValue = // extract property
        String descValue = // extract property

        boolean wasAdded = putRawValue(KEY_CREATION_DATE, dateValue,
            rawProperties);
        wasAdded = putRawValue(KEY_TITLE, titleValue, rawProperties);
        wasAdded = putRawValue(KEY_DESCRIPTION, descValue,
            rawProperties);

    } finally {
        if (is != null) {
            try {
                is.close();
            } catch (IOException e) { /* log error */ }
        }
    }

    return rawProperties;
}
```

We now have an XML metadata extractor that will extract all the possible properties into this raw map of properties (that is, `rawProperties`). The system needs to know how to map them to content model properties. This is done in a properties file that we will need to create:

```
#
# SampleXMLMetadataExtractor - default mapping

# Namespaces
namespace.prefix.cm=http://www.alfresco.org/model/content/1.0

# Mappings
# XML file property -> Content Model Property
Date = cm:created
Title = cm:title
Desc = cm:description
```

For this mapping, the file that is to be picked up automatically by the system should have the same name as the class. However, the extension should be `.properties` (that is, `SampleXMLMetadataExtractor.properties`) and it should be put in the same package as the class. This is called the default property mapping and it can be overridden later on via Spring configuration if needed.

The last thing we need to do is create a Spring bean definition for the new XML metadata extractor, so that it is loaded the next time we start Alfresco:

```
<bean id="extracter.XML" class =
    "com.bestmoney.repo.content.metadata.SampleXMLMetadataExtractor"
    parent="baseMetadataExtractor" />
```

Make sure to set the `baseMetadataExtractor` as the parent bean, so that we get all the necessary bean definitions for the `AbstractMappingMetadataExtractor` class. This bean definition also calls the `register` method in the `MetadataExtractorRegistry` class, so the XML metadata extractor will be available next time anybody adds an XML document to the repository. The bean definition should be added to the `custom-metadata-extractors-context.xml` file. (After installation this file is called `custom-metadata-extractors-context.xml.sample`, so you need to rename it.)

If someone later on would like to override the property mappings, this can be done by overriding the bean definition as follows:

```
<bean id="extracter.XML" class =
    "com.bestmoney.repo.content.metadata.SampleXMLMetadataExtractor"
    parent="baseMetadataExtractor">
```

```

<!--Bring with us all the default mappings -->
<property name="inheritDefaultMapping">
    <value>true</value>
</property>

<!-- Override just one of the properties -->
<property name="mappingProperties">
    <props>
        <prop key="namespace.prefix.cm">
            http://www.alfresco.org/model/content/1.0
        </prop>
        <prop key="date">cm:modified</prop>
    </props>
</property>
</bean>

```

Here, we override the default mapping of date to Created Date and set it to be stored in the Modified Date instead. We can use the same technique with existing metadata extractors to override and change property mappings.

It is also possible to skip the property file all together and specify the property mappings in the bean definition as follows:

```

<bean id="extracter.XML" class =
    "com.bestmoney.repo.content.metadata.SampleXMLMetadataExtractor"
    parent="baseMetadataExtractor">

    <property name="mappingProperties">
        <bean class =
            "org.springframework.beans.factory.config.PropertiesFactoryBean">
            <property name="properties">
                <props>
                    <prop key="namespace.prefix.cm">
                        http://www.alfresco.org/model/content/1.0
                    </prop>
                    <prop key="date">cm:created</prop>
                    <prop key="title">cm:title</prop>
                    <prop key="desc">cm:description</prop>
                </props>
            </property>
        </bean>
    </property>
</bean>

```

Content Transformation API

The default content transformers are defined in the `alfresco/tomcat/webapps/alfresco/WEB-INF/classes/alfresco/content-services-context.xml` file (that is, the same file as the metadata extractors are defined in). And as we have seen, the system comes with lots of transformers and we can often combine them to create new transformations by using a complex transformer, so there are not that many cases when we would need to create our own transformer.

To make sure that there is no transformation available for what we want, first try to do the transformation via the Alfresco user interface and a transformation action. If it reports that the transformation cannot be done, then it is time to create our own transformer. This is the case when we try to convert an MS Visio diagram to a PDF file; it is not a supported transformation.

Most transformers use an external tool to do the transformation and there are plenty of command-line tools to convert between MS Visio (that is, `application/vnd.ms-visio`) and PDF (that is, `application/pdf`) that can be used for this. To create our own custom transformer that uses a command-line tool for the transformation, we use a proxy content transformer class as follows:

```
<bean id="transformer.visio2pdf" class=
  "org.alfresco.repo.content.transform.ProxyContentTransformer"
  parent="baseContentTransformer">
  <property name="worker">
    <ref bean="transformer.worker.visio2pdf"/>
  </property>
</bean>
```

The proxy transformer delegates to the real content transformer, the worker, and makes it possible to separate the third-party transformation tool from the transformer registry. The transformation worker is defined as follows and uses the runtime executable content transformer:

```
<bean id="transformer.worker.visio2pdf" class =
  "org.alfresco.repo.content.transform.
  RuntimeExecutableContentTransformerWorker">
  <property name="mimetypeService">
    <ref bean="mimetypeService"/>
  </property>

  <property name="explicitTransformations">
    <list>
      <bean class =
        org.alfresco.repo.content.transform.ExplicitTransformationDetails">
        <property name="sourceMimetype">
```

```
        <value>application/visio</value>
      </property>
      <property name="targetMimetype">
        <value>application/pdf</value>
      </property>
    </bean>
  </list>
</property>

<property name="checkCommand">
  <bean class="org.alfresco.util.exec.RuntimeExec">
    <property name="commandsAndArguments">
      <map>
        <entry key=".*">
          <list>
            <value>${visio2pdf.exe}</value>
            <value>-help</value>
          </list>
        </entry>
      </map>
    </property>
    <property name="errorCodes">
      <value>2</value>
    </property>
  </bean>
</property>

<property name="transformCommand">
  <bean class="org.alfresco.util.exec.RuntimeExec">
    <property name="commandsAndArguments">
      <map>
        <entry key=".*">
          <list>
            <value>${visio2pdf.exe}</value>
            <value>${source}</value>
            <value>-o</value>
            <value>${target}</value>
          </list>
        </entry>
      </map>
    </property>
    <property name="errorCodes">
```



```
        <value>2</value>
      </property>
    </bean>
  </property>

</bean>
```

Here we first define the transformation that we support – MS Visio to PDF – by setting the `explicitTransformations` property.

Then we define the `checkCommand` that is used to verify that the command-line tool is available and that the transformer will be able to function. The transformer will be disabled if the `checkCommand` returns an error (defined with `errorCode` property) or is inaccessible. With this configuration, we assume that the command-line tool can be run with the `-help` option to just check that it is available and working.

The last thing we define is the `transformCommand`, which does the transformation from MS Visio to PDF. The transformation mechanism performs substitutions of the variables `${source}` and `${target}`, which are the full file paths of the source and target files for the transformation. It is assumed here also that the command-line option `-o` is used to specify output/target files.

The command-line executable is specified as a variable, so we also need to add it to the repository properties file. Add the location of the `visio2pdf` executable to `custom-repository.properties` (can be found here `tomcat/shared/classes/alfresco/extension`).

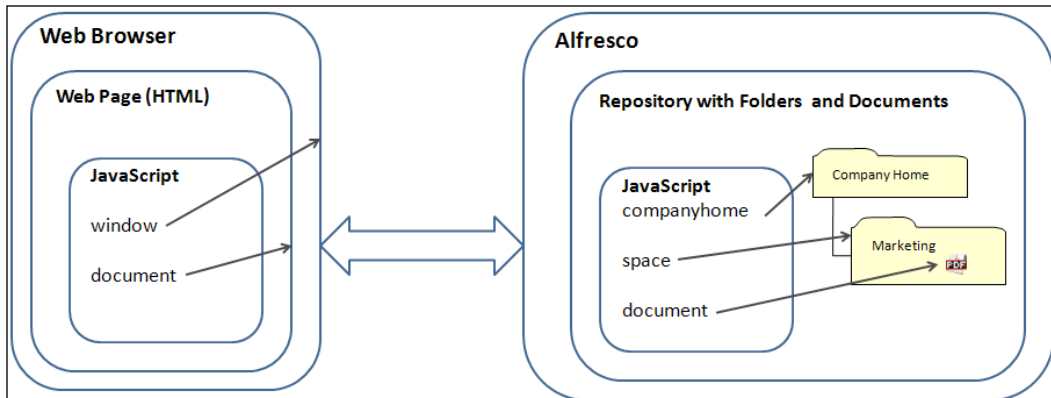
```
# External executable locations
visio2pdf.exe=/usr/bin/visio2pdf
```

Add the bean definitions to a file called `custom-content-transformers-context.xml` (this file does not exist, you need to create it) and put it in the `tomcat/shared/classes/alfresco/extension`.

The JavaScript API

The JavaScript API is one of the most useful programming interfaces for Alfresco. It allows us to quickly make customizations to the content management functionality. And it is easy to do this while the system is running because the development time is faster than if using Java.

It is though, important to know that this is server-side JavaScript implemented with the Mozilla Rhino JavaScript engine. It does not contain objects (that is, window, document, location, and alert) or methods for manipulating HTML documents:



Alfresco has extended the Rhino JavaScript implementation with some extra object such as **companyhome**, **space**, and **document**, as shown in the previous screenshot.

When using the JavaScript API, we implicitly use the Transaction Service and the Security Service, so if anything goes wrong the transaction is rolled back and the system is still in a consistent state.

An Alfresco JavaScript is usually called in the context of a space and a document, so these variables will contain references to the current folder (that is, space variable) and the current file (that is, document variable).

Here is how to create a new file node:

```
var filename = "helloworld.txt";
var file = space.createFile(filename);
```

This creates metadata for a file `helloworld.txt` that will be uploaded or created sometime in the future. The `space` variable will point to the folder where the script is executed.

We can get the file node reference for the new node:

```
var newFileNodeRef = file.nodeRef;
```

As we said, the new file node is not yet associated with any physical content, so let's fix that:

```
file.mimetype = "text/plain";
file.content = "Hello World!";
```

This will automatically set the node's `cm:content` property to point to the `helloworld.txt` file.



Here we are just adding a piece of text as content and it automatically creates the physical file in the Content Store. If we already have the file in the filesystem and want to read it, and set it as content for the metadata, we cannot do that with JavaScript as we are normally not allowed to access the filesystem.

In this case, upload the file first to the repository and then search for it as follows:

```
var myFile = companyhome.childByNamePath("/User Homes/
martin/mystuff.doc");
```

If this is not possible or you need to handle a lot of binary files, use the Java Foundation Services API instead.

If we want to add a title and description property to the new file node, we can do that by adding the `titled` aspect:

```
var props = new Array(2);
props["cm:title"] = "Hello World!";
props["cm:description"] =
    "This is the traditional Hello World example.";

file.addAspect("cm:titled", props);
```

To create a folder node, we can do the following:

```
var folderName = "MyFolder";
var folder = space.createFolder(folderName);
```

Here we create a folder under the parent folder from where the script was executed.

As we can see, there is no big difference between creating a file node and creating a folder node. It is when we add content that it differs, as only a file node has content.

If we wanted to get all file and subfolder nodes for a node, and then for each child node check what type and properties they have, we could use the following code:

```
var childNodes = space.children;
for each (childNode in childNodes) {
    var nodeRef = childNode.nodeRef;
    var nodeProperties = childNode.properties;
    var type = childNode.type;
```

```

    if (childNode.isDocument) {
        // Do something with the file metadata or content
        var filename = nodeProperties["cm:name"];
    } else if (childNode.isContainer) {
        // Do something with folder metadata
        var foldername = nodeProperties.name;
    }
}

```

When we run this script it will loop through all folders and files under the folder from which the script is run. For each node found, it gets the node reference, properties, and also the QName for the type of node. It then checks if it is a document or a folder and extracts the name of the content.

When we want to remove nodes, we do as follows:

```

file.remove();
folder.remove();

```

This removes both the metadata from the database and also moves the physical file from the Working Store to the Archive Store.

If we have the node reference for a folder or a file and want to get to the parent folder containing it, we can use the following code:

```

var parentFolder = file.parent;

```

To check if a node has a particular aspect applied to it, do this:

```

var isVersioned = file.hasAspect("cm:versionable");

```

In this case, we are checking if a node has versioning turned on. To read the physical content file associated with a node, use the following code:

```

var folder = companyhome.childByNamePath("/User Homes/martin");
var childNodes = folder.children;
for each (childNode in childNodes)
{
    if (childNode.isDocument) {
        if (childNode.mimetype == "text/plain") {
            var text = childNode.content;
        } else if (childNode.mimetype == "text/html") {
            var html = childNode.content;
        }
    }
}
}

```

Here we are accessing the content of all text and HTML files in the /Company Home/User homes/martin folder.

When using JavaScript, the security layer is enabled, so it is not possible to access or create files where we should not be allowed to. However, in some cases, it is good to be able to check if the user has permission to add a file to a folder, for example.

Here is how to do that:

```
if (space.hasPermission("CreateChildren"))
{
    // Ok, we can go ahead and create a folder or file
}
else
{
    logger.log("User (" + person.properties.userName + ") does not have
        permission to create a file in the (" + space.name + ") folder");
}
```

When checking permissions, we use the `hasPermission` method and pass in the permission group name. These are the lower level permission groups, as mentioned earlier (for example, `ReadProperties`, `ReadChildren`, `ReadContent`, `WriteProperties`, `WriteContent`).

To print out the username of the current user executing the script, we use another root object called `person` that references a `cm:person` object.

To search for nodes with JavaScript is easy and we can use Lucene queries:

```
var store = "workspace://SpacesStore";
var query = "+PATH:\"/app:company_home/*\"
    +ASPECT:\"cm:versionable\"";
var versionableContentItems = search.luceneSearch(store, query);

for each (versionableContentItem in versionableContentItems)
{
    // Do something with a versionable content
}
```

This search looks for all nodes that have the "versionable" aspect applied. The way you create the Lucene queries is the same as we discussed in the *Using the Search Service* section under Java Foundation Services.

To find out what methods are available to call on a so-called script node (that is, folder or file node) you can have a look at the `org.alfresco.repo.jscript.ScriptNode` class.

For more information about the Java Script API, see http://wiki.alfresco.com/wiki/3.3_JavaScript_API.

JavaScript event handlers

When we discussed the event API, which is part of the Foundation Service Java API, we did not mention that we can write event handlers in JavaScript. Here is an example of how to associate the `onUpdateNode` event with JavaScript code:

```
<bean id="eventHandler.onUpdateContent" class =
  "org.alfresco.repo.policy.registration.ClassPolicyRegistration"
  parent="policyRegistration">

  <property name="policyName">
    <value>{http://www.alfresco.org}onUpdateNode</value>
  </property>

  <property name="className">
    <value> {http://www.alfresco.org/model/content/1.0}content
    </value>
  </property>

  <property name="behaviour">
    <bean class="org.alfresco.repo.jscript.ScriptBehaviour"
      parent="scriptBehaviour">
      <property name="location">
        <bean class =
          "org.alfresco.repo.jscript.ClasspathScriptLocation">
          <constructor-arg>
            <value>alfresco/extension/scripts/someOnUpdateNode.js
            </value>
          </constructor-arg>
        </bean>
      </property>
    </bean>
  </property>
</bean>
```

This bean definition registers the `someOnUpdateNode.js` JavaScript to be called each time the `onUpdateNode` event happens for any `cm:content` type.

Debug logging

It is very useful to have debug logging turned on when developing JavaScript. This can be done via the `/alfresco/tomcat/webapps/alfresco/WEB-INF/classes/log4j.properties` file:

```
log4j.logger.org.alfresco.repo.jscript=debug
log4j.logger.org.alfresco.repo.jscript.ScriptLogger=debug
```

JavaScript or Java?

The more you can stick to JavaScript, the more flexible and easier it will be to maintain and update the code and the faster you will be able to develop your solutions.

Customers will also have a much easier way of maintaining your solution as they do not have to rebuild the module and redeploy just to make a little update. JavaScript can even be updated without restarting the server.

However, there are situations where you might be developing a product extension to Alfresco and it is delivered as an AMP and you have specific requirements that require new Java libraries to be used. Then it makes sense to use Java for this customization.

Client-server APIs

These are **Representational State Transfer (REST)** based application programming interfaces where the client is remote in a different process context than the Alfresco server. They are implemented as Web Scripts, which provide RESTful access to content held within the Alfresco Repository. Alfresco is delivered with a couple of these APIs out of the box, for example, the CMIS API and the Repository API.

CMIS API

The **Content Management Interoperability Services (CMIS)** API was created to provide users of CMS systems a common and standard way of accessing and searching for content. It should not matter if your client is accessing an Alfresco system or another vendor's CMS system, the client code should still be the same and not have to change if you decide to switch to another CMS system.

We can think of it in the same way we think about SQL for databases. An SQL query can be used to access data in any database.

To use the CMIS API construct an URL and make a HTTP call to the Alfresco server. For example, if we wanted to access all child folders of the /Company Home/User Homes folder we could use an URL as follows in any browser: `http://localhost:8080/alfresco/service/cmisp/User%20Homes/children`.

This would return an ATOM feed with the home folders after we have logged in with admin credentials. The **Apache Abdera** library may be used to parse the ATOM-formatted CMIS responses, but we could also use a basic XML parser to do the same. Unfortunately, the CMIS web scripts do not have the possibility to return JSON or HTML. However, some browsers such as Mozilla Firefox will show you the ATOM Feed as HTML for testing.

To see for example, what URL templates are available for a CMIS web script such as children, you can call another URL as follows: `http://localhost:8080/alfresco/service/script/org/alfresco/cmisp/children.get`.

CMIS also provides SQL-like query languages so you can do queries like this:

```
SELECT * FROM Document WHERE IN_FOLDER(
'workspace://SpacesStore/04761070-5e25-11dd-8ab6-b324934c9dae')
```

This query would return all documents in the folder specified by the Node Reference.

Repository API

In addition to the CMIS standard, Alfresco has exposed the Repository API for services such as workflow, activity feed, blogs, tagging, thumb nailing, user management, and more that are not available when using the CMIS API.

Custom APIs

With Web Scripts, we can build our own RESTful interface using lightweight scripting technologies such as JavaScript and FreeMarker. A **Web Script** is simply a service bound to a URI that responds to HTTP methods such as GET, POST, PUT, and DELETE.

The Alfresco Web Script Framework is implemented in a standalone fashion, so it can be run either embedded in the Alfresco server or in an external web server.

When talking about Web Scripts, we divide them into two categories:

- **Data Web Scripts:** They encapsulate access and modification of file content and file metadata stored in the repository. Therefore, these are available only via the Alfresco server.
- **Presentation Web Scripts:** They allow us to build user interfaces such as a Dashlet for Alfresco Explorer or Alfresco Share, a portlet for a JSR-168 portal, an UI component within Alfresco SURF, a website, or a custom application. They typically render HTML and unlike Data Web Scripts they may be hosted in the Alfresco server or in a separate web server. When hosted separately, they usually interact with Data Web Scripts.

To get an idea about how this works, let's implement a Web Script that runs the Foundation Service test that we have gone through in this chapter. The first thing we need to do is to define the script in the Spring configuration file as follows:

```
<bean id="webscript.com.bestmoney.test.foundationservice.testfs.get"
      class="com.bestmoney.test.foundationservice.
        TestFoundationServicesWebScript" parent="webscript">
  <property name="myCmsService">
    <ref bean="myCmsService"/>
  </property>
</bean>
```

This defines a new Web Script that the Alfresco server will pick up based on the naming convention specified in the `id` attribute:

- `webscript` tells the system that this is a Web Script definition.
- `com.bestmoney.test.foundationservice` is the package name where the Web Script descriptor file will be found, this package must be located under `alfresco/extension/templates/webscripts` in the classpath.
- `testfs` is the name of the Web Script.
- `get` is the HTTP method to be used when calling this Web Script.

The `class` attribute refers to the class that implements the code that should be executed when the Web Script is called. And we pass in the Foundation Service bean (that is, `myCmsService`) that is used to test different Foundation Services calls.

The `TestFoundationServicesWebScript` class looks like this:

```
public class TestFoundationServicesWebScript extends
    AbstractWebScript
{
    private MyCmsService m_myCmsService;
```

```

    public void setMyCmsService(MyCmsService myCmsService)
    {
        m_myCmsService = myCmsService;
    }

    public void execute(WebScriptRequest req, WebScriptResponse res)
        throws IOException {
        m_myCmsService.writeSomething();
        m_myCmsService.readSomething();

        res.getWriter().write("Done with the Foundation Service
            tests!");
    }
}

```

It extends the `AbstractWebScript` class to get some methods implemented for free and then we just have to implement the `execute` method to call the `myCmsService` bean.

Last thing we need to do is create a descriptor file for the Web Script and put this file in the `alfresco/extension/templates/webscripts/com/bestmoney/test/foundation-service` directory. It should be called `testfs.get.desc.xml` and looks like this:

```

<webscript>
  <shortname>Test Foundation Services</shortname>
  <description>This Web Script is used to test Foundation
    Services</description>
  <url>/3340_04/testfs</url>
  <authentication>user</authentication>
</webscript>

```

The descriptor defines the URL template, which in this case is `/3340_04/testfs` and means that we can call this Web Script with the following URL:
http://localhost:8080/alfresco/service/3340_04/testfs.

This will execute the different calls to the Foundation Services inside a transaction after we have logged in. We have specified that the user has authentication value, which means that a user has to log in before this Web Script will be run. This also means that the Web Script will be run implicitly inside a transaction.

Bootstrap APIs

There are several ways to bootstrap the system with custom functionality or new content.

Patches

We can create our own patches and plug them into the system by creating a class that implements the `org.alfresco.repo.admin.patch.Patch` interface. The easiest way to do this is to extend the `org.alfresco.repo.admin.patch.AbstractPatch` class that implements this interface and comes with member variables and setters and getters for common services:

```
public class MyPatch extends AbstractPatch
{
    private ImporterBootstrap importerBootstrap;

    public void setImporterBootstrap(
        ImporterBootstrap importerBootstrap)
    {
        this.importerBootstrap = importerBootstrap;
    }

    @Override
    protected String applyInternal() throws Exception
    {
        StoreRef storeRef = importerBootstrap.getStoreRef();
        NodeRef rootNodeRef = nodeService.getRootNode(storeRef);

        // Do some patching of the system...

        return "Success";
    }
}
```

This patch uses the `org.alfresco.repo.importer.ImporterBootstrap` object to get to the store reference and then to the store root node via the Node Service.

The patch is registered with the system as a Spring bean:

```
<bean id="patch.my" class="com.bestmoney.bootstrap.MyPatch"
      parent="basePatch">
    <property name="id">
        <value>patch.my</value>
    </property>
```

```

    <property name="description">
      <value>Test patch</value>
    </property>
    <property name="fixesFromSchema">
      <value>0</value>
    </property>
    <property name="fixesToSchema">
      <value>${version.schema}</value>
    </property>
    <property name="targetSchema">
      <value>10000</value>
    </property>
    <property name="importerBootstrap">
      <ref bean="spacesBootstrap"/>
    </property>
    <property name="nodeService">
      <ref bean="nodeService"/>
    </property>
  </bean>

```

If we have several patches to execute and they should be in a specific order, we can control that with the `targetSchema` value. The `fixesToSchema` value is set to Alfresco's current schema version (that is, via the `version.schema` variable), which means that this patch will always be run no matter what version of Alfresco is being used.

Importers

When using an importer, it is defined in Spring via a bean that is implemented with the `ImporterModuleComponent` class, which is a generic module component that can be configured to import data into the system. The bean extends the `module.baseComponent` and the definition looks like this:

```

<bean id="importer.presentationTemplate"
  class="org.alfresco.repo.module.ImporterModuleComponent"
  parent="module.baseComponent">
  <property name="moduleId" value="com_bestmoney_module"/>
  <property name="name"
    value = "com_bestmoney_module.bootstrapSpaces"/>
  <property name="description" value = "Initial data requirements"/>
  <property name="sinceVersion" value="1.0"/>
  <property name="appliesFromVersion" value="1.0"/>
  <property name="importer" ref="spacesBootstrap"/>
  <property name="bootstrapViews">
    <list>

```

```
<props>
  <prop key="path">
    /${spaces.company_home.childname}/${spaces.dictionary.childname}/
    ${spaces.templates.content.childname}
  </prop>
  <prop key="location"> alfresco/module/com_bestmoney_module/
bootstrap/presentation_template.xml</prop>
</props>
</list>
</property>
</bean>
```

When defining an importer bean, we specify the `path` in the repository where we want to store the imported content. And we also specify the `location` of the content that should be imported. In this case, the presentation template is defined in an XML file that could for example, be generated by exporting content via the **export** command-line tool.

Summary

This chapter has taken us through most of the available application programming interfaces for the Alfresco platform. We started off by going through how to set up an embedded custom service, including transaction management, and we then used the different Foundation Services, such as the Node Service to do low-level manipulation of content as an embedded customization.

We then created a Metadata Extractor for an XML document and saw how node properties can be set automatically when an XML document is added to the repository.

Alfresco by default does not come with a transformer for MS Visio documents to PDF documents, so we had a look at how this can be set up by configuring a custom Content Transformer.

In the first chapter, we learned that rules can be set up on folders to execute certain business logic when documents are added, deleted, or updated to a specific folder. In this chapter, we also went through how to implement repository wide rules with the so-called event handlers.

One of the nice things with Alfresco is that it also provides a JavaScript API that can be used to manipulate content in the repository. Using JavaScript is the preferred way of manipulating content, as it is quicker to write business logic with JavaScript than with Java, and the server does not have to be stopped and restarted while we are updating our customizations.

Finally, we had a look at how third-party applications can talk to Alfresco remotely via Web Scripts. Web Scripts are basically just HTTP requests that return JSON or XML. If you want a no-lock in solution, then have a look at CMIS, which is an effort to provide a standard Web Scripts API between ECM vendors.

In the next chapter, we will look in more detail at how an Alfresco extension project is set up and how to construct a build process that can be used to build extensions for both the Alfresco Explorer web application and the Alfresco Share web application.

3

Setting Up a Development Environment and a Release Process

When doing any kind of software development it is a good idea to plan and set up a development environment at the beginning of a project. So everybody involved knows how we are going to build the system, debug it, how it is organized, how continuous integration and testing will be done, and what deployment method will be used. If we can create a company-wide standard for how this is done, even better.

When setting up a development environment for building extensions for a CMS system such as Alfresco, we will have to set it up for a lot of CMS specifics, so it is good to have this documented.

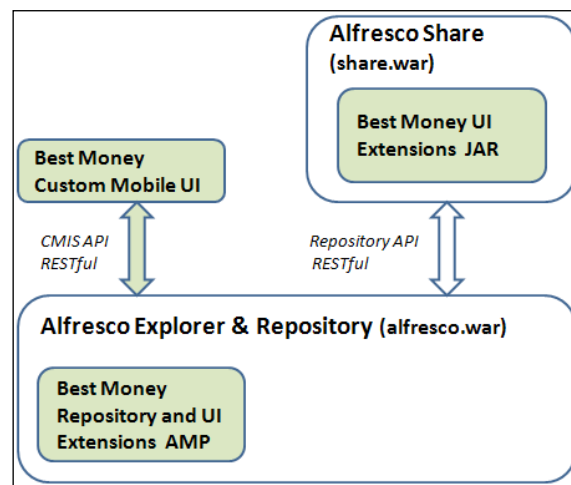
Here are some more reasons why we might want to create a standard development environment:

- To find files based on a standard organization
- To set up a naming convention for certain files, making it easy to spot what they are used for
- To organize similar files into the same directory
- To organize files according to how they are deployed
- To determine and set up the build file structure
- To decide how unit testing should be done
- To decide how continuous integration should be done to improve regression testing
- To scale the project both with new people and features

For the Best Money Content Management project, we will be building three different kinds of extensions for Alfresco:

- **Repository extensions:** Used to add custom content model, rules, workflows, web scripts
- **User Interface customizations:** Custom property sheets/forms, wizards, dashlets, pages
- **Standalone clients:** Mobile client, mashup client

The following figure shows the different extensions and how they relate to the different Alfresco web applications:



As we can see in the figure, when working with Alfresco, there are two web applications that we can customize and deploy extensions for. The main Alfresco web application that includes the Repository and the Alfresco Explorer user interface (that is, `alfresco.war`) and the Alfresco Share user interface (`share.war`).

Any custom clients that we build will most likely communicate with Alfresco via a REST-based API. Alfresco Share also communicates with the Alfresco repository via a REST-based API.

Every time we release a new version of the CMS extension to our client Best Money, we need to have some kind of process in place, so that there is no confusion about what was released and what tests have been done. To handle this, we will look at how a release process can be set up.

In this chapter, you will learn:

- How to set up a project directory structure to manage extensions to Alfresco Repository and Alfresco UI clients
- How to set up the build file that will create the extension modules
- How to set up a **continuous integration (CI)** solution
- How to set up a release process

Setting up a development environment

The Best Money content management extensions for Alfresco cover a wide area of features and functionality, so we need to set up the project directory structure and build file accordingly. We will build in-process repository extensions that are delivered via an AMP file, UI extensions delivered via a JAR file, and standalone remote clients.

Alfresco Extension projects

We will build two different kinds of extensions – one for the Alfresco Explorer (`alfresco.war`) web application and one for the Alfresco Share (`share.war`) web application.

Alfresco Explorer and repository extensions

The `alfresco.war` application is usually extended with a so-called **Alfresco Module Package (AMP)** file. An AMP file is basically a piece of a web application archive (WAR). It is merged with the `alfresco.war` file to enable custom content management extensions. Alfresco provides the Alfresco Module Management Tool (that is, `alfresco-mmt.jar`) to help with the merging and it can be found in the `alfresco/bin` directory in a standard installation.

There are several reasons why we need to merge our extensions into the `alfresco.war`:

- So that resources such as third-party Java libraries are picked up from `WEB-INF/lib`
- So that any custom Spring configuration is picked up correctly (it contains beans to load custom content models, message property files, patches, service implementations, and so on)

- So that JSP pages, CSS, scripts, and images can be loaded from `webapps/alfresco`
- So that the configuration of customizations for the Alfresco Explorer web client are picked up

Sometimes we also want to override Alfresco Explorer pages to insert or remove logic or widgets and this is also handled by the **Module Management Tool (MMT)**.

The MMT does not only manage the merging of the AMP into the WAR file, but it also manages:

- **Module versioning:** We can specify from what version a particular AMP file is applicable, and if somebody tries to install the AMP file in an older version the MMT will not allow it.
- **Module updates:** If an older module is installed, then the MMT will first remove all files related to the old module before installing the updated module.
- **Module listing:** The MMT can be used to list installed modules.
- **Backup of changes:** When we use the MMT to install a new AMP it will first back up any overwritten files and back up the current `alfresco.war` file.

The MMT tool expects the AMP file to be structured in a certain way. The following directory structure is the default and expected one:

```
| - /config
| - /lib
| - /web
|   | - /jsp
|   | - /css
|   | - /images
|   | - /scripts
| - module.properties
```

These AMP directories are mapped into the WAR as follows:

- `config: /WEB-INF/classes`
- `lib: /WEB-INF/lib`
- `web/jsp: /jsp`
- `web/css: /css`
- `web/images: /images`
- `web/scripts: /scripts`

The `module.properties` file is required to exist in the AMP file. It contains metadata about the module such as the ID and version.

If we want to customize the way the AMP file is mapped into the WAR file with the MMT application, then we can use the `file-mapping.properties` file to specify mapping exceptions.

This file should be put in the root directory together with the `module.properties` file. Each property key in the file points to a directory in the AMP file and each property value points to a directory in the WAR file.

For example, if we wanted to call the web directory, `webapp`, we could set up the property file as follows:

```
/webapp=/
```

The standard set of mappings will be included by default unless we set the property `include.default=false` in the `file-mapping.properties` file.

Alfresco Share UI extensions

The Alfresco Share web application contained in the `share.war` file does not need to be extended in the same way as the `alfresco.war` via an AMP. The **Spring Surf** web framework is used to build UI extensions for the Alfresco Share application and from Alfresco 3.3 and onwards, these extensions can be packaged in a JAR file and just dropped into the `webapps/share/WEB-INF/lib` directory.

The Spring Surf web framework knows how to locate resources such as configuration files and web resources from inside the JAR file.

The following directory structure is the default and the expected one for an Alfresco Spring Surf JAR file:

```
| - /alfresco
| - /site-data
|   | - /chrome
|   | - /components
|   | - /component-types
|   | - /configurations
|   | - /content-associations
|   | - /pages
|   | - /page-associations
|   | - /page-types
|   | - /template-instances
|   | - /template-types
|   | - /themes
```

```
| - /site-webscripts
| - /templates
| - META-INF
```

alfresco/site-data

The `site-data` directory contains the model objects that the Spring Surf web framework manages and refers to in order to render the web application properly. These model objects are instantiated in memory as standard Java POJOs and when written to the disk they are serialized as XML.

The `site-data` directory contains a subdirectory for each type of model object that is managed by the Spring Surf web framework.

alfresco/site-webscripts

The `site-webscripts` directory will contain all the Web Scripts that are used to render the web application components. There can be both, data Web Scripts and presentation Web Scripts, but for a Spring Surf application it will be mostly presentation Web Scripts generating the user interface via Freemarker templates.

These **Freemarker templates** are stored in the `alfresco/templates` directory.

META-INF

The `web resources/assets` required by the web application is saved under the `META-INF` directory. This directory is now basically the root for all URLs that are processed by the Resource Servlet.

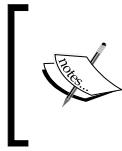
For example, to access an image called `logo.png` under the `META-INF/global/images` directory, we can use the following HTML:

```

```

The important thing to remember in this URL is the `/res` part, which is mapped to an URL Rewriter Filter (`org.tuckey.web.filters.urlrewrite.UrlRewriteFilter`). This URL Rewriter will change the `/res` part to `/page/resource/` and the request is then picked up by the Spring Dispatcher Servlet (that is, the Resource Servlet).

Alfresco has added an extra Spring configuration via a `UrlConfigSource` bean to be able to pick up resources in the `META-INF` folder in JARs (see the `tomcat/webapps/share/WEB-INF/classes/alfresco/slingshot-application-context.xml` file. Note, **Slingshot** was the first code name for the Alfresco Share web client, so you will see it a lot in the code).



If the Alfresco Share extension is more complex and involves Java classes and third-party Java libraries such as Spring configuration, then we are better off using an AMP to extend the Alfresco Share web application.

Project directory structure

The following directory structure will be used for Best Money's extension projects and it is general enough to be used for most Alfresco extensions – only some directory names will have to be changed to adjust it to your implementation:

```
| - bestmoney
  | - alf_extensions
    | - trunk
      | - /_alfresco
        | - /config
        | - /lib
        | - /source
        | - /test
        | - /web
      | - /_share
        | - /config
        | - /data
        | - /diagrams
        | - /schemas
        | - /sql
    | - alf_clients
      | - mobile
        | - trunk
          | - /lib
          | - /source
          | - /test
          | - /web
      | - mashup
        | - trunk
```

The `alf_extensions` directory contains all of the files for both Alfresco Explorer and Alfresco Share extensions. The `_alfresco` directory contains the files for the Alfresco Explorer (`alfresco.war`) extension and the subdirectories mean the following:

- `config`: Contains all configuration files including custom content model definitions, workflow definitions, messages, and web scripts.
- `lib`: Any libraries that are not available in the Alfresco SDK. These are libraries that we would need for more complex extensions to the Alfresco repository, such as supporting a new protocol.
- `source`: The Java and JavaScript source code, and any other source code.
- `test`: Java source code for unit tests and libraries related to unit testing.
- `web`: Web application resources such as JSPs and CSS.

The `_share` directory contains the files for the Alfresco Share (`share.war`) extension and the subdirectories mean the following:

- `config`: Contains all configuration files including web application components, pages, resources, web scripts.

The `alf_client` directory contains the files for any new web clients that we will be building.

The other directories contain the following type of files:

- `data`: Contains exported **Alfresco Content Packages (ACP)**. When we set up more complex folder structures with rules and permission settings, it is a good idea to export them into ACP packages and store them in the version control system. The same is true for any space templates that we create. These packages are also useful to include in releases.
- `diagrams`: Contain UML and workflow diagrams.
- `schemas`: For example, JPBm schemas that are referred to from the editor.
- `sql`: SQL scripts for things such as reporting and status updates.

`_alfresco/config`

The `_alfresco/config` directory will contain all of the configuration files for the Best Money's repository extensions and any UI customizations for the Alfresco Explorer client. This can be definitions of advanced workflows, UI customizations, custom content models, audit configuration, i18n messages, web script definitions, and more.

It contains the following subfolders and files:

```

|- config
  |- /alfresco
    |- /extension
      |- /templates
        |- /webscripts
          |- /com
            |- /bestmoney
              |- /cms
    |- /module
      |- /com_bestmoney_module_cms
        |- /bootstrap
        |- /context
          |- /bootstrap-context.xml
          |- /services-context.xml
        |- /messages
          |- /patch-messages.properties
        |- /model
          |- /content-model.xml
          |- /workflow-model.xml
        |- /templates
        |- /ui
          |- /web-client-config-custom.xml
          |- /webclient.properties
        |- /workflows
        |- /log4j.properties
        |- /module-context.xml
        |- /module.properties
  |- /META-INF
    |- /faces-config.xml

```

When the MMT tool is executed on the AMP, the `config/alfresco` directory is merged into the `/WEB-INF/classes` directory of the Alfresco WAR file.

alfresco/extension

The extension directory is one of the places where Alfresco will always look for custom configuration files, workflows definitions, and web script definitions. At first, it might be tempting to put everything in this directory, but that is not a good idea as other extension projects might have the same idea, which can cause problems like:

- **Name clashes:** Let's say, we do some custom configuration in the `webclient.properties` and `web-client-config-custom.xml` files. If another extension does the same thing, then the last to deploy with the MMT tool will be overwriting previous versions of these files.

- **Confusion:** It might be difficult to figure out what file is related to what module and extension.

Instead, put as much of the custom configuration as possible in the specific `module` directory, so that all of the files are kept under the module namespace.

There are some exceptions to this recommendation and that has to do with Web Scripts and how they are loaded. For them to be loaded properly, they have to be put in one of the few places. They can live in the repository as content or in the AMP file. The following list is specified in the sequence in which Alfresco searches for web script implementation files:

- Repository folder: `/Company Home/Data Dictionary/Web Scripts Extensions`
- Repository folder: `/Company Home/Data Dictionary/Web Scripts`
- Classpath directory: `/alfresco/extension/templates/webscripts`
- Classpath directory: `/alfresco/templates/webscripts`

Within any of these folders, we can use subfolders to organize Web Scripts and that is why we have added the `com/bestmoney/cms` subdirectory, so that we can put the Web Scripts related to the Best Money's CMS project into their own namespace.



Another case when we have to use the `alfresco/extension` directory is if we wanted to extend the number of available MIME types in Alfresco. We would then have to add the `mimetypes-extension-context.xml` file to the `alfresco/extension` directory and update it with a pointer to the file with the custom MIME type definitions, such as `bestmoney-mimetypes-extension.xml`.

alfresco/module

The module name has been set to `com_bestmoney_module_cms` to make it unique, so that all directories and files under this module will be unique among module deployments. This module will contain all Alfresco Explorer-related content management extensions for all Best Money departments.

If we have several teams developing extensions for Alfresco, it might be a good idea to come up with a standard naming convention for modules. For example:



- * `com_bestmoney_module_dm`: Document Management extensions
- * `com_bestmoney_module_workflows`: Workflow extensions
- * `com_bestmoney_module_rm`: Records Management extensions
- * `com_bestmoney_module_wcm`: Web Content Management extensions

Or you could also divide extensions by department:

- * `com_bestmoney_module_marketing`
- * `com_bestmoney_module_engineering`
- * `com_bestmoney_module_it`

Also be sure to use underscores (`_`) in the module directory name instead of dots (`.`). Otherwise, it will not be possible to load, for example, message property files.

The individual directories and files under the module are used for the following:

- `bootstrap`: Contains bootstrap view XML files that describe import definitions for content such as groups, users, spaces, e-mail templates, and presentation templates.
- `context`: Contains all Spring configuration files and any custom property files used by the extension module. The `services-context.xml` file is used to define beans for any new module services that we build—for example, event handlers. The `bootstrap-context.xml` file is used for patch definitions, importers, message loaders, loading custom web client configuration, and loading custom content model. These two Spring configuration files are loaded by the `module-context.xml` file.
- `message`: Should contain all property files for application messages. Patches are used quite often so we have created a file called `patch-messages.properties` to standardize the name of this file. And new content model properties are stored in the `model-messages.properties`. These files are loaded by a Spring configuration in the `bootstrap-context.xml` file.
- `model`: Contains the custom content models that we create. They can be divided into custom content models that are used to classify documents in the repository, contained in the `content-model.xml` file, and custom content models used by workflows, contained in the `workflow-model.xml` file. These files are loaded by a Spring configuration in the `bootstrap-context.xml` file.

- `templates`: Contains Freemarker templates such as presentation and e-mail templates. They are loaded by import definitions specified in XML files in the `bootstrap` directory.
- `ui`: Contains the standard Alfresco Explorer custom configuration file, `web-client-config-custom.xml`, and the associated `webclient.properties` file. These files are loaded by a Spring configuration in the `bootstrap-context.xml` file. An example of the XML file can be found in the `/tomcat/shared/classes/alfresco/extension` directory.
- `Workflows`: Contains JBPM workflow definitions for all workflows associated with this extension. Each workflow definition will be kept in a separate subdirectory.
- `log4j.properties`: Logging configuration for this extension module will contain only one line to set log levels for this module: `log4j.logger.com.bestmoney.cms=info`.
- `module-context.xml`: The main Spring configuration file for this module. It is picked up by Alfresco's Spring instance and it will in turn load the `services-context.xml` and `bootstrap-context.xml` files.
- `module.properties`: Contains the ID, description, and version number for this module.

META-INF

The `META-INF` directory contains the `faces-config.xml` JSF configuration file that is used when we want to add custom JSF managed beans. This directory will be part of the module JAR file that is contained in the module AMP.



If we wanted to overwrite existing Alfresco JSF managed beans or change the configuration for one of them, then we would have to use another configuration file called `faces-config-custom.xml`, which needs to be put in the `alfresco/tomcat/webapps/alfresco/WEB-INF` directory. The MMT tool has some problem updating this file automatically for us, so at the moment this has to be done manually.

`_alfresco/source`

The `_alfresco/source` directory will contain all the Java and JavaScript source code for the Best Money project. The directory structure under it looks like this:

```
| - source
    | - /javascript
    | - /java
```

```

|- /com
  |- /bestmoney
    |- /cms
      |- /bootstrap
      |- /eventhandler
      |- /model
      |- /service
      |- /util
      |- /web
        |- /ui
          |- /action
          |- /bean
          |- /script
        |- /workflow
        |- /actionhandler

```

The `javascript` directory will contain source code for business rules that are part of the document management setup. These JavaScript files are typically deployed into the `/Company Home/Data Dictionary/Scripts` folder and then used by the rules.

The `java` directory has the following subfolders:

- `bootstrap`: All patch implementations
- `eventhandler`: Any event handler implementation
- `model`: Contains a class that provides constants for content model QNames
- `service`: All custom services implemented for the extension are stored here
- `util`: Different utilities such as an interface with application constants
- `web/ui/action`: Classes related to UI actions such as an action evaluator
- `web/ui/bean`: Different JSF managed bean implementations for things such as wizards and workflow management
- `web/ui/script`: Web Scripts implemented in Java
- `workflow/actionhandler`: Contains Java action handlers for workflows

`_share/config`

The `_share/config` directory will contain all the configuration files for the Best Money's UI customizations for the Alfresco Share client. This can be definitions of pages, dashlets, Share client custom configuration, messages, Web Scripts, and more.

This directory contains the following subfolders:

```
| - config
  | - /alfresco
    | - /messages
    | - /extension-app.properties
  | - /site-data
    | - /chrome
    | - /components
    | - /component-types
    | - /configurations
    | - /content-associations
    | - /pages
    | - /page-associations
    | - /page-types
    | - /template-instances
    | - /template-types
    | - /themes
  | - /site-webscripts
    | - /com
    |   | - /bestmoney
    |   |   | - /cms
    |   |   |   | - /components
    |   |   |   | - /utils
  | - /templates
    | - /com
    |   | - /bestmoney
    |   |   | - /cms
  | - /web-extension
    | - /custom-slideshow-application-context.xml
    | - /fdk-config-custom.xml
    | - /share-config-custom.xml
    | - /web-framework-config-custom.xml
    | - /webscript-framework-config-custom.xml
  | - /META-INF
    | - /components
    | - /dashlets
    | - /images
```

This directory structure has had some extra directories added compared to the standard Alfresco Share JAR extension structure. The `messages` directory has been added and will contain all Web Scripts-related properties for this custom extension. The properties will be contained in the `extension-app.properties` file. This properties file is loaded via the `custom-slideshow-application-context.xml` Spring configuration file.

We have also added the `com/bestmoney/cms` subdirectory in some places. This is so we can put Web Scripts and templates in their unique Best Money namespace. This is important as there might be JAR extensions deployed from many different third parties in the future, and we do not want to have any name clashes between JAR extensions.

We also want to organize resources under `META-INF` into directories, so for example, Dashlet components will be saved in the `META-INF/components/dashlets` directory and associated Dashlet images in the `dashlets/images` directory.

The last thing we have added is the `web-extension` directory, which contains five configuration files that can be used to customize the following:

- `custom-slideshow-application-context.xml`: The custom Spring bean configuration file for Alfresco Share. Here you can override the Web Script search path and the template and script search path. This file also loads the `extension-app.properties` file.
- `fdk-config-custom.xml`: Form definitions for the new Alfresco **Forms Development Kit (FDK)** that can be used in Spring Surf applications such as Alfresco Share and Alfresco Mobile.
- `share-config-custom.xml`: Used for customizations of the Alfresco Share web client. We can customize for example, search term minimum and maximum length, username minimum length, and so on.
- `web-framework-config-custom.xml`: Can be used to enable or disable the cache for components and pages. We can also set up what theme we want the Alfresco Share web client to use.
- `webscript-framework-config-custom.xml`: We can use this file to override the endpoints for the remote Alfresco server when we want to run the Alfresco Repository and Alfresco Explorer (`alfresco.war`) on one machine and the Alfresco Share web application (`share.war`) on another machine. The structure and syntax of this file is similar to the `share-config-custom.xml` file.

Examples of these four configuration files can be found in the `tomcat/shared/classes/alfresco/web-extension` after a normal installation.

If you are familiar with the `web-client-config-custom.xml` file, which is used to customize the Alfresco Explorer web client, then it should be easy to use the `*-config-custom.xml` files, as they have a similar concept of how to configure things.

Building and deploying

Now we have a good idea of what the project directory structure looks like for both an AMP extension and a JAR extension. Let's see how we can create an Apache Ant script to build and deploy these extensions.

The Build file

The Apache Ant build file is constructed to be able to build both the AMP to extend the `alfresco.war` and the JAR to extend the `share.war`.

It is recommended that you download the files associated with this chapter from the *Packt website*, so that you can easily follow the walkthrough of the build file (`build.xml` + `build.properties`), as not all Ant targets are explained – only the ones that are related to building Alfresco extension artefacts.

Ant targets for the `alfresco.war` AMP Extension

The first Ant target that we will look at is the one that packages the files needed for the JAR file that goes into the AMP; it is called `package-alfresco-jar` and looks like this:

```
<target name="package-alfresco-jar"
  depends="clean, mkdirs, compile">
  <jar destfile="${alfresco.jar.file}">
    <zipfileset dir="${classes.alfresco.dir}" includes="**/*.class"/>
    <zipfileset dir="${source.alfresco.dir}"
      includes="**/*.xml,**/*.properties"/>
  </jar>
</target>
```

This Ant target is dependent on some standard targets such as `clean`, `mkdirs`, and `compile`. Nothing special with these targets and they are usually found in most build files and so are not explained here.

What the `package-alfresco-jar` target does then is uses the `jar` task to create the JAR file with the classes and any configuration files. The name of the JAR file comes from a property definition, `alfresco.jar.file`, which is specified in the `build.properties` file that usually accompanies the `build.xml` file. This JAR file will be contained in the AMP file that is generated by the next target called `package-alfresco-amp`:

```
<target name="package-alfresco-amp" depends="package-alfresco-jar">
  <zip destfile="${alfresco.amp.file}">
    <zipfileset dir="${project.dir}/build" includes="lib/*.jar"/>
    <zipfileset dir="${project.dir}" includes="lib/*.jar"/>
  </zip>
</target>
```

```

    <zipfileset dir="${alfresco.ext.dir}" includes="config/**/*.*"
      excludes="**/module.properties, **/file-mapping.properties"/>
    <zipfileset
      dir="${config.alfresco.dir}/alfresco/module/${module.id}"
      includes="module.properties, file-mapping.properties"/>
    <zipfileset dir="${alfresco.ext.dir}" includes="web/**/*.*/>
  </zip>
</target>

```

This Ant target builds an extension AMP with a name that comes from the `alfresco.amp.file` property that is specified in the `build.properties` file.

The AMP is built by including:

- Any library from the project's `build/lib` directory that would be the JAR file built with the first Ant target, `package-alfresco-jar`.
- Any library from the project's `_alfresco/lib` directory and this can be any extra JAR file not available in the Alfresco distribution but needed by the extension.
- All files under `_alfresco/config` except the `module.properties` and `file-mapping.properties` files.
- The `module.properties` and `file-mapping.properties` files and they will be at the top level in the AMP.
- All web resources stored under the project's `_alfresco/web` directory.

The last target we need for the AMP management is the one that deploys the AMP into the `alfresco.war` file. This target is called `deploy-alfresco-amp` and looks like this:

```

<target name="deploy-alfresco-amp"
  depends="clean-reset-war, package-alfresco-amp">
  <java dir="." fork="true"
    classname="org.alfresco.repo.module.tool.ModuleManagementTool">
    <classpath refid="src.class.path"/>
    <arg line="install ${alfresco.amp.file}
      ${alfresco.war.file} -force -verbose -nobackup"/>
  </java>
</target>

```

To deploy the AMP, we will use the MMT, which can be run from an Ant task. We configure a Java task to do this and we supply it with the AMP filename and the `alfresco.war` file location and name. The MMT is also instructed not to make a backup of the original `alfresco.war` (that is, `-nobackup`). We do not want a backup as the build script will automatically restore the original `alfresco.war` file from an `alfresco.war.bak` file that we have created before running this target. This is done via the `clean-reset-war` target that this deploy target is dependent on.



When working with AMP extensions, it is quite important to know that every time I do a new deployment, the `alfresco.war` will contain only the exact files that are in the AMP and not some old extension file(s) that has been removed in this new version of the AMP, for example, a library.

This is why the deploy target always restores the original `alfresco.war` before each new deployment. If we are working with several AMPs, then the original `alfresco.war` might not be a clean `alfresco.war`, as seen after installation of Alfresco, but one with some extra AMPs in it, except the AMP that we are currently working on.

Ant targets for the share.war JAR extension

The Ant targets for the Alfresco Share JAR extension are quite simple as the packaging is simple and the deployment is just about copying a file. Here is the first Ant target, `package-share-jar`, used to package the JAR:

```
<target name="package-share-jar"
  depends="clean, mkdirs, compile">
  <delete file="${share.jar.file}" />
  <jar destfile="${share.jar.file}">
    <zipfileset dir="${config.share.dir}" includes="**/*.*" />
  </jar>
</target>
```

What this target does is basically packages all files under the `_share/config` directory into a JAR file with a name specified with the `share-jar-file` property. The deployment target that copies this file into the `tomcat/webapps/share/WEB-INF/lib` directory looks like this:

```
<target name="deploy-share-jar" depends="package-share-jar">
  <copy file="${share.jar.file}"
    todir="${tomcat.webapps.dir}/share/WEB-INF/lib"/>
</target>
```

Using the build file to deploy extensions

When we have coded an `alfresco.war` extension and a `share.war` extension, we can build and deploy as follows. But first change the following properties in the `build.properties` file, so that they match your installation:

```
alfresco.dir=C:/Alfresco3.3Book
alfresco.sdk.dir=C:/tools/alfresco-community-sdk-3.3
jdk.dir=C:/Program Files/Java/jdk1.6.0_11
```

During the various builds and deployments, we will see a new temporary build directory structure created:

```
| - trunk
  | - /build
    | - /classes
    | - /dist
    | - /lib
```

This directory structure is safe to delete at any time as it is created during each build.

The `dist` directory will contain the built AMPs and the `lib` directory will contain any built JARs, if we wanted to get hold of them.

Deploying the AMP extension

Make sure that the Alfresco server is stopped by executing `alfresco/alf_stop.bat`.

Create a backup copy of `alfresco.war` called `alfresco.war.bak` and put it also in the `alfresco/tomcat/webapps` directory. This is done so that we do not have to manually restore the `alfresco.war` to its original state between builds. The build script copies `alfresco.war.bak` to `alfresco.war` before each build to prevent old files that might have been deleted hanging around.

Run the `deploy-alfresco-amp` Ant target to build and deploy the AMP:

```
C:\3340_03_Code\bestmoney\alf_extensions\trunk>ant -q deploy-alfresco-amp
[echo] Deleting webapps/alfresco dir and copying back original
alfresco.war from alfresco.war.bak
[echo] Packaging extension JAR for AMP
[echo] Packaging extension AMP file for alfresco.war
[echo] Installs extension AMP file into alfresco.war
```

BUILD SUCCESSFUL

Total time: 9 seconds

To test the new extension that was just installed, start the Alfresco server by executing `alfresco/alf_start.bat`. When the server starts, we should see a log indicating that the new extension module has been successfully installed:

```
11:02:50,756 INFO [org.alfresco.repo.module.ModuleServiceImpl] Starting
module 'com_bestmoney_module_cms' version 1.0.
```



The AMP deployment will remove the `webapps/alfresco` directory, as this is necessary for the AMP to be picked up by the Tomcat server. If the exploded WAR directory was not removed, then the AMP would not be loaded. The same problem exists if we would deploy any AMP from command line with the `alfresco/bin/alfresco-mmt.jar` tool.

So in this case, it is important that we have not done any configuration of Alfresco via any configuration file found in the exploded WAR file. Instead, all configuration should be done in the configuration files that can be found in the `tomcat/shared` directory.

Deploying the Share JAR extension

When working with Spring Surf extensions for Alfresco Share it is not necessary to stop and start the Alfresco server between each deployment. We can set up Apache Tomcat to watch the JAR file we are working with and tell it to reload the JAR every time it changes.

Update the `tomcat/conf/context.xml` configuration file to include the following line:

```
<WatchedResource>WEB-INF/lib/3340_03_Share_Code.jar</WatchedResource>
```

Now every time we update this Share extension, JAR Tomcat will reload it for us and this shortens the development cycle quite a bit. The Tomcat console should print something like this when this happens:

```
INFO: Reloading context [/share]
```

To deploy a new version of the JAR just run the `deploy-share-jar` ant target:

```
C:\3340_03_Code\bestmoney\alf_extensions\trunk>ant -q deploy-share-jar
[echo] Packaging extension JAR file for share.war
[echo] Copies extension JAR file to share.war WEB-INF lib
```

```
BUILD SUCCESSFUL
```

```
Total time: 0 seconds
```

Debugging extensions

Sooner or later we will come into a situation when we have a problem and it is no longer possible to figure out what is going on by just looking at the source code. Then we have to start debugging.

Alfresco Explorer and repository debugging

To debug AMP extensions, start the Alfresco server so that it listens for remote debugging connections; or more correctly, start the JVM so that it listens for remote debugging connection attempts. This can be done by adding the following line to the operating system as an environment variable:

```
CATALINA_OPTS=-Dcom.sun.management.jmxremote -Xdebug -
Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5005
```

This means that any Alfresco installation that we have installed locally on our development machine will be available for debugging as soon as we start it.

Change the address as you see fit according to your development environment.

With this setting we can now debug both into Alfresco's source code and our own source code at the same time.

Alfresco Share debugging

When we develop extensions for Alfresco Share, it is going to be more about UI development; so install, for example, Firefox and the Firebug extension. This will give you the possibility to debug JavaScript, inspect CSS, and so on.

To be able to debug client-side JavaScript with a default Alfresco Share installation, we first have to change some debug settings. Open up the `share-config-custom.xml` file in the `config/web-extension` folder and add the following configuration:

```
<alfresco-config>
...
  <config replace="true">
    <flags>
      <client-debug>true</client-debug>
      <client-debug-autologging>>false</client-debug-autologging>
    </flags>
  </config>
...
</alfresco-config>
```

The first flag, `client-debug`, sets up debug mode for client scripts in the browser. So why do we need to do this and should it not be enough to use Firebug? Alfresco has done some performance improvements to the JavaScript code and Alfresco Share will include minimized versions of its client-side JavaScript files during page rendering, and it will also combine some of the files. Setting the debug flag to true forces Alfresco Share to load the original uncompressed JavaScript source with all comments and formatting intact. It also ensures the various YUI library files are uncompressed.

Setting the `client-debug-autologging` flag to `true` automatically activates the logging output window in Alfresco Share, so we can use traditional Log4J statements, as follows, in our JavaScript code:

```
if (Alfresco.logger.isDebugEnabled()) {  
    Alfresco.logger.debug("Foo Bar");  
}
```

We can use the standard Log4J INFO, WARN, ERROR, and FATAL levels.

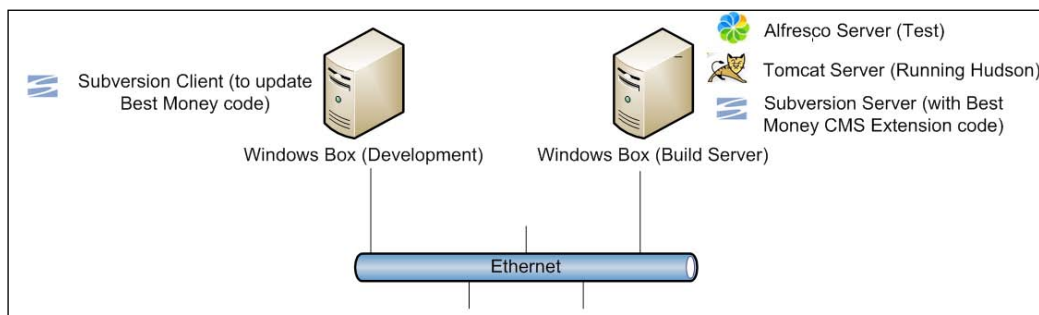
[ After changing the `share-config-custom.xml` file, a server restart is required.]

Setting up a continuous integration solution

In most development projects, it is a good idea to set up a **Continuous Integration (CI)** solution to be able to catch problems at an early stage. All extensions that we are building should be continuously built and regression testing should be carried out as soon as somebody checks in any changes to the project files.

One open source build server that we can use is **Hudson**. It can be configured to use project files in a Subversion repository and can be built with Apache Ant. In this section, we will have a look at how we can get started with continuous integration builds.

We will set up the following CI environment:



So the only thing we need in addition to our development environment is another Windows box. We will then set up Alfresco, Tomcat, and Subversion on this server.

Here is what we need to download to the build server:

- JDK
- Apache Ant
- Alfresco – the full installation including MySQL
- Apache Tomcat
- Hudson (download the WAR file – `hudson.zip`)
- CollabNet Subversion Server

After downloading these software packages, install them starting with the JDK, then Alfresco, followed by Apache Ant. During these installs, just follow the default options. Now copy the `alfresco/tomcat/webapps/alfresco.war` to `alfresco.war.bak`.

Then install Apache Tomcat and when this is done change the connector port to 8081 and the server port to 8006, so that it does not clash with Alfresco's Tomcat installation.

Last thing we need to do is install the Subversion server and during installation we can just follow the default options. The Subversion server needs at least one repository created, so let's do that:

```
c:\svn_repository\svnadmin create repo
```

For the Subversion client on the development machine to be able to access the build server, we need to open port 3690 in Windows firewall on the build server. We can test access to Subversion from the developer box by doing:

```
telnet <build server hostname> 3690.
```

Now, let's start all the servers as follows:

- Alfresco
- Tomcat
- Subversion server

Deploy Hudson in the Tomcat server by renaming `hudson.zip` to `hudson.war` and then copy it to `/Program Files/Apache Software Foundation/Tomcat 6.0/webapps`. To check if Hudson is up, access the `http://localhost:8081/hudson` URL.

We also need to import the code into Subversion and we can do this from the command line as follows:

```
C:\work>svn import -m "Initial Import" bestmoney http://<hostname>/svn/
repo
Adding          bestmoney\alf_extensions
Adding          bestmoney\alf_extensions\trunk
Adding          bestmoney\alf_extensions\trunk\sql
Adding          bestmoney\alf_extensions\trunk\_share
Adding          bestmoney\alf_extensions\trunk\_share\config
...
```

Okay, so we got the code in Subversion and Hudson is up and running. Now we just need to tell Hudson about the code and what Ant targets to build. We do this by configuring Apache Ant in Hudson and then setting up a **job**:

1. Access Hudson: `http://<hostname>:8081/hudson`.
2. Configure Apache Ant location by clicking on the **Manage Hudson** link and then **Configure System**: set the name of this Ant instance to `Apache Ant 1.8` and Home to `C:\apache-ant-1.8.1`.
3. Click on the **New Job** link.
4. Set name of Job to **Best Money Alfresco CMS Extensions**.
5. Select Job Type: **Build a free-style software project**.
6. Under **Source Code Management**, click on **Subversion**.
 - Set up Subversion URL: `http://localhost/svn/repo`.
7. Under **Build**, select **Invoke Ant**.
 - Pick **Apache Ant 1.8**.
 - Targets: `package-alfresco-amp,package-share-jar`.
 - Click on the **Advanced** button.
 - Build File: `alf_extensions/trunk/build.xml`.

To check if the build will work, click on the **Build Now** link. If things go well, we should see something like this:



So far, this does not use the Alfresco server. What we want is the build to stop the server and install the AMP and the JAR extensions. Then, start the server and see if that worked okay.

To do this, we need a couple of new Ant targets. We are going to need one Ant target that we can use to start the Alfresco server and the MySQL server. For this, we will use the `alf_start.bat` file that comes with the installation:

```
<target name="start-alfresco">
  <exec executable="cmd" dir="{${alfresco.dir}" spawn="true">
    <arg line="/c alf_start.bat"/>
  </exec>
  <waitfor maxwait="10" maxwaitunit="second" checkevery="5000">
    <http url="http://localhost:8080"/>
  </waitfor>
</target>
```

Once we have called the BAT file to start Alfresco and MySQL, we will wait in a loop for Tomcat to become available on `http://localhost:8080`. When that URL is accessible we know that the server is up and running.



If we are running on Linux, we would have to change the script we are using to the `alfresco.sh` start script and also update the `exec` task.

The next Ant target that we are going to need is the one that can be used to stop the Alfresco server. When we want to stop the server, it is first easy to just assume that we can use the same technique as when we started it and call `alf_stop.bat`. This will not work as that BAT file will not be able to completely stop Tomcat.

We could then resort to trying and controlling Tomcat deployments and undeployments with specific Tomcat Ant tasks. However, that will not work either as JARs used by the web application will still be locked by Tomcat, and they will not be removed during an undeployment. This prevents us from doing a deployment again. We could then try and set up so that resource locking is disabled in Tomcat (that is, `<Context antiJARLocking="true" antiResourceLocking="true">`). This will not work either as the web application will not start properly after this.

So, because of these reasons, we will use an Ant target that completely kills the Tomcat process. Any JVM process can be found by using the `jps.exe` tool that comes with the JDK and it returns a list of PID and Process Name:

```
<target name="kill-tomcat-server">
  <exec executable="${jdk.dir}/bin/jps" output="procidlist.out"/>
  <loadfile srcfile="procidlist.out" property="pid.out">
    <filterchain>
      <linecontains>
        <contains value="Bootstrap"/>
      </linecontains>
      <tokenfilter>
        <deletecharacters chars="Bootstrap"/>
        <trim/>
        <ignoreblank/>
      </tokenfilter>
      <striplinebreaks/>
    </filterchain>
  </loadfile>
  <exec spawn="true" executable="taskkill">
    <arg line="/PID ${pid.out}"/>
    <arg line="/F"/>
  </exec>
  <delete file="procidlist.out"/>
</target>
```

This target stores the output of `jps` in a file called `procidlist.out`, which is then loaded and filtered to find the process with the name `Bootstrap`. The process ID (PID) for this process is used when calling the `taskkill` command.



This way of killing the process will work only on Windows. On Linux, this Ant target has to be rewritten. On Linux, it might even work to just use `alfresco.sh stop`.

This just kills the Tomcat process and we also need a target to stop MySQL, as it is started by the `start-alfresco` Ant target. Here is how we can do that:

```
<target name="stop-mysql-server">
  <exec executable
    ="${alfresco.dir}/mysql/bin/mysqladmin.exe">
    <arg line="-u root shutdown"/>
  </exec>
</target>
```

We have now got enough functionality to be able to create a new Ant target that we can call from Hudson, which will run all JUnit tests, stop Alfresco, deploy new AMP, start Alfresco, and run Web Script tests:

```
<target name="stop-deploy-start-test"
  depends="unit-test, kill-tomcat-server,
    stop-mysql-server, deploy-alfresco-amp,
    start-alfresco, webscript-test">
</target>
```

To test Share extensions, just change the `deploy-alfresco-amp` target to `deploy-share-jar`. Now just change the following setting in Hudson to use the new target:

- Under **Build**, select **Invoke Ant**.
 - Pick **Apache Ant 1.8**.
 - Targets: **stop-deploy-start-test**.
 - Click on the **Advanced** button.
 - Build File: `alf_extensions/trunk/build.xml`.

To get these builds to kick off automatically whenever something is checked in to Subversion, we can use a plug-in that comes with Hudson. The plug-in is bundled with `hudson.war` and once it is installed, you'll see Subversion as one of the options in the SCM. This means that Hudson can now poll Subversion for changes. However, the best way is to let Subversion call Hudson whenever something is checked in. We can set up a `post-commit-hook` in Subversion that triggers Hudson to start the build.

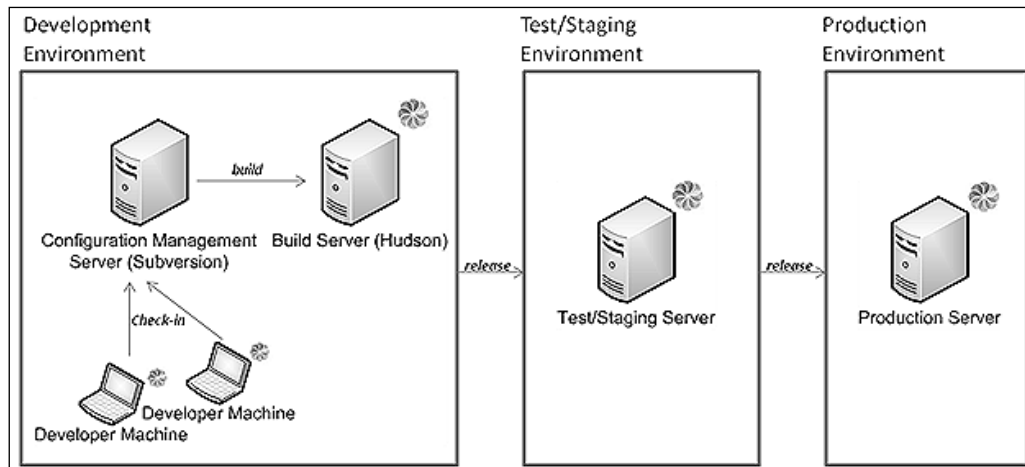
This should give us a good starting point for building a continuous integration solution for an Alfresco extension project.

Setting up a release process

When we start implementing a content management solution for a client, we will most likely start working in a development environment. The development environment is likely to be quite different from the production environment in terms of what hardware it runs on, what operating system it uses, how authentication and synchronization of users and groups are done, the number of users accessing it and what operating system they use, the amount of data processed and stored, and so on.

Because of this, a testing or staging environment is often put in place that mirrors the production environment as much as possible. So when we have implemented a new version and want to release it, this is done first to the test environment where a selected group of real users will take the new version for a spin. If everything is working out okay, the new version is later deployed to production.

We can call the process of packing up a new version, deploying it to test for user acceptance testing, and then finally deploying it to production for the **release process**. The following figure illustrates:



Here, we can see that the build server and the configuration management server have also been included in the development environment. This is because they are part of the development process.

To get the best result when releasing a new version of an Alfresco CMS extension, it is important to put in place as much testing as possible before the new version is deployed to production.

Testing is usually done in different stages – we have unit testing and smoke testing in the development environment and then user acceptance testing and load testing in the test environment.



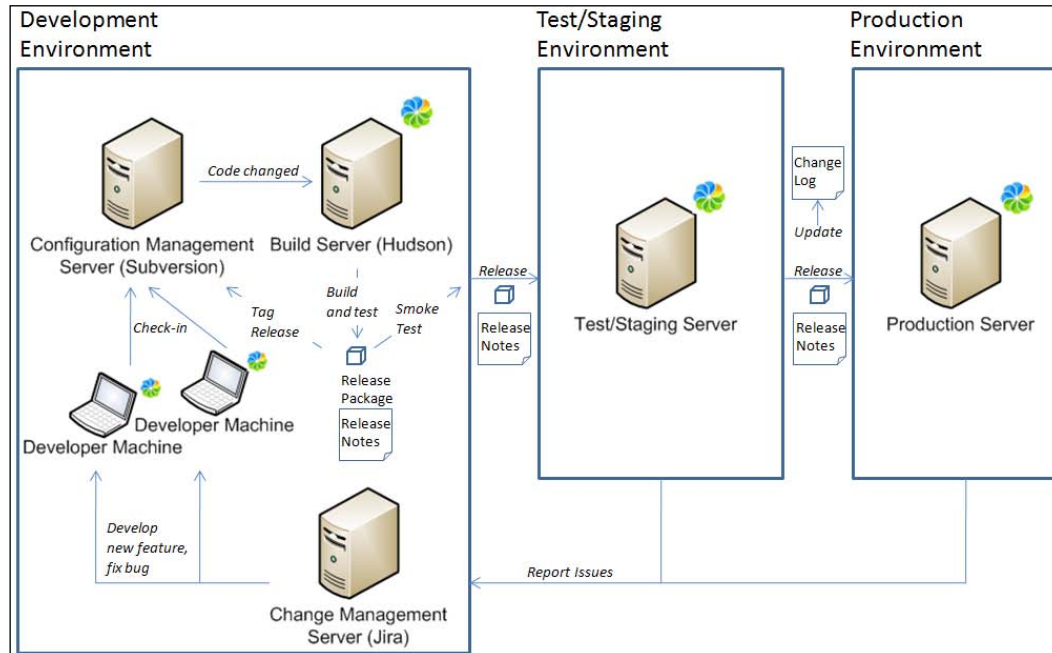
A **smoke test** in this context is when the developer runs thru a list of the most used use cases/requirements and verifies that they all pass. This is different from unit tests that just verify the internal logic of one specific component or service. Also, the unit tests do not require the Alfresco server to be started, which a smoke test does, as it should test the system through complete usage scenarios.

During the user acceptance testing phase, we need a way to organize and manage any issues that the users experience. They need a way of logging problems that they are experiencing. For this we can put in place a change management system and it can also be used to manage the requirements for upcoming versions.

When releasing a new version to production, a change log should be kept of what was updated. This change log will contain more things than just information about new module extension deployments. It will also contain information about major changes to folder hierarchies, folder hierarchy templates, configuration changes unrelated to the module extension deployment, and so on.

When releasing to either the test environment or the production environment, release notes should accompany the release package. This is so that everybody involved will know exactly what features have been fixed in the release and what new features are included, and what files might need to be manually updated.

The release process figure now looks a little bit more crowded:



In a lot of cases, the development environment will not consist of three servers (like in the preceding figure), but instead the configuration management system, build system, and change management system will run on the same server.

If we look at an iteration of a typical development and release process scenario, it might look something like this:

1. (Developer) implements some new features and fixes bugs.
2. (Developer) builds package locally and runs unit tests.
3. (Developer) checks in new features and fixed bugs.
4. (Build Server) is triggered by Subversion and runs automated tests and builds a new package.
5. (Developer) does a smoke test (that is, deploys the package onto local Alfresco and goes thru a list of the most used use cases and verifies that they do not break).
6. (Developer) writes a release note to follow with the release package.
7. (Developer) tags the new release in Subversion (so that it is always possible to get the source code for the package that runs in production, for emergency patching, if necessary).

8. (Developer) deploys package in test environment and gives release notes to system integrators and testers at the client side.
9. (Test Users) runs through a list of tests that have to pass for the release to be accepted. If a bug is found, it is reported via the Change Management system. If the acceptance test was successful, then we go on to the next step, otherwise we go back to Step 1 again.
10. (System Administrator) deploys new version to production and updates change log.
11. (End Users) reports any issues via the Change Management system.

The build server step in this setup could be left out in smaller development environments where there is just one developer implementing the whole solution. But the test and staging environment is needed for any kind of implementation. If we do not have a test environment, it is going to be too costly to discover all problems in the live environment.

Release notes template

The following template can be used to describe a release. The top section contains information about the release from the implementer and the bottom section contains information about the release from the client:

RELEASE NOTES	
Release version number:	major.minor.patch
Date released to <client name>	
Implementer (Consulting Company)	
Sent by	
Description of Content (business words):	
Issues fixed in this release	
Deploy AMP (Explorer & Repo)	[yes no]
Deploy JAR (Share)	[yes no]
Update space templates	[yes no]
Deploy workflows	[yes no]
Update faces-config-custom.xml	[yes no]
Included files:	

RELEASE NOTES

<client name>

Date received:

Received by:

Comments:

Due to release in Test:

Due to release in Production:

The following example shows how we can use this template when dealing with the Best Money client:

RELEASE NOTES

Release Version Number:

1.2

Date released to Best Money

2010-05-26

Implementer (Consulting Company)

Opsera Ltd

Sent by

Martin Bergljung

Description of Content (business words):

Updated Content Model and added a Meeting type and it is now possible to search for content based on the meeting type.

Updated the Activity property in the Workflow Model and removed the mandatory setting. This is to let old ongoing Job process instances use the new Workflow Model without having to assign a value to the Activity property.

Added extra Presentation folder to the Meeting Folder Template.

Issues Fixed in this Release

BM-2, BM-4, BM-12

Deploy AMP (Explorer & Repo)

Yes

Deploy JAR (Share)

No

Update Space Templates

Yes

Deploy Workflows

No

Update faces-config-custom.xml

No

Included files:

best_money_cms_1.2.0.amp

best_money_meeting_folder_template_1.2.0.acp

Best Money

Date received: 2010-05-27

Received by:

John Sysadmin

Comments:

Due to release in Test:

2010-05-30

Due to release in Production:

Updating Change Log

The change log that is updated every time there is a change in the production environment can have many formats. It should be kept on a wiki page for easy access. Here is one format that can be used:

Change Log for Best Money CMS		
Date	Description	Who
2010-02-15	Best Money AMP upgraded to 1.3.0. Fixed BM-28, BM-30, BM-38, BM-42, BM-46.	mbergljung
2010-01-27	Upgraded Alfresco from 3.1.1 to 3.2.2. Fixes problems with opening office docs from CIFS. Changed name of folder:	mbergljung
2010-01-15	/Company Home/Internal/Consulting to /Company Home/Internal/Customers & Prospects.	kanderson
2010-02-23	Updated space template Project with new Technical Doc folder.	ojonson

Training

In any content management system implementation, there is going to be a need to train users in both Alfresco and in the Alfresco extensions that we build. The test environment is usually used for this.

The best way to manage the training is probably to split it up into a number of steps:

1. **Step 1** teaches the users how to use a standard Alfresco installation with focus on Document Management. We should try and cover interaction via Alfresco Explorer, Alfresco Share, and CIFS.
2. **Step 2** (optional) teaches users how to use the Alfresco Share collaboration and sharing features.
3. **Step 3** (optional) teaches users how to use the **Records Management (RM)** module of Alfresco Share.
4. **Step 4** (optional) teaches users how to manage store and e-mails in Alfresco.
5. **Step 5** teaches the users about the content management extensions that have been implemented to support their business needs, business rules, and business processes.

Step 2 is optional and needs to be included only if we are going to use the Alfresco Share collaboration and sharing environment with Share sites. The same goes for Step 3, which is necessary only to include if the project involves records management with the Alfresco Share RM Module. Step 4 is also optional as not all projects are going to manage e-mails in Alfresco.

Summary

This chapter has walked through how to set up a development environment and a release process when implementing Alfresco content management extensions. In this chapter, you have learned how to set up a project directory structure so that a project can scale, fit deployment structure, and be easy for people to get up to speed on.

We created a build file that can build and deploy both Alfresco Explorer AMP extensions and Alfresco Share JAR extensions. The build file also had targets for automatically starting Alfresco, installing extensions, running tests, and then stopping Alfresco.

We saw how continuous integration can be integrated into the development environment for continuous regression testing as soon as something is checked into Subversion. When we have built our Alfresco extension, a release process can be used to manage project releases with a release notes template, production site change log, and so on.

In the next chapter, we will look at how to hook up Alfresco to directory servers for authentication and synchronization of user and group information. We will also see how **Single sign-on (SSO)** can be set up. This chapter also gives an example of how a custom authenticator can be implemented.

4

Authentication and Synchronization Solutions

Most companies will have some kind of directory service setup where they manage all users, groups, and the members of each group. So it is quite natural that when we are about to install Alfresco at a client site, they will sooner or later want us to hook it up to this directory service. Otherwise, they would be forced to manage users and groups in two different systems, which is not ideal to say the least.

There are also different authentication mechanisms that we might have to integrate with such as HTTP basic authentication, NTLM, or Kerberos. So we need to know how they work and how to set up Alfresco to authenticate with them.

In this chapter, we will look at:

- How different authentication mechanisms such as HTTP Basic, NTLM, and Kerberos work
- How directory servers such as OpenLDAP and MS Active Directory work, and what is the difference between them when we want to integrate with Alfresco
- Setting up an authentication solution that allows both users in MS Active Directory and users in an Apache Directory server to log in to Alfresco
- Enabling users in both OpenLDAP and Active Directory to connect to Alfresco via CIFS
- Configuring **single sign-on (SSO)**, so that users do not have to log in every time they want to access Alfresco
- Using Apache Directory Studio to inspect LDAP directory data
- Building our own authenticator

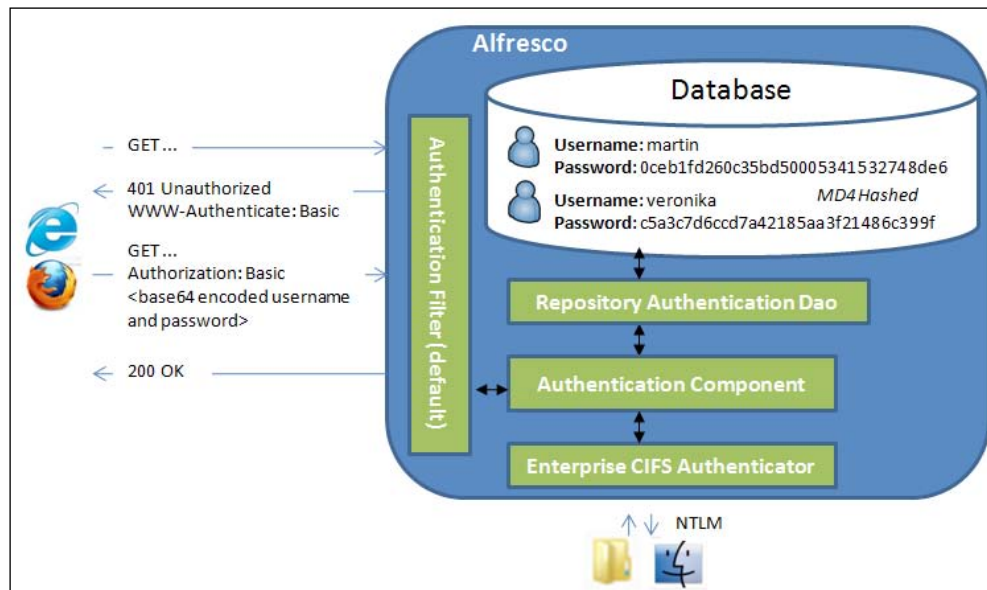
Authentication and synchronization concepts

Before we go into looking at configuring directory services, authentication, and synchronization, we will go through some concepts that might be useful to understand.

Basic authentication

After installing Alfresco, it will be configured to authenticate users via its internal database. This is handled via normal HTTP Basic authentication in the browser.

The following diagram illustrates this:



The advantages with this authentication method are that it is supported by most browsers and we can get going with our Alfresco installation immediately without any further configuration. However, if we are not using HTTPS then the passwords are basically sent in plain text over the wire. They are sent as base64 encoded, but that is easy to decode with lots of available libraries.



Note that if we access Alfresco after a default installation, we will be logged in as a guest and not prompted for username and password. In this case, we have to click on the login link to authenticate. Automatic guest login can be turned off as we will see later in this chapter.

An MD4 message digest of the password is stored in the database and CIFS authentication relies on this for authentication. So any authentication we set up that should work with CIFS has to store the MD4 hash of the password.

This authentication mechanism is stateless, so it will not remember that we have logged in when the browser is closed and the session ends.



CIFS authentication is always using NTLM or Kerberos. So if your Windows credentials do not correspond with a username and password in Alfresco's database then CIFS will not work. However, when mapping a network drive, you can specifically set what username and password to use so it matches an existing account, or just use admin/admin if it is a brand new installation and you want to try out CIFS.

NTLM authentication

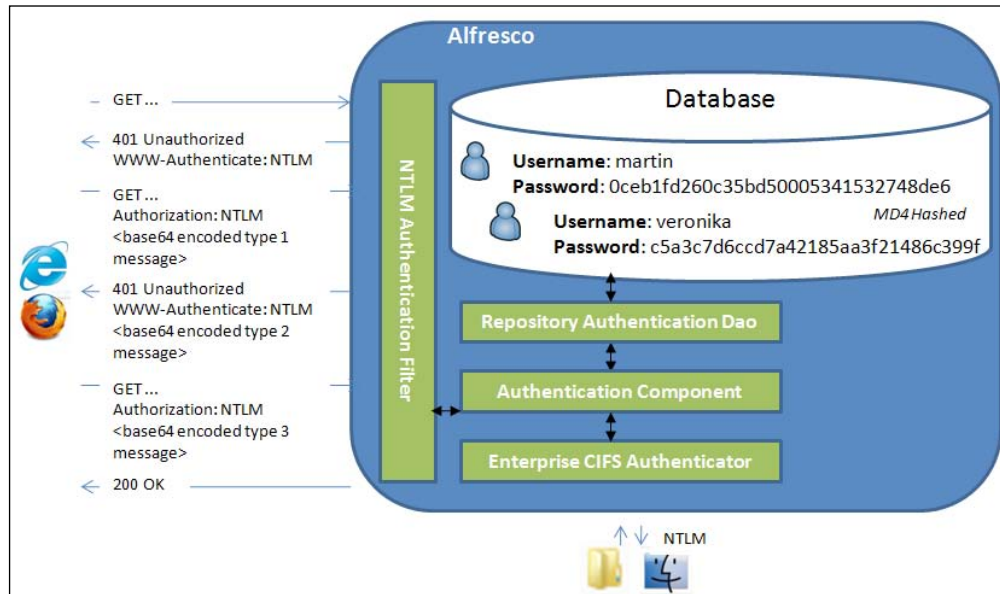
Alfresco can be set up to support NTLM authentication in order to achieve single sign-on (SSO) for the users, so they never have to log in to Alfresco. Instead, their Windows credentials will be used to log in automatically as soon as they go to the Alfresco URL in the browser.

NTLM, or NT LAN Manager, is a Microsoft authentication protocol and it is embedded in several Microsoft implementations of protocols such as HTTP, SMTP, POP3, and CIFS.

The protocol is a challenge-response protocol that requires the transmission of three messages between the client and the server:

- Client sends message 1 (type 1 – negotiation) with features it supports (such as encryption key sized).
- Server sends message 2 (type 2 – challenge) with similar content of what features it supports, and a random challenge generated by the server.
- Client uses the challenge obtained from message 2 and its credentials to calculate a response.
- Client sends the calculated response in message 3 (type 3 – authentication) to the server. MD4 hash is usually used to encode the response.

Alfresco implements NTLM via a Servlet Filter that handles the NTLM message communication for web client requests and a CIFS Authenticator to manage CIFS connections. The following diagram illustrates this:



NTLM is a lot more secure than HTTP Basic Auth as it never ships the password over the wire. There are different versions of NTLM depending on how secure the communication should be such as LM—DES, NTLMv1—MD4, and NTLMv2—MD5.

 The more secure NTLMv2 is supported and if the client does not support it then an automatic downgrade to NTLMv1 happens.

When NTLM is enabled, Internet Explorer will use your Windows login credentials when requested by Alfresco. Mozilla Firefox also supports the use of NTLM, but you need to add the URI to the Alfresco site that you want to access to `network.automatic-ntlm-auth.trusted-uris` option (to update this property enter `about:config` in the URL field) to allow the browser to use your current credentials for login purposes.



NTLM has mostly been replaced by Kerberos as authentication protocol for Microsoft domain-based scenarios. However, Kerberos is a trusted third-party scheme and cannot be used when there is no trusted third party. For example, a server that is not part of the domain, local Windows accounts, and resources is in an un-trusted domain. In these cases, NTLM authentication has to be used.

Alfresco CIFS and NTLM authentication

Microsoft CIFS clients and Mac Finder use NTLM for authentication and this is handled in Alfresco via an Enterprise CIFS Authenticator (this is the default CIFS authenticator after installation). It can handle a number of ways in which the CIFS client might want to connect:

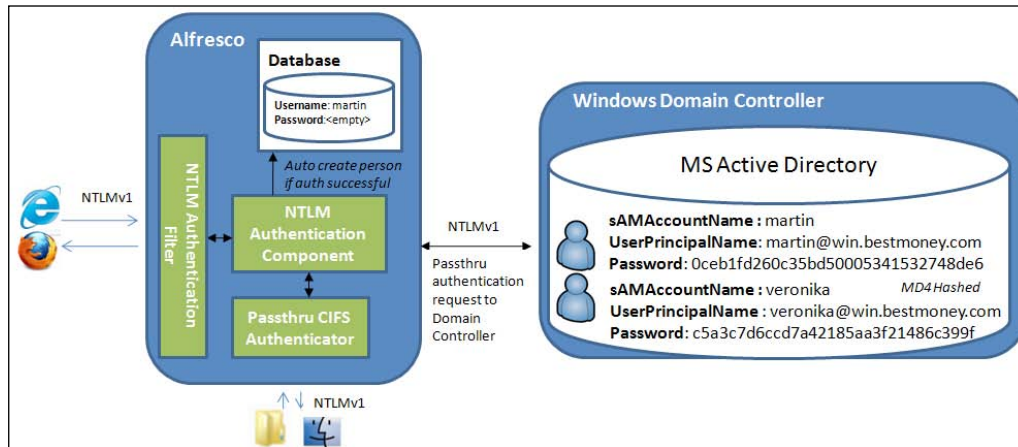
- **NTLMv1**: MD4 hashed passwords
- **NTLMv2**: MD5 hashed passwords/blob
- **NTLMSSP**: Two stage session setup with NTLMv1/NTLMv2
- **SPNEGO**: Session setup with Kerberos or NTLMSSP

Because CIFS uses NTLM or Kerberos, we usually do not have to log in more than the first time when we access Alfresco via the CIFS drive. Any later access of the drive, even if Alfresco is restarted, does not require a login.

Alfresco NTLM passthru authentication

In Alfresco, we can set up passthru authentication so the NTLM authentication is handled by a Windows domain controller or other server on the network, and not by Alfresco. This is called **passthru authentication** in the Alfresco world. To handle this in Alfresco, we use a different authentication component and another CIFS authenticator specially designed for passthru authentication.

Alfresco implements passthru authentication via an **NTLM Authentication Filter**, **NTLM Authentication Component** and a special **Passthru CIFS Authenticator**. The following diagram illustrates this:



When a client successfully logs in via NTLM passthru authentication for the first time, a minimal user account is also created in the Alfresco database. This is how a home folder can be associated with the user and so the user can easily be added to groups. It also makes it easier to set up permissions for the individual user.

When NTLM passthru authentication is used, we cannot use NTLMv2 as it will protect against man-in-the-middle attacks, and this is just what Alfresco is in this case.



Quite often passthru authentication is used in combination with LDAP authentication when an organization has Microsoft Active Directory installed. This is so CIFS can authenticate. It is also safer to let the passthru authenticator handle all authentications and let the LDAP authenticator handle only synchronization of users and groups; more on this later.

Kerberos authentication

Kerberos is a computer network authentication protocol that allows nodes communicating over a non-secure network to prove their identity to one another in a secure manner. Windows 2000 and later use Kerberos as their default authentication method. Most Unix-like operating systems such as Red Hat, Mac OS, Sun Solaris, AIX, and so on include software for Kerberos authentication.

Kerberos makes use of a trusted third party called the **Key Distribution Center (KDC)**, which contains two logically separate parts: an **Authentication Server (AS)** and a **Ticket Granting Server (TGS)**. Users and services log in via the AS and users request tickets to access services via TGS.

Some of the key points and characteristics of the Kerberos protocol are:

- The user's password never travels over the network.
- The user's password is never stored in any form on the client machine.
- The user's password is never stored in an unencrypted form even in the Authentication Server database.
- The user is asked to enter a password only once per work session. Therefore users can transparently access all the services they are authorized for without having to re-enter the password during this session, achieving single sign-on.
- Authentication information management is centralized and resides on the Authentication Server. The application servers such as Alfresco do not contain the authentication information for their users.
- The administrator can disable the account of any user by acting in a single location without having to act on the several application servers providing the various services.
- When a user changes his/her password, it is changed for all services at the same time.

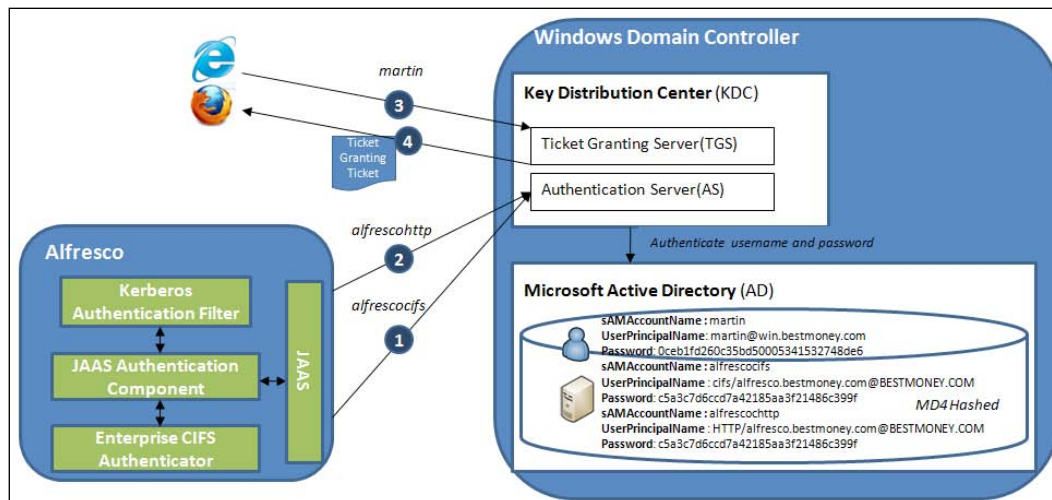
The following list contains definitions for the components and terms that we will come in contact with when managing a Kerberos installation:

- **Realm:** The term realm indicates an authentication administrative domain. It is used to establish the boundaries within which an authentication server has the authority to authenticate a user, host, or service. The name of a realm is case sensitive, but upper case letters are usually used. It is also good practice, in an organization, to make the realm name the same as the DNS domain, for example, BESTMONEY.COM.
- **Principal:** A principal is the name used to refer to the entries in the Authentication Server database. A principal is associated with each user, host, or service of a given realm. A principal in Kerberos 5 is of the following type: `component1/component2/.../componentN@REALM`. For an entry referring to a user, the principal takes the following form: `Name[/Instance]@REALM`. The instance is optional and is normally used to better qualify the type of user, such as administrator users normally have the admin instance. For example: `martin@BESTMONEY.COM`, `admin/admin@BESTMONEY.COM`. If, instead, the entries refer to services, the principals assume the following form: `Service/Hostname@REALM`. For example: `cifs/alfresco.bestmoney.com@BESTMONEY.COM`.

- **Ticket:** A ticket is something a client presents to an application server to demonstrate the authenticity of its identity. Tickets are issued by the Ticket Granting Server and are encrypted using the secret key of the service they are intended for.
- **GSS-API:** The **Generic Security Services Application Program Interface (GSSAPI)** is an application programming interface for programs to access security services. The main GSSAPI implementation in use is Kerberos.
- **GSS-SPNEGO:** SPNEGO (**Simple and Protected GSSAPI Negotiation Mechanism**) is a GSSAPI "pseudo mechanism" that is used when a client application wants to authenticate to a remote server, but neither end is sure what authentication protocols the other supports. SPNEGO's most visible use is in Microsoft's "HTTP Negotiate" authentication extension. It was first implemented in Internet Explorer 5.01 and IIS 5.0 and provided single sign-on capability, later marketed as Integrated Windows Authentication. The negotiable submechanisms included NTLM and Kerberos, both used in Active Directory.

User and service login via KDC AS

The first thing that happens in a Kerberos authentication is that users and services log in to the KDC via the Authentication Server:



The communication sequence in the preceding picture is as follows:

- I am the Alfresco CIFS server please authenticate me, my username is `alfrescocifs` and here is my password.
- I am the Alfresco HTTP server please authenticate me, my username is `alfrescohttp` and here is my password.
- I am the Martin user and need a **Ticket Granting Ticket (TGT)**. My username is `martin` and here is my password.
- Here is a TGT – if you can decrypt this response with your password hash. You can use it later to get service tickets for accessing Alfresco HTTP and CIFS.

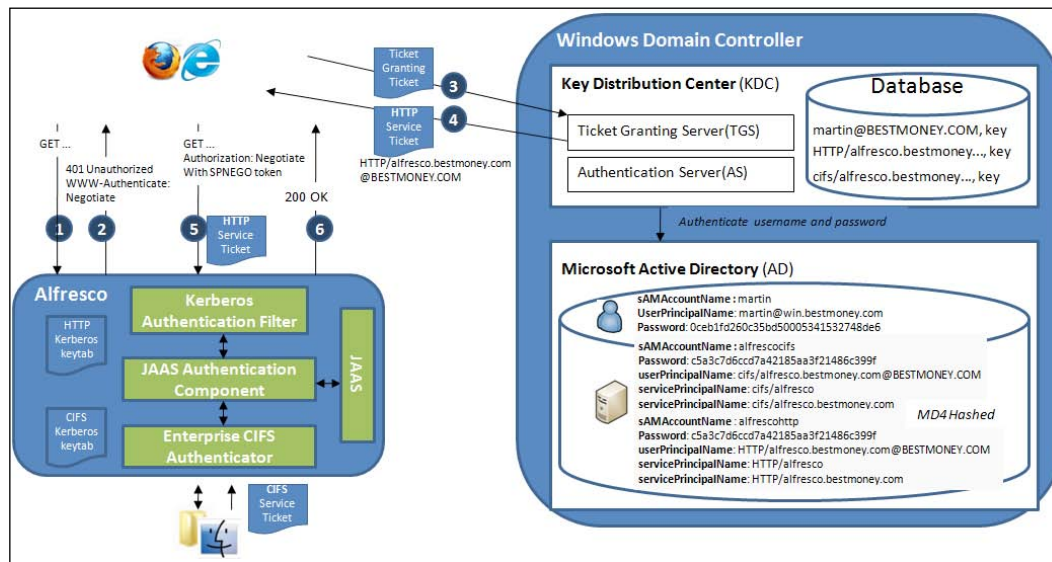
For this to happen, accounts must exist in Active Directory for user `martin` and for the `alfrescocifs` and `alfrescohttp` services. Users log in to the Windows domain at the start of the day and when the Alfresco server is started the CIFS and HTTP server will also log in to the KDC.

Accessing Alfresco via KDC TGS

Every server on the network has an implicit trust in the KDC, as they share a secret. Users gain access to the Alfresco HTTP or CIFS servers by presenting tickets with encrypted information from the KDC, which the server can verify.

As the KDC is the only place that knows every encryption key, it can share secrets with users and servers throughout the network to verify their authenticity. This way, as each principal trusts the KDC, the entire network is secure as long as the Kerberos server is secure.

The KDC contains a database of all users and services in the Kerberos realm. Each entry in this database is called a **SPN (Service Principal Name)**, and includes an associated encryption key. For users, this encryption key is derived from the user's password. Kerberos works with tickets that prove the identity of users and tickets are acquired via the Ticket Granting Server (TGS):



The communication sequence in the preceding picture is as follows:

1. User martin accesses Alfresco.
2. Alfresco responds with unauthorized, not logged in, and wants to negotiate about the authentication protocol to use.
3. Hi, I am user martin and here is my TGT, please give me a Service Ticket for Alfresco HTTP.
4. Here is your Alfresco HTTP Service Ticket (ticket and session key for both the client and the remote server – cached locally).
5. Here is my Service Ticket (session key for the remote server), authenticate me.
6. Client Server session starts.

As we can see, Kerberos is much more secure and it offloads the user from having to remember loads of passwords for different computers and different services on those computers.



Internet Explorer does not support the "WWW-Authenticate: Kerberos" header. It does support "WWW-Authenticate: Negotiate". The setting "Enable Integrated Windows Authentication" controls whether IE responds to the Negotiate header or not. This is particularly important because of one shortcoming of IE's authentication. As the Negotiate SSPI supports both Kerberos and NTLM, IE has the choice when presented with the Negotiate header of which authentication protocol to use. If you've set up your server and client correctly to enable Kerberos auth, it will use Kerberos over Negotiate; if you haven't, you'll get NTLM over Negotiate. The shortcoming is that there's no mechanism to restrict the Negotiate package to use only Kerberos, never NTLM.

LDAP authentication

If a company or organization is not a Microsoft shop then they will probably manage their users via an LDAP server such as Apache Directory Server or OpenLDAP on Linux.

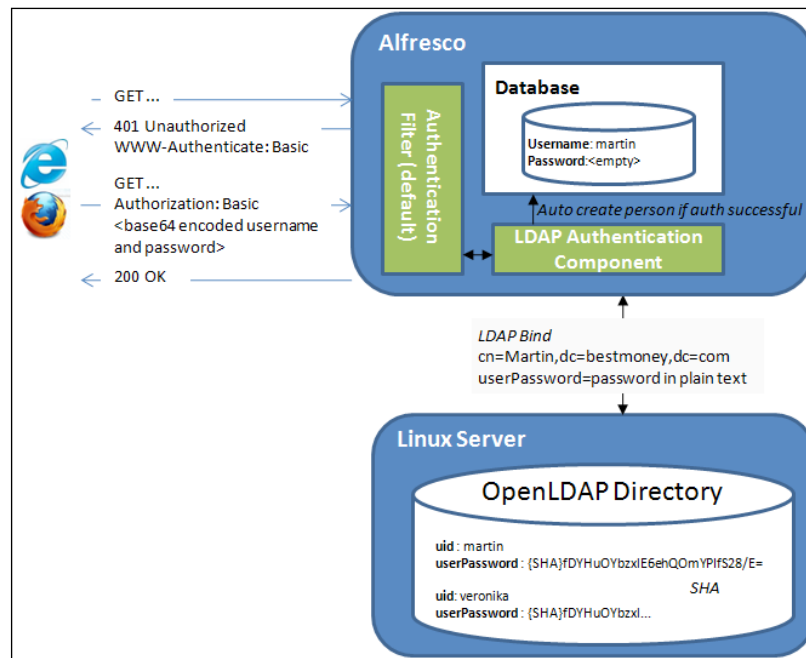


Microsoft Active Directory server is compatible with the LDAP protocol.

LDAP is primarily a directory access protocol. NTLM and Kerberos are on the other hand authentication protocols. They do different things. LDAP has a primitive authentication mechanism called "simple bind" that applications can use to verify credentials, if they can't handle other authentication protocols.

Simple binds can send a user's password in plain text, so it's a good idea to use SSL. In fact, Alfresco is configured to use simple LDAP authentication by default, so passwords will be sent in plain text.

The following diagram shows a typical setup when Alfresco is used with LDAP Authentication:



When a client successfully logs in via LDAP authentication for the first time, a minimal user account is also created in the Alfresco database. This is so a home folder can be associated with the user and so the user can easily be added to groups. It also makes it easier to set up permissions for the individual user.

As we can see, there is no support for authenticating CIFS connections. This is because there is no provider of MD4 hashed passwords and no component driving the NTLM authentication.

If CIFS is absolutely necessary – which it often is, being the main interface that users use when working with Alfresco – we can install Samba on the Linux box and configure it to use the LDAP server as the account database. To hook up Alfresco with the Samba server, we can use the NTLM passthru authentication mechanism.

If a Samba server is not an option for some reason, then we can extend the LDAP accounts with some extra Samba properties to store the password as MD4 or so. We can then build an extra NTLM Authenticator that uses the Samba object in LDAP to authenticate CIFS connections. An example of how to build such an authenticator will be presented later in this chapter.



LDAPv3 also includes an extensible authentication framework called **Simple Authentication and Security Layer (SASL)** that allows alternate authentication protocols to be added. If we are using Active Directory then it supports GSS-API (Kerberos), GSS-SPNEGO (Windows negotiate authentication, which selects between Kerberos and NTLM), Digest and External (for client cert auth). Thus, if the client understands any of those SASL mechanisms, it can actually use that for the authentication. As such, Kerberos may be used by an application during an LDAP bind operation if the client understands this.

Checking what SASL mechanisms the LDAP server supports

We can use an LDAP browser such as Apache Directory Studio to check the values of the `supportedSASLMechanisms` attributes on the root node of your LDAP server.

If we are using OpenLDAP, then we can do this query via the `ldapsearch` command line tool:

```
ldapsearch -h localhost -p 389 -x -b "" -s base -LLL
supportedSASLMechanisms
dn:
supportedSASLMechanisms: DIGEST-MD5
supportedSASLMechanisms: NTLM
supportedSASLMechanisms: CRAM-MD5
```



The simple bind authentication method will not be reported as it is not a SASL mechanism.

LDAP synchronization

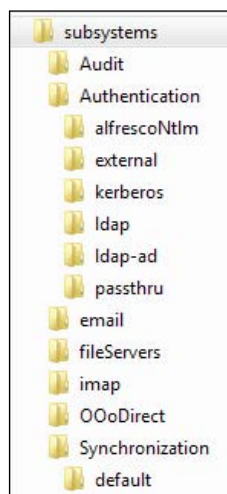
Alfresco can be set up to import users and groups from external directory services such as Microsoft Active Directory. This is called synchronizing Alfresco's database with the external directory. By default Alfresco will create minimal accounts in Alfresco even when an external directory service is used solely for authentication and user management (that is, synchronization is turned off).

However, to get First Name, Last Name, and Email Address imported from LDAP and set up for each new account, we have to set up synchronization properly.

Alfresco authentication and synchronization subsystems

Many organizations have more than one directory server and multiple ways for users to authenticate themselves to different services. To be able to handle this in Alfresco, so-called 'subsystems' are used to configure different authentication mechanisms and synchronization properties. Using subsystems is good as they are independent and they can be chained together to form an authentication chain, in which each authenticator will be tried until one succeeds.

Out of the box Alfresco comes with several pre-configured authentication and synchronization subsystems. Each authentication subsystem pretty much matches the authentication concepts and methods that we previously talked about in this chapter. In a new Alfresco installation, we can find these subsystem definitions in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems` location:



Subsystems are organized according to category (for example, Authentication) and type (for example, `alfrescoNtlm`). By default, Alfresco is configured to use the `alfrescoNtlm` authentication subsystem. This is set up in the `repository.properties` file (located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco` directory) with the `authentication.chain=alfrescoNtlm1:alfrescoNtlm` property.

The property value format is `logical-name:type,logical-name:type,...` where we can use the logical name as a reminder of what server or directory service the authentication subsystem is authenticating against.

The logical name of the authentication subsystem in this case is `alfrescoNtlm1` as it will just authenticate locally with Alfresco's database. The logical name of a subsystem instance can be anything we like as long as it is unique in the authentication chain.

The `alfrescoNtlm1` subsystem instance name suggests that by default Alfresco is actually set up with NTLM authentication. This is not so as NTLM (that is, SSO) is turned off by default, so basic authentication will be used instead.

This configuration setting can be found in the `ntlm-filter.properties` file (located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/alfrescoNtlm` directory) and the `ntlm.authentication.sso.enabled=false` property.

Alfresco authentication and application zones

When users and groups are imported into Alfresco, they are grouped under a so-called 'authentication zone'. A zone is related to the directory server from which the users and groups were imported. For example, the authentication configurations that we will set up in this chapter will result in the following authentication zones being created:

`AUTH.EXT.bestmoneyLDAP`

`AUTH.EXT.bestmoneyAD`

Here `AUTH` stands for authentication zone and `EXT` stands for authorities defined externally. The last name is the name we gave the authentication subsystem.

Authorities that have been defined and created in the Alfresco database are organized under the authentication zone `AUTH.ALF` (that is, they have not been synchronized/imported).

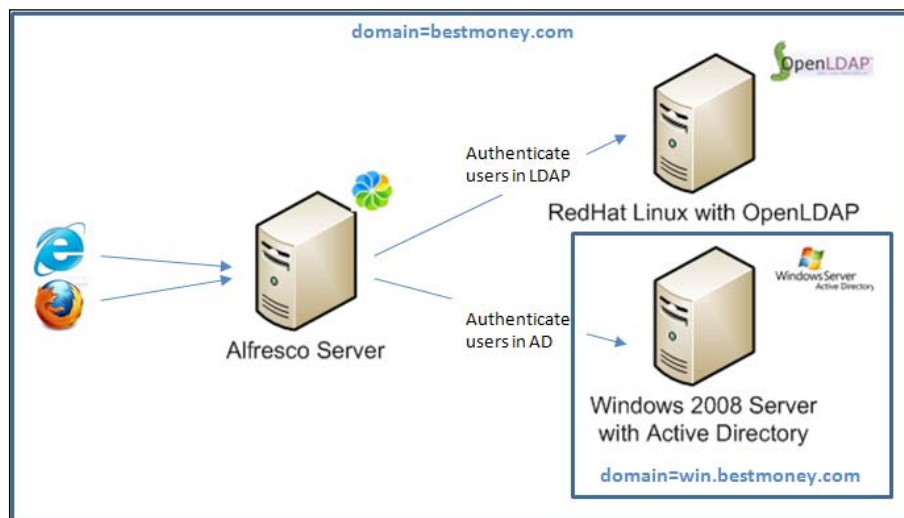
There are also application zones that are used to contain users and groups that should not show up in any of the user interfaces. Some of these zones are:

- APP.SHARE: All hidden authorities related to Alfresco Share
- APP.RM: All hidden authorities related to Alfresco Records Management

One application zone APP.DEFAULT is an exception—in that it contains all default users and groups that should show up in normal searches in the Alfresco UIs.

Setting up authentication and synchronization with Remote Directory servers

In this section, we will look at setting up Best Money's Alfresco server so it can authenticate against both the OpenLDAP server and the MS Active Directory server that they have:

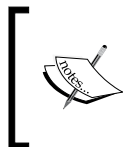


They also need First Name, Last Name, and Email Address from each account in the directory services to be imported into Alfresco, so we will see how to set up synchronization.

Configuring authentication and synchronization against OpenLDAP

To configure Alfresco to authenticate and synchronize against an OpenLDAP server means using the predefined `ldap` subsystem and customize the configuration for it. First let's add this subsystem to the authentication chain. To do this we do not change any files under `tomcat/webapps/alfresco` as those will be overwritten as soon as Tomcat decides to redeploy the web application; we install a new AMP, or upgrade Alfresco to a new version.

Instead un-comment the `authentication.chain=alfrescoNtlm1:alfrescoNtlm` property in the `alfresco-global.properties` file located in the `tomcat/shared/classes` directory.



The `alfresco-global.properties` file is the preferred file to do all custom configurations in. It will not be overwritten by any upgrades or redeployments and it is nice to have all custom configurations in one place.

Now change the property value so it uses the `ldap` authentication subsystem instead:

```
authentication.chain=bestmoneyLDAP:ldap
```

This disables anyone from logging in with an account in the Alfresco database.

Configuring user authentication with OpenLDAP

To set up authentication with OpenLDAP, we have to tell the `ldap` subsystem how to connect to the Best Money OpenLDAP server. Best Money also wants the guest login feature to be turned off and the OpenLDAP user Administrator to have administrator privileges in Alfresco.

The properties that we can configure for the `ldap` subsystem can be found in the `ldap-authentication.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Authentication/ldap` directory. We are particularly interested in the following properties:

- `ldap.authentication.allowGuestLogin`: Turn on and off guest access. Default is on which will bypass the login dialog when you hit the `http://localhost:8080/alfresco` URL.

- `ldap.authentication.userNameFormat`: This is a template that defines how Alfresco user IDs, that we enter in the login dialog, are mapped into OpenLDAP user principal names. The default LDAP authentication method is simple, so we can specify the template in clear text and use the `%s` placeholder variable where we want the Alfresco username to be inserted.
- `ldap.authentication.java.naming.provider.url`: This is the URL used to access the LDAP server.
- `ldap.authentication.defaultAdministratorUserNames`: A list of OpenLDAP user names that should be considered administrators in Alfresco. When we are turning off the `alfrescoNtlm` authentication subsystem we do not have an admin anymore, so we need to set one up. It is enough to have one user as admin as he or she can add more admin users by adding their usernames to the `ALFRESCO_ADMINISTRATORS` group.

Copy these four properties to the `alfresco-global.properties` file and set the property values as follows:

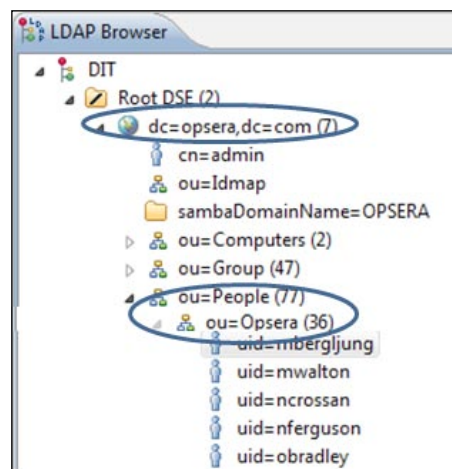
```
ldap.authentication.allowGuestLogin=false
```

```
ldap.authentication.userNameFormat= uid\=%s,ou\=People,dc\  
=bestmoney,dc\=com
```

```
ldap.authentication.java.naming.provider.url= ldap://ldap.bestmoney.  
com:389
```

```
ldap.authentication.defaultAdministratorUserNames=Administrator
```

To figure out what `userNameFormat` to use, we can use an LDAP client like Apache Directory Studio and navigate to where the users are stored. We should see something like the following screenshot:



Here it is easy to see what organizational unit (`ou=Opsera,ou=People`) the users are under and what domain component (`dc=opsera,dc=com`) the organizational unit is under.

Before starting Alfresco, we need to turn off synchronization as it is turned on by default and we have not configured enough properties for it to work. So copy the following property to the `alfresco-global.properties` file and set it to false:

```
ldap.synchronization.active=false
```

We will be able to start Alfresco now and it will test the access to the LDAP server. We should see the following lines in the log if things went okay:

```
18:28:39,842 INFO [management.subsystems.
ChildApplicationContextFactory] Starting 'Authentication' subsystem, ID:
[Authentication, managed, bestmoneyLDAP]
18:28:40,325 INFO [management.subsystems.
ChildApplicationContextFactory] Startup of 'Authentication' subsystem,
ID: [Authentication, managed, bestmoneyLDAP] complete
18:28:40,326 WARN [org.alfresco.fileserver] No enabled CIFS
authenticator found in authentication chain. CIFS Server disabled
```

It is now possible to log in to Alfresco via an LDAP account and we should see that the guest login is not active, which means that the login dialog is always displayed when accessing Alfresco.

Worth noting here is that the CIFS server has been turned off as there is no authentication subsystem available to support NTLM authentication or provide MD4 hashed passwords.

Configuring user and group synchronizing with OpenLDAP

If we look at the bare bones user account created after a successful login to LDAP, it contains only the username, which is also set as first name. It would be nice to import First Name, Last Name, and Email Address from LDAP and have it set in the Alfresco account database.

For this we need to turn on synchronization again and configure a couple more properties, so Alfresco can log in to the LDAP server and do queries for user and group information:

```
ldap.synchronization.active=true

ldap.synchronization.java.naming.security.principal= cn\=alfresco,ou\
=Services,dc\=bestmoney,dc\=com

ldap.synchronization.java.naming.security.credentials=<password>
```

It is a good idea to create a new user in LDAP, such as `alfresco` in this case, which is used by Alfresco to log in and query for user and group information. This user is also not put under the `People` organizational unit but under `Services` instead to distinguish it from normal users. This service user has to have access to all users and groups in LDAP.

Besides these properties, we also need to set up the search base in LDAP for where to find users and groups (if you are unsure about what search base to use then navigate to the users and groups from Apache Directory Studio and you will see what base to use):

```
ldap.synchronization.userSearchBase=ou\=People,dc\=bestmoney,dc\=com
```

```
ldap.synchronization.groupSearchBase=ou\=Groups,dc\=bestmoney,dc\=com
```

Copy these five properties to the `alfresco-global.properties` file and restart Alfresco.

The logs should look something like this now:

```
19:05:47,057 User:System INFO [security.sync.
ChainingUserRegistrySynchronizer] Synchronizing users and groups with
user registry 'bestmoneyLDAP'
...
19:05:49,474 WARN [security.sync.ChainingUserRegistrySynchronizer]
Updating group 'ALFRESCO_ADMINISTRATORS'. This group will in future be
assumed to originate from user registry 'bestmoneyLDAP'.
...
19:05:52,487 User:System INFO [security.sync.
ChainingUserRegistrySynchronizer] bestmoneyLDAP Group Creation and
Association: Processed 53 entries out of 53. 100% complete. Rate: 20 per
second. 0 failures detected.
...
19:05:53,778 User:System WARN [security.sync.
ChainingUserRegistrySynchronizer] Updating user 'martin'. This user will
in future be assumed to originate from user registry 'bestmoneyLDAP'.
...
19:05:58,841 User:System INFO [security.sync.
ChainingUserRegistrySynchronizer] opseraLDAP User Creation and
Association: Processed 31 entries out of 31. 100% complete. Rate: 4 per
second. 0 failures detected.
...
19:05:58,948 User:System INFO [security.sync.
ChainingUserRegistrySynchronizer] Finished synchronizing users and groups
with user registry 'opseraLDAP'
```

The user `martin` that I logged in with, when synchronization was turned off– has now been updated with First Name, Last Name, and Email Address and all other users and groups have been imported into the Alfresco database.

If the First Name, Last Name, and Email Address has not been imported correctly for some reason, then have a look at the following properties and make sure they match the properties in LDAP:

```
ldap.synchronization.userIdAttributeName=uid
```

```
ldap.synchronization.userFirstNameAttributeName=givenName
```

```
ldap.synchronization.userLastNameAttributeName=sn
```

```
ldap.synchronization.userEmailAttributeName=mail
```

```
ldap.synchronization.userOrganizationalIdAttributeName=o
```

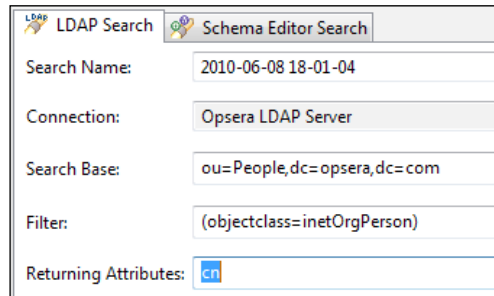
Open up a user via, for example, Apache Directory Studio and check that the LDAP user properties match:

sn	Bergljung
sshPublicKey	Binary Data (495 Bytes)
uid	mbergljung
uidNumber	1004
displayName	Martin Bergljung
gecos	Martin Bergljung
givenName	Martin
loginShell	/bin/bash
mail	martin.bergljung@opsera.com

We can also verify that the user's query set by the following property really works:

```
ldap.synchronization.personQuery=(objectclass\=inetOrgPerson)
```

Use Apache Directory Studio as follows:



LDAP Search	
Search Name:	2010-06-08 18-01-04
Connection:	Opsera LDAP Server
Search Base:	ou=People,dc=opsera,dc=com
Filter:	(objectclass=inetOrgPerson)
Returning Attributes:	cn

This should give us a list of all the users if things are set up correctly in the LDAP installation. We can test the groups query in the same way:

```
ldap.synchronization.groupQuery=(objectclass\=groupOfNames)
```

After the first initial import of users and groups, Alfresco does delta imports, meaning it will only import when LDAP information changes (that is, new users, new groups, delete group, update user, and so on). The queries that handle this can be configured with the following properties:

```
ldap.synchronization.personDifferentialQuery=(&(objectclass\=inetOrgPerson)(!(modifyTimestamp<=\{0\})))
```

```
ldap.synchronization.groupDifferentialQuery=(&(objectclass\=groupOfNames)(!(modifyTimestamp<=\{0\})))
```

There are also a couple of other synchronization-related properties that are useful to know about. These properties cannot be found in the LDAP properties file but instead in the `default-synchronization.properties` file located in the synchronization subsystem directory (that is, `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Synchronization/default`):

- `synchronization.synchronizeChangesOnly`: This determines whether the delta updates/differential synchronizing should be turned on or off (default is on).
- `synchronization.import.cron`: The time when there should be a cron job kicking off an import of users and groups (default is to kick off an import every night).
- `synchronization.syncWhenMissingPeopleLogIn`: This specifies if the system should trigger a delta/differential import for a user if they login successfully but the First Name, Last Name, and Email is missing in the Alfresco database (default is true).

- `synchronization.syncOnStartup`: This tells the system if an import should be performed as soon as Alfresco has started up (default is true).
- `synchronization.autoCreatePeopleOnLogin`: This is true if a user should be autocreated in the Alfresco database, if missing, after a successful authentication with LDAP.

Now let's move on and hook up the Alfresco server with Best Money's Microsoft Active Directory server.

Configuring authentication and synchronization against Microsoft Active Directory

To configure Alfresco to authenticate and synchronize with an MS Active Directory server, we have to use the predefined `ldap-ad` subsystem and customize the configuration for it.

Let's start by adding this subsystem to the authentication chain. Open up the `alfresco-global.properties` file located in the `tomcat/shared/classes` directory. Now change the property value so it also uses the `ldap-ad` authentication subsystem:

```
authentication.chain=bestmoneyLDAP:ldap,bestmoneyAD:ldap-ad
```

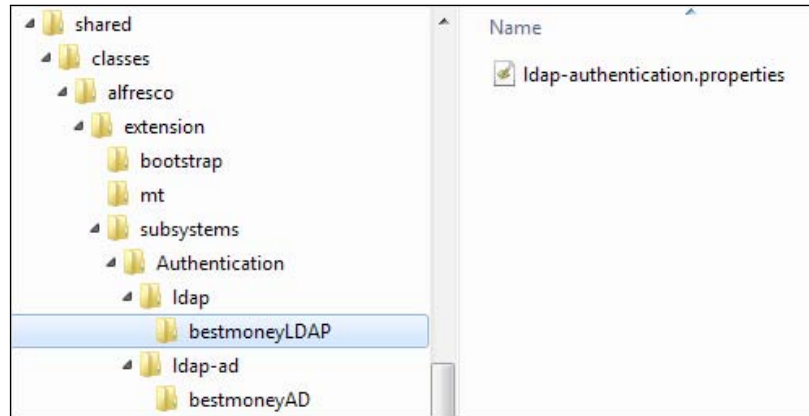
This will make Alfresco first try the OpenLDAP server to authenticate a user, and if that does not work, Alfresco will try and authenticate the user against the Active Directory server.

Configuring multiple LDAP authentication subsystems

To set up authentication with **Active Directory (AD)** we have to tell the `ldap-ad` subsystem how to connect to the Best Money AD server. This LDAP subsystem uses the same properties as the `ldap` subsystem that we just used to connect to OpenLDAP.

This presents a problem. How do we set up these properties to connect to two LDAP servers at the same time? The solution is to **not** keep the properties in the `alfresco-global.properties` file but instead in separate directories under `tomcat/shared`.

Create the `ldap/bestmoneyLDAP` and `ldap-ad/bestmoneyAD` directories, as shown in the following screenshot:



Copy the `ldap-authentication.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Authentication/ldap` directory to the `ldap/bestmoneyLDAP` directory and then copy the `ldap-ad-authentication.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Authentication/ldap-ad` directory to the `ldap-ad/bestmoneyAD` directory.

The structure of these directories follows this template: `alfresco/extension/subsystems/[category]/[type]/[logical name]` where logical/instance name is the name we specified in the `alfresco-global.properties` file.

Moving OpenLDAP subsystem configuration to its own directory

After this, we need to remove the `ldap` subsystem properties from the `alfresco-global.properties` file and set them in the properties file located in the `tomcat/shared/.../ldap/ldap-authentication.properties` file. The global property file now only contains the following property that sets up the authentication chain:

```
authentication.chain=bestmoneyLDAP:ldap,bestmoneyAD:ldap-ad
```

Note that the property that turns off guest login should be set to `false` in both LDAP properties configuration files:

```
ldap.authentication.allowGuestLogin=false
```



Any property configuration in any file in a subdirectory to the `tomcat/shared/classes/alfresco/extension` directory will override the same property configuration in the `tomcat/shared/classes/alfresco-global.properties` file.

Configuring authentication and synchronization with Active Directory

Now open up the `tomcat/shared/.../ldap-ad/ldap-ad-authentication.properties` file and set up the properties to connect to the AD server:

```
ldap.authentication.userNameFormat=%s@win.bestmoney.com

ldap.authentication.java.naming.provider.url=ldap://ad.bestmoney.com:389

ldap.synchronization.java.naming.security.principal=alfresco@win.bestmoney.com

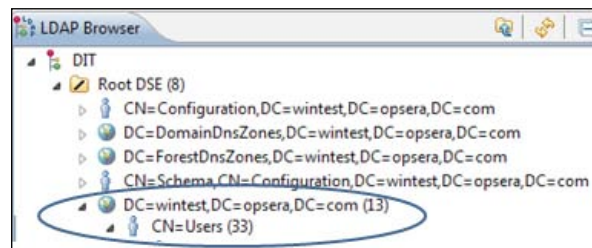
ldap.synchronization.java.naming.security.credentials=<password>

ldap.synchronization.userSearchBase=CN\=Users,dc\=win,dc\=bestmoney,dc\=com

ldap.synchronization.groupSearchBase=CN\=Users,dc\=win,dc\=bestmoney,dc\=com
```

Best Money's AD server has all groups stored under the `Users` container, so that's why the search base queries are the same for users and groups.

If we are unsure about the `userNameFormat` and the `userSearchBase` then we can use the Apache Directory Studio to find this out. If we connect and navigate to the users, we should see something like the following screenshot:



In this case, we can see that the `userSearchBase` must be `CN\=Users, dc\=wintest, dc\=opsera, dc\=com` and if we have a look at one of the users, we can figure out the user name format:



In this case, it should be set up as `%s@wintest.opsera.com` where `%s` will contain the username that is entered in the Alfresco login dialog.

If we start up Alfresco, we should see a log where both the Best Money authentication subsystems are loaded:

```
11:52:07,711 INFO [alfresco.config.JndiPropertiesFactoryBean]
Loading properties file from file [C:\Alfresco\tomcat\shared\classes\
alfresco\extension\subsystems\Authentication\ldap\bestmoneyLDAP\ldap-
authentication.properties]
11:52:07,749 INFO [alfresco.config.JndiPropertiesFactoryBean] Loading
properties file from file [C:\Alfresco\tomcat\shared\classes\alfresco\
extension\subsystems\Authentication\ldap-ad\bestmoneyAD\ldap-ad-
authentication.properties]
...
19:48:55,813 INFO [org.alfresco.repo.management.subsystems.
ChildApplicationContextFactory] Starting 'Authentication' subsystem, ID:
[Authentication, managed, bestmoneyLDAP]
19:48:55,836 INFO
[org.alfresco.repo.management.subsystems.ChildApplicationContextFactory]
Startup of 'Authentication' subsystem, ID: [Authentication, managed,
bestmoneyLDAP] complete
19:48:56,286 INFO [org.alfresco.repo.management.subsystems.
ChildApplicationContextFactory] Starting 'Authentication' subsystem, ID:
[Authentication, managed, bestmoneyAD]
19:48:56,881 INFO [org.alfresco.repo.management.subsystems.
ChildApplicationContextFactory] Startup of 'Authentication' subsystem,
ID: [Authentication, managed, bestmoneyAD] complete
19:48:56,882 WARN [org.alfresco.fileserver] No enabled CIFS
authenticator found in authentication chain. CIFS Server disabled
```

We should now be able to log in to Alfresco with any user in either Best Money's OpenLDAP server or in Best Money's Active Directory server.

Customizing group imports

If we want to narrow down the number of groups that are imported into Alfresco, we can do that by extending the query as in the following example:

```
ldap.synchronization.groupQuery=(&(objectclass\=group)(CN\=Denied
RODC Password Replication Group))
ldap.synchronization.groupDifferentialQuery=(&(objectclass\
=group)(CN\= Denied RODC Password Replication Group)(!(modifyTimestam
p<\={0})))
```

This example imports only those groups that are members of the Denied RODC Password Replication Group. Don't forget to also specify the differential query in the same way. To find out what containers/groups an entry is a member of check out the following attribute:

memberOf	CN=Denied RODC Password Replication Group,CN=Users,DC=wintest,DC=opsera,DC=com

Accessing via the CIFS interface

We now got authentication and synchronization working with OpenLDAP and Active Directory. However, we cannot connect via CIFS to the Alfresco server and Best Money requires that all users in both the directory servers are able to access Alfresco via CIFS.

We could add passthru authentication with the Windows Domain Controller, which runs Active Directory, and then any user in AD would be able to connect to Alfresco via CIFS.

This does not solve the problem for the OpenLDAP users, but it is needed by the AD users. So to configure passthru authentication for AD users add the `bestmoneyADPassthru` authentication component to the authentication chain specified in `alfresco-global.properties`:

```
authentication.chain=bestmoneyLDAP:ldap,bestmoneyAD:ldap-
ad,bestmoneyADPassthru:passthru
```

```
passthru.authentication.domain=win.bestmoney.com
```

```
passthru.authentication.servers=ad.bestmoney.com
```

For passthru to work, we need to also specify what Windows Domain Controller (the machine that runs AD) server to connect to when doing NTLM authentication. This is done with the `passthru.authentication.servers` property.

We can specify all passthru properties in the `alfresco-global.properties` file as there is only one passthru authentication subsystem in use.

The passthru properties can be copied from the `passthru-authentication-context.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Authentication/passthru` directory.

Starting Alfresco should display the following in the logs:

```
15:49:32,843 INFO [management.subsystems.  
ChildApplicationContextFactory] Starting 'Authentication' subsystem, ID:  
[Authentication, managed, bestmoneyADPassthru]  
15:49:34,316 INFO [management.subsystems.  
ChildApplicationContextFactory] Startup of 'Authentication' subsystem,  
ID: [Authentication, managed, bestmoneyADPassthru] complete
```

It is now possible for a user in Best Money's Active Directory to connect to Alfresco via the CIFS interface. However, if we now try to log in via a web browser that might not work as Alfresco will try and do an automatic login via NTLM as the `ntlm.authentication.sso.enabled` property is set to true by default in the passthru subsystem.

This setting overrides the basic authentication that the `bestmoneyLDAP` and `bestmoneyAD` subsystem supports. And if the windows user credentials do not match any user in the AD directory then it will not be possible to log in via the web browser. So disable SSO by setting the `ntlm.authentication.sso.enabled=false` in the `alfresco-global.properties` file. After a restart of Alfresco, we should get the login dialogs again in the browser and it should be possible to log in.

Best Money does not have a Samba server installed on the Linux box that runs OpenLDAP, so we cannot use that approach for passthru authentication with OpenLDAP. We need to come up with another solution such as creating a custom Authenticator that can authenticate against an extra Samba object stored with each LDAP account.



If there was a Samba server available to connect to OpenLDAP, then we could just add that server address to the `passthru.authentication.servers` property and the problem is solved.

In this case, we should also turn off authentication for the two LDAP subsystems and have authentication done via the passthru authentication subsystems as it is much more secure NTLM authentication than the LDAP simple bind.

Implementing a custom authenticator for CIFS authentication against an LDAP server

For users to be able to use the CIFS interface, we normally need to have Alfresco set up to authenticate with its internal database or via an MS Active Directory. Both these databases/directories support MD4 hashed passwords and both servers support NTLM authentication.

This is not available when we use an LDAP server such as OpenLDAP. We have to come up with a solution that:

- Provides a directory that supports MD4 hashed passwords
- Supports NTLM authentication

Nothing like this exists out of the box, but one possible solution is to add an extra Samba object to the LDAP structure and then create an authenticator that can authenticate against this new Samba account. What we need to do to implement this solution is:

- Add a `sambaSamAccount` object to each OpenLDAP user's account
- Have each user create MD4 hashed passwords via a custom change password dialog
- Implement an NTLM Authenticator that uses the `sambaSamAccount` object for authentication
- Configure a new subsystem and add to the authentication chain

This is not an ideal solution as each user would have to reset their password to have the new `sambaSamAccount` generated, but it is a good solution if a Samba server is not available to hook up to OpenLDAP. It can also be a good solution if there is a problem getting the Samba and LDAP combination to work.

Adding a `sambaSamAccount` to LDAP structure

Each LDAP user needs to store the password as an MD4 hash and we will solve this by adding a `sambaSamAccount` object class to the user profile. This object class and its attributes are defined in the `samba.schema` file, which is part of the `samba-doc` package.

To install this new schema, have a look at the documentation for your Linux distribution in the sections talking about how to add an LDAP schema to OpenLDAP.

When this Samba schema is installed, we should see something like the following screenshot for an LDAP user account entry:

objectClass	sambaSamAccount (auxiliary)
objectClass	shadowAccount (auxiliary)
objectClass	top (abstract)
alfrescoImport	TRUE
cn	Martin Bergljung
gidNumber	1004
homeDirectory	/home/mbergljung
sambaSID	S-1-5-21-3068138994-4050941570-20
sn	Bergljung
sshPublicKey	Binary Data (495 Bytes)
uid	mbergljung
uidNumber	1004
displayName	Martin Bergljung
gecos	Martin Bergljung
givenName	Martin
loginShell	/bin/bash
mail	martin.bergljung@opsara.com
sambaAcctFlags	[U]
sambaBadPasswordCount	0
sambaBadPasswordTime	0
sambaLMPassword	B25665C2F588339B3D704C80C060D1A6
sambaNTPassword	64F0A7D930BDA66FCCF2D98DA70BAD22

A `sambaSamAccount` has been added to this `martin` entry and he has also changed his password via the new password change dialog that updates the LM and NT password hashes.

Generating MD4 passwords

Most companies and organizations have a page where its employees can reset or change their password. Best Money's OpenLDAP users have a special custom built **Change Password** page. What we need to do is update the logic behind it to set up the Samba account attributes (specifically generate the MD4 password hash) whenever a user changes his or her password.

This is out of the scope for this chapter.

Building a custom NTLM authenticator

So, now we have got the LDAP user entries prepared with Samba accounts and there is a way for users to update their LDAP entries with Samba account information. Let's implement an authenticator that can use this new Samba account for authentication.

This authenticator is basically the same as Alfresco's standard authentication component except that we want to get the password MD4 hash from LDAP instead of from Alfresco's database. The easiest way to implement our authenticator is to extend Alfresco's standard authenticator and then re-implement the method that gets the MD4 hash.

Start by creating a new authentication component class as follows:

```
public class LdapCifsAuthenticationComponentImpl
    extends AuthenticationComponentImpl {
```

The new authentication component class extends the `org.alfresco.repo.security.authentication.AuthenticationComponentImpl` class, which in turn implements the `NTLMAuthenticator` interface. This is the key to enabling the authenticator for CIFS authentication.

Now add the Spring LDAP Template class `org.springframework.ldap.core.LdapTemplate` so we can talk to the LDAP server. We also need to store the LDAP base for user entries:

```
private LdapTemplate m_ldapTemplate;
private String m_userBase;
```

The LDAP Template class is available in Spring's LDAP library (for example, `spring-ldap-core-1.3.0.jar`). Add a constructor that calls the super class and Spring DI setters as follows:

```
public LdapSambaAuthenticationComponentImpl() { super(); }
public void setLdapTemplate(LdapTemplate ldapTemplateObj) {
    m_ldapTemplate = ldapTemplateObj; }
public void setUserBase(String userBase) { m_userBase =
    userBase; }
```

Next thing we need to do is to override the method that gets the MD4 hash:

```
@Override
public String getMD4HashedPassword(String userName) {
    final SambaSamAccount sam = getSambaSamAccount(userName);
    if (sam == null) {
        logger.error("SambaSamAccount was null for user " +
            userName + ", please set it up in LDAP directory.");
        return null;
    } else {
        return sam.getSambaNtPassword();
    }
}
```


This method calls the `getSambaSamAccount` method with the passed in username. The `getSambaSamAccount` method will do the job of talking to LDAP and extracting the MD4 hash for the username and put it in a custom mapping class called `SambaSamAccount`:

```
private SambaSamAccount getSambaSamAccount(String uid) {
    String andFilter = new AndFilter()
        .and(new EqualsFilter(SambaSamAccount.ATTR_OBJECT_CLASS,
            SambaSamAccount.OBJECT_CLASS))
        .and(new EqualsFilter(SambaSamAccount.ATTR_UID,
            uid)).encode();
```

The first thing we do is to create an LDAP search filter that will extract information from LDAP entries with `objectClass=sambaSamAccount` and attribute `uid=username`. So if the username matches an entry and this entry has the `sambaSamAccount` object class then it will match.

A mapper class will be used to pass the LDAP data from the LDAP context to the Authenticator Java class context:

```
ContextMapper mapper = new ContextMapper() {
    public Object mapFromContext(final Object o) {
        final String dUid = ((DirContextOperations)
            o).getStringAttribute(SambaSamAccount.ATTR_UID);
        final String dHash = ((DirContextOperations)
            o).getStringAttribute(SambaSamAccount.ATTR_NT_PASS);
        return new SambaSamAccount(dUid, dHash);
    }
};
```

The mapper extracts the data from LDAP and puts it in this new custom class `SambaSamAccount` that we will create in a second. The last thing that needs to be done is to do the search with the user base, filter, and the context mapper:

```
try {
    return (SambaSamAccount)
        m_ldapTemplate.searchForObject(m_userBase, andFilter, mapper);
} catch (Exception e) {
    logger.error(e.getClass().getSimpleName() + " when searching for
        SambaSamAccount for user " + uid +
        " [message=" + e.getMessage() + "]");
    return null;
}
}
```

Finally, create the mapper class as follows:

```
public class SambaSamAccount {
    public static final String OBJECT_CLASS = "sambaSamAccount";
    public static final String ATTR_OBJECT_CLASS = "objectClass";
    public static final String ATTR_NT_PASS = "sambaNTPassword";
    public static final String ATTR_UID = "uid";

    private final String m_sambaNtPassword;
    private final String m_uid;

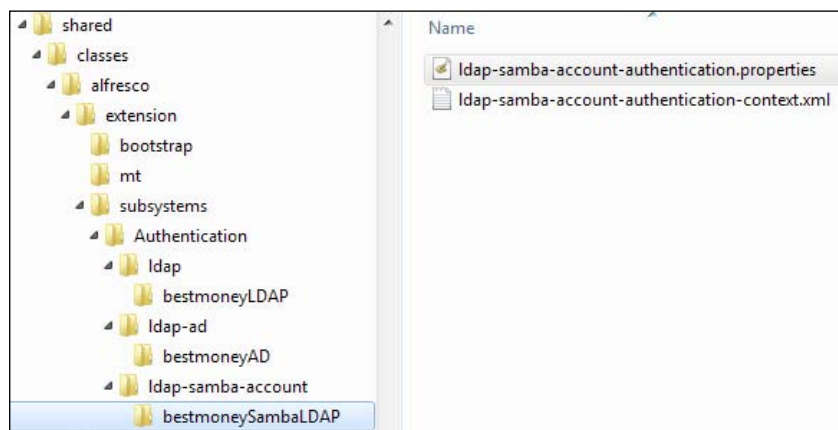
    public SambaSamAccount(final String uid, final String hash) {
        m_uid = uid;
        m_sambaNtPassword = hash;
    }

    public String getUid() { return m_uid; }
    public String getSambaNtPassword() { return
        m_sambaNtPassword; }
}
```

We can put these two classes in a new package called `com.bestmoney.cms.authentication` located in the following directory `bestmoney/alf_extensions/trunk/_alfresco/source/java/com/bestmoney/cms/authentication`.

Custom authentication subsystem configuration

To configure this new authenticator, we can create a new subsystem directory in the same place as the other subsystems we have configured. Add a subsystem called **bestmoneySambaLDAP** under the new authenticator type **ldap-samba-account** under the **Authentication** subsystem category:



Add a property file called `ldap-samba-account-authentication.properties` in the `bestmoneySambaLDAP` directory. Then add the following properties to it:

```
ldap.samba.authentication.java.naming.provider.url=ldap://ldap.
bestmoney.com:389
```

```
ldap.samba.authentication.base=dc=bestmoney,dc=com
```

```
ldap.samba.authentication.userbase=ou=People
```

```
ldap.samba.java.naming.security.principal=cn\=alfresco,ou\
=Services,dc\=bestmoney,dc\=com
```

```
ldap.samba.java.naming.security.credentials=<password>
```

```
alfresco.authentication.authenticateCIFS=true
```

```
alfresco.authentication.allowGuestLogin=false
```

We are making most of these properties unique (that is, using `ldap.samba`) so they can be set independently of the standard LDAP properties if needed. These properties are set in the associated Spring configuration file `ldap-samba-account-authentication-context.xml`.

A good starting point for this file is the `alfresco-authentication-context.xml` Spring configuration file from the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/Authentication/alfrescoNtlm` directory. After copying the contents of this file, add the following LDAP-related bean definitions to the beginning of the file:

```
<beans>
  <bean id="ldapContextSource" class=
    "org.springframework.ldap.core.support.LdapContextSource">
    <property name="url"
      value="\${ldap.samba.authentication.java.naming.provider.url}"/>
    <property name="base" value=
      "\${ldap.samba.authentication.base}"/>
    <property name="userDn"
      value="\${ldap.samba.java.naming.security.principal}"/>
    <property name="password"
      value="\${ldap.samba.java.naming.security.credentials}"/>
  </bean>

  <bean id="ldapTemplate" class=
    "org.springframework.ldap.core.LdapTemplate">
```

```

    <constructor-arg ref="ldapContextSource"/>
    <property name="ignorePartialResultException"
        value="true"/>
</bean>

```

This defines the two LDAP beans that the custom authentication component needs when talking to OpenLDAP. Now add the bean for the custom authenticator by copying the definition from the existing `authenticationComponent` bean. Then change the class name to the new custom class and add two properties for the LDAP template and the user base:

```

<bean id="authenticationComponent"
    class="com.bestmoney.cms.authentication.LdapSambaAuthentication
    ComponentImpl"
    parent="authenticationComponentBase">
    <property name="ldapTemplate" ref="ldapTemplate"/>
    <property name="userBase"
        value="${ldap.samba.authentication.userbase}"/>
    <property name="authenticationManager">
        <ref bean="authenticationManager"/>
    </property>
    <property name="allowGuestLogin">
        <value>${alfresco.authentication.allowGuestLogin}</value>
    </property>
    <property name="nodeService">
        <ref bean="nodeService"/>
    </property>
    <property name="personService">
        <ref bean="personService"/>
    </property>
    <property name="transactionService">
        <ref bean="transactionService"/>
    </property>
</bean>

```

Last thing we need to do is to comment out the original `authenticationComponent` bean definition and we are ready to go.

Deploying the needed classes for the custom authenticator

If we have both the `LdapSambaAuthenticationComponentImpl.java` and the `SambaSamAccount.java` files in the `bestmoney/alf_extensions/trunk/_alfresco/source/java/com/bestmoney/cms/authentication` directory, it is easy to create a JAR that we can deploy.

Just run the `package-alfresco-jar ant` target and it will create a JAR file in the `3340_04_Code\bestmoney\alf_extensions\trunk\build\lib` directory that can be dropped into the `tomcat/webapps/alfresco/WEB-INF/lib` directory.

Then restart the Alfresco server.

Testing the new custom authenticator

It should now be possible for users in the OpenLDAP directory to also use the CIFS interface to interact with Alfresco.

Making authentication more secure and using SSO

At the moment, the **bestmoneyLDAP** and **bestmoneyAD** authentication subsystems are handling the authentication. This is not secure as a normal LDAP simple bind is used where the passwords are sent in plain text.

It would be better if we could let the NTLM authenticators (that is, **bestmoneyADPassthru** and **bestmoneySambaLDAP**) handle all authentications, not just CIFS authentications. We can do this by turning off authentication for the **bestmoneyLDAP** and **bestmoneyAD** subsystems.

Set the following property to false for both of these subsystems:

```
ldap.authentication.active=false
```

This effectively makes the **bestmoneyLDAP** subsystem and the **bestmoneyAD** subsystem to only manage synchronization of user and group information.

After restarting Alfresco users in the OpenLDAP directory, we will be able to login to Alfresco via a browser. This works as the **bestmoneySambaLDAP** authentication subsystem is pretty much the same as the **alfrescoNtlm** subsystem and therefore supports the standard Authentication Filter that supports HTTP basic authentication.

Active Directory users will not be able to log in via a browser as the **bestmoneyADPassthru** subsystem does not support HTTP basic authentication but only NTLM authentication and Single Sign-on (SSO). NTLM authentication is turned off by us in the `alfresco-global.properties` file:

```
ntlm.authentication.sso.enabled=false
```

```
authentication.chain=bestmoneyLDAP:ldap,bestmoneyAD:ldap-ad,bestmoneyADPassthru:passthru,bestmoneySambaLDAP:ldap-samba-account
```

```
passthru.authentication.domain=win.bestmoney.com
```

```
passthru.authentication.servers=ad.bestmoney.com
```

When NTLM is turned off, the associated NTLM Servlet Filter is also turned off so users will not be logged in. So change the configuration and turn on NTLM by commenting out the `...sso.enabled` configuration line:

```
#ntlm.authentication.sso.enabled=false
```

Now when Internet Explorer (IE) is used to browse to the Alfresco site, we are logged in automatically if the Windows user has an account in Active Directory with matching username and password.

If we use FireFox (FF) to browse to the Alfresco site, we will be presented with a login dialog as FF has to be configured to pick up the Windows credentials automatically. If we logout the following dialog will be displayed:



If we click the **Re-login** link we will be automatically logged in. To set up the Alfresco site to be trusted in FF and NTLM to be enabled correctly, enter `about:config` in the URL field and set up the following property:

<code>network.automatic-ntlm-auth.trusted-uris</code>	user set	string	<code>http://localhost:8080/alfresco</code>

Now FF users will also get logged in automatically.

Troubleshooting NTLM authentication and SSO

If we access Alfresco via the IE or FF browser and get the following error:

```
net.sf.acegisecurity.AuthenticationServiceException: Failed to open
passthru auth session
at org.alfresco.repo.security.authentication.ntlm.
NTLMAuthenticationComponentImpl.authenticatePassthru(NTLMAuthenticationCo
mponentImpl.java:783)
at org.alfresco.repo.security.authentication.ntlm.
NTLMAuthenticationComponentImpl.authenticate(NTLMAuthenticationComponentI
mpl.java:554)
...
```

Then we have a problem with the setup of NTLM or the Windows machine. To figure out what is going on, we should turn on debug logging for the NTLM filter. Open the `log4j.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes` directory and set up debug mode for the following package:

```
log4j.logger.org.alfresco.web.app.servlet.NTLMAuthenticationFilter=debug
```

Restart the server and try logging in again, something like the following logs might appear:

```
14:45:39,432  DEBUG [app.servlet.NTLMAuthenticationFilter] Received type1
[Type1:0xa208b207,Domain:WORKGROUP,Wks:MBERGLJUNG-PC]
14:45:39,434  DEBUG [app.servlet.NTLMAuthenticationFilter] Client domain
WORKGROUP
14:45:39,493  DEBUG [app.servlet.NTLMAuthenticationFilter] Processing
request: /alfresco/wcservice/api/search/keyword/description.xml SID:null
14:45:39,494  DEBUG [app.servlet.NTLMAuthenticationFilter] Found
webscript with no authentication - set NO_AUTH_REQUIRED flag.
14:45:39,495  DEBUG [app.servlet.NTLMAuthenticationFilter] Authentication
not required (filter), chaining ...
```

The NTLM Filter is set up to work with users in one or more domains and we have set the domains as follows:

```
passthru.authentication.domain=win.bestmoney.com
```

This will not really work in this case as the domain is WORKGROUP and the NTLM filter will just ignore this user as he or she is not logged into a Windows domain. We have to make sure that the Windows machine is connected to the correct domain and that the user is logged into the domain.

Using directory servers in a Development Environment

In a lot of cases, Alfresco is run in both the Development Environment and the Test Environment without being hooked up to a directory server until a very later stage. Maybe we are getting within a few weeks of the go-live date before we are configuring Alfresco to integrate with the directory service.

This might be because we do not have access to a directory service during development and testing. However, there are some good reasons for installing our own directory service at an early stage of the project:

- We have a chance to test authentication and synchronization in the way the client wants it done.
- If there are several developers involved in the project then they do not all have to set up the same number of users, groups, and assign members to each group. This is done once in the directory and then everybody can use it.
- If we need to test the module we build with different versions of Alfresco, we can easily hook up each Alfresco installation to the directory.
- We can spot any problems caused by using external authentication on an early stage, instead of having to figure them out a week or two before we are supposed to go live.

Summary

This chapter has really shown us the vast possibilities that exist in the Alfresco platform for authentication with all kinds of different systems and being able to easily import users and groups from different directory servers.

In the first part of this chapter, we looked closer at how different authentication mechanisms such as LDAP and NTLM work and how they are supported by different components in Alfresco.

We then went on to set up and configure Alfresco to authenticate with both OpenLDAP and MS Active Directory at the same time. We saw that Alfresco comes out of the box with authentication subsystem configurations for these two types of directory servers.

After setting up authentication with the directory servers we extended the Alfresco configuration to also synchronize/import user and group data with these directories. When configuring Alfresco to talk to these directory servers, it is helpful to use a tool such as Apache Directory Studio to inspect LDAP directory structure and data.

Sometimes it might not be enough with the out of the box authentication subsystems and we saw how a custom authenticator can be implemented and configured in a separate subsystem.

Finally, we got a lot of tips for how to troubleshoot problems. For example, CIFS requires NTLM authentication to be supported by one of the authenticators, otherwise, the CIFS server will not start. Also, Windows versions newer than 2000 might try and use NTLMv2, which does not work with Alfresco.

In the next chapter, we will dig into CIFS and see how it works and how we can troubleshoot any problems with CIFS. We will also look at WebDAV filesystem access solutions as this protocol is becoming more and more common.

5

File System Access Solutions

The Alfresco repository can be accessed from many different client interfaces and which one of them to use is often dependent on the task at hand. There are the web interfaces Alfresco Explorer and Alfresco Share that have to be used when participating in for example, workflows, collaboration scenarios, and records management, and we have the mobile interface, e-mail interfaces, and so on. However, when it comes to document management most users prefer to work through a filesystem interface on a day-to-day basis.

Alfresco provides an **SMB (Server Message Block)** protocol implementation known as **Alfresco CIFS (Common Internet File System)** that gives the user access to the document repository in the form of a shared drive. CIFS is an open version of SMB with Internet-specific modifications. In the rest of this chapter, we can assume that CIFS and SMB are synonymous.

CIFS gives the user the possibility to work in a familiar Windows Explorer, Mac Finder, or SMB Client environment when accessing documents in Alfresco. This chapter will give you an overview of CIFS and walk through how to configure CIFS on Windows and Linux.

Although this chapter is primarily about CIFS, it might not always be the best solution for larger installations or in situations when a lot of the client computers are connecting remotely, then CIFS might not achieve the performance or stability that is needed. In these cases, the **Alfresco WebDAV (Web-based Distributed Authoring and Versioning)** interface might be a better solution. We will have a look at a couple of different WebDAV clients.

In this chapter, you will learn:

- How CIFS uses NetBIOS over TCP/IP for transport
- How CIFS uses NetBIOS Naming Service for name registration and resolution

- How CIFS can use TCP Directly for transport and DNS for name resolution
- How Alfresco CIFS is installed by default on Windows and Linux
- What the File Servers subsystem is
- Configuring the Alfresco CIFS server on different Windows versions and on Linux
- How to connect to Alfresco through a WebDAV client
- Troubleshooting CIFS connections

File access concepts

Before we start configuring the CIFS server in Alfresco, let's have a look at the underlying protocol and how it works.

CIFS protocol overview

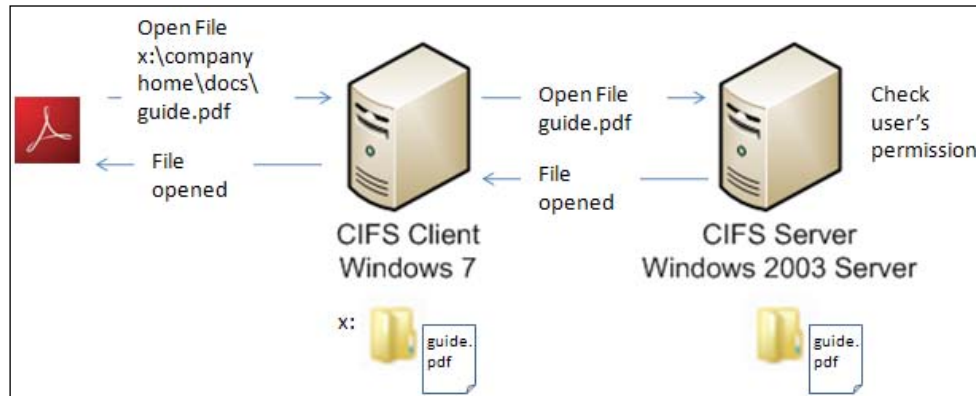
CIFS is a client-server application and protocol that is mainly used to access files and printers in a Microsoft Windows network. CIFS clients can work with files and directories on remote servers as if they were on a local hard-disk.

CIFS Server support comes preinstalled in Microsoft's operating systems, so no need to install any software to support it. CIFS makes it possible to see hard disks and printers on other Windows computers through the Network Neighborhood feature. If a CIFS server is needed on Linux, we would have to install Samba.

Some of the high level features offered by the Common Internet File System are:

- **Access:** CIFS servers are accessible from multiple client operating system platforms including Windows (NT, XP, 2003 Server, Vista, 7, 2008 Server) Macs (OS/X 10.x) Linux, and UNIX.
- **Security and Granularity:** CIFS servers support both anonymous transfers and secure, authenticated access to named files. File and directory security policies are easy to administer.
- **Data Integrity:** CIFS supports the usual set of file operations; open, close, read, write, and seek. CIFS also supports file and record lock and unlocking. CIFS allows multiple clients to access and update the same file while preventing conflicts by providing file sharing and file locking.
- **Optimization for Slow Links:** The CIFS protocol has been tuned to run well over slow-speed dial-up lines.
- **Unicode File Names:** File names can be in any character set, not just character sets designed for English or Western European languages.

CIFS works by sending a request such as open file to the server. The server receives the request, verifies that the client has appropriate permissions, then processes the request and returns a response to the client:



CIFS is a fairly high-level network protocol. In the OSI model, it is probably best described at the Application/Presentation layer. This means CIFS relies on other protocols for transport. The most common protocol used for reliable transport is NetBIOS over TCP (NBT) or TCP directly. Other protocols have been used for the transport layer, however, with the enormous popularity of the Internet, NBT, or Direct TCP has become the de-facto standard.

Although file sharing is CIFS's primary purpose, there are other functions that CIFS is commonly associated with. Most CIFS implementations are also capable of determining other CIFS servers on the network (browsing), printing, and even complicated authentication techniques.

To connect to a CIFS server in Windows we typically use Windows Explorer and through the **Tools** menu we select **Map Network Drive**. We can also map a drive from command line as follows:

```
net use z: \\AlfrescoServerA\Alfresco <password> /USER:<username>
```

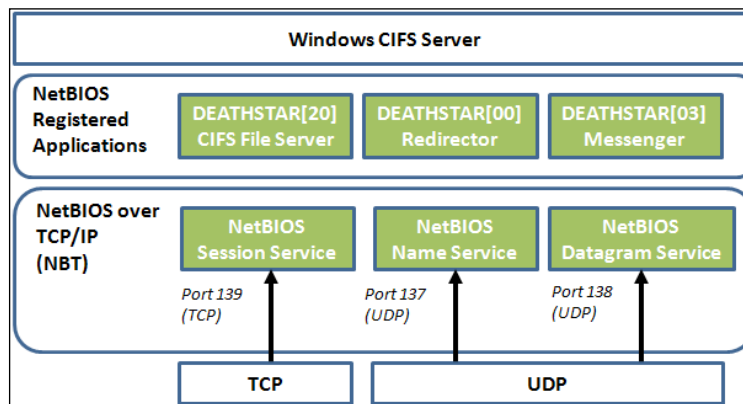
CIFS Transport—NetBIOS over TCP/IP (NBT)

The most commonly used transport protocol for CIFS/SMB is NetBIOS over TCP/IP (NBT). NetBIOS was developed in the 1980s and it defines both a programming interface (API) and a transport protocol. The NBT specifications (RFC1001 and 1002) define three services:

- **Naming service:** Provides a mapping between a name and an address

- Two communication services:
 - **Session service:** Connection-oriented data transmission between NetBIOS computers
 - **Datagram service:** Connectionless data transmission between NetBIOS computers

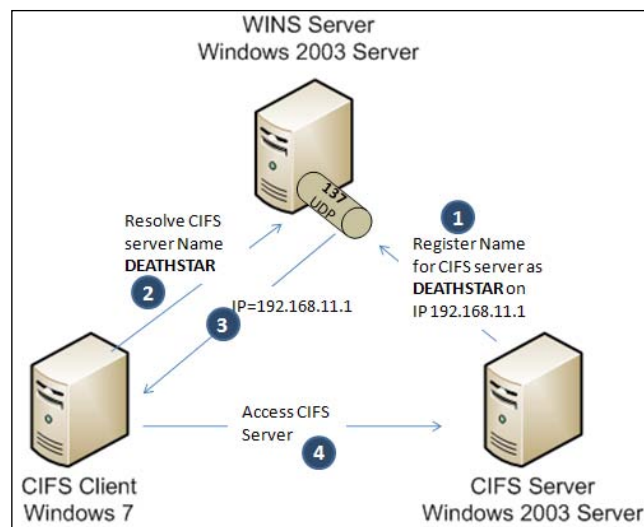
The following figure gives an overview of CIFS and NBT on a Windows box:



Naming service

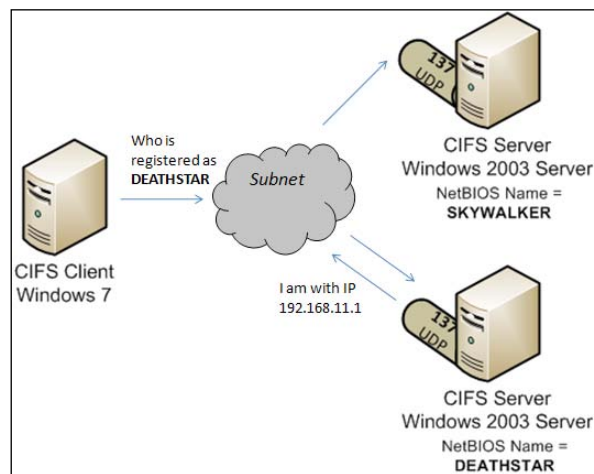
The naming service uses human readable names to identify computers and they can typically be seen in the network neighborhood in a Windows system. These names can be a maximum of 16 characters long (you can only use 15 as the last character is used to specify service type such as file sharing service) and exist in a flat namespace (that is, not dot-separated namespaces as in DNS names). NetBIOS names have the same purpose as domain names used by the DNS system in the TCP/IP environment.

With NetBIOS names end users can refer to specific computers by a name instead of by a transport address, such as an IP. NetBIOS provides a system for mapping a name to an IP address. In a Windows environment, the implementation of the NetBIOS Naming Service is called WINS (Windows Internet Naming Service). The naming service is available on UDP port 137 for registration and resolution of names:



Besides using a NetBIOS naming server to register and resolve names, a broadcasting mechanism can also be used. In this case, the CIFS server broadcasts a registration packet to the subnet via UDP port 137. The CIFS server repeats this three times and the registration is successful, if no other workstation or server on the subnet responds with a message that the name is already taken.

A workstation wishing to resolve a name via broadcast sends a resolve message with the name to the subnet via UDP port 137. This message is sent three times with five seconds between the messages and if the resolve message is successful an IP address will be returned:





Broadcasts aren't meant to cross subnet boundaries, so if we've got computer nodes separated by routers we might have to resort to using WINS.

There are different NetBIOS Node Types depending on if they are using a naming service, broadcast, or both:

- **Broadcast:** Uses broadcast registration and resolution only.
- **Peer-to-Peer:** Uses naming service registration and resolution only.
- **Mixed:** Uses broadcast for registration. If successful, it notifies the naming service of the result. Uses broadcast for resolution; uses the naming service if broadcast is unsuccessful.
- **Hybrid:** Uses the naming service for registration and resolution; uses broadcast if the naming service is unresponsive or inoperative.

It is easy to find out what node type a workstation or server is configured with by using for example, `ipconfig`. On my Windows 7 box I get the following result:

```
C:\Users\mbergljung>ipconfig /all
```

Windows IP Configuration

```
...  
Node Type . . . . . : Broadcast  
...
```

In this case, the Node Type was Broadcast. Trying this on a Windows 2003 Server gives Node Type Hybrid:

```
C:\>ipconfig /all
```

Windows IP Configuration

```
...  
Node Type . . . . . : Hybrid  
...
```

To see what NetBIOS names are registered on a particular NBT computer we can use the Windows command utility `nbtstat`. On my Windows 7 box I get the following names:

```
C:\Users\mbergljung>nbtstat -a mbergljung-PC
```

Local Area Connection:

Node IpAddress: [192.168.0.2] Scope Id: []

NetBIOS Remote Machine Name Table

Name		Type	Status

MBERGLJUNG-PC	<00>	UNIQUE	Registered
WORKGROUP	<00>	GROUP	Registered
MBERGLJUNG-PC	<20>	UNIQUE	Registered
WORKGROUP	<1E>	GROUP	Registered
WORKGROUP	<1D>	UNIQUE	Registered

Here we can see that the MBERGLJUNG-PC NetBIOS name has been registered twice – one for service type 00, which is the Workstation Service (that is machine/computer name registration), and one for service type 20, which is the File Server Service (it means the computer can share files and printers).

A NetBIOS name does not have to be UNIQUE; it can also denote a GROUP, as in this case when the computer belongs to the WORKGROUP.

Communication services

So, now we have got a way to find out the IP address of a NetBIOS service name for a File Server, but how do we start communicating with it? For this we use the NBT Session Service and Datagram Service.

Session service

The NBT Session service is used to establish long lived connections from client computers to file sharing services on a server. The client sends information to the file sharing service about what files it wishes to open, what data it wants to send, and so on. These communications can take place over a very long time, sometimes days, and if there is ever an error there will be retransmissions until data is received successfully.

The NetBIOS Session service has three phases:

- **Session establishment:** It is during this phase that the IP address and TCP port of the called name is determined, and a TCP connection is established with the remote server and service.
- **Steady state:** It is during this phase that NetBIOS data messages are exchanged over the session. Keep-alive packets may also be exchanged if the participating clients and servers are so configured.
- **Session close:** A session is closed whenever either a client or a server (in the session) closes the session or it is determined that one of them has gone down.

The session service functionality takes place over TCP port 139 and is very similar to TCP with a few exceptions. The main difference is that TCP is stream-oriented and does not preserve message boundaries. This is in contrast to the session service, which sends one message at a time over the network and is expected to read one message at a time from the network.

The session service primitives offered by NetBIOS are:

- **Call**—a client opens a session to a remote NetBIOS name on top of a TCP stream
- **Listen**—a server listens for attempts to open a session to a NetBIOS name on top of a TCP stream
- **Hang Up**—closes a session
- **Send**—sends a packet to the computer on the other end of a session
- **Send No Ack**—similar to Send, but doesn't require an acknowledgment
- **Receive**—waits for a packet to arrive from a Send on the other end of a session

The session service makes it possible to handle many NetBIOS names (that is services) on the same TCP/IP connection. We will see later whether a Windows Share and an Alfresco Share can coexist on the same computer.

CIFS uses the session service to send and to receive all upper layer commands, including all file and printer operations. Therefore, the first step in any CIFS network communication is usually to establish a NetBIOS session between the client and server.

Datagram service

The NBT Datagram service is an unreliable, non-sequenced, and connectionless service. The datagram service is used by NetBIOS applications as a fast, broadcast-capable, and low-overhead method of transferring data. The UDP port 138 is used to implement the NetBIOS datagram service within the TCP/IP suite of protocols. UDP is very similar to the NetBIOS datagram service, but still must be slightly modified to emulate all of the datagram service functionality.

All NetBIOS datagram packets that are to be sent over UDP have a header prepended to them. This header contains two important pieces of information—the NetBIOS name of the packet sender, and whether or not the NetBIOS datagram was fragmented to be sent via UDP. With this information, the NetBIOS datagram service can be completely emulated over the UDP protocol.

CIFS implementations typically use the NetBIOS datagram service for browsing. Browsing is the process of discovering the NetBIOS names of CIFS servers that are on the network (Windows then displays this list in the Network Neighborhood).

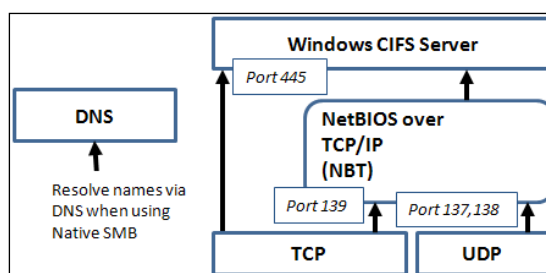


Browsing is not a part of the standard CIFS protocol. Although most CIFS implementations also implement browsing, they are separate entities. Therefore, a pure CIFS implementation would not need to implement the NetBIOS datagram service, only the session and name services described earlier.

For example, the CIFS transport described next—Native SMB does not use NetBIOS at all.

CIFS transport—TCP/IP (Native SMB)

Starting with Windows 2000, Microsoft added the possibility to run SMB/CIFS directly over TCP/IP, without the extra layer of NBT. For this they use TCP port 445 with names resolved via the standard Domain Name Service (DNS). The protocol diagram looks as follows with both Native and NBT transport depicted:



To identify active SMB transport protocols on a Windows system, the `net config rdr` (`rdr` = redirector) and `net config srv` (`srv` = server) commands can be used:

```
C:\Users\mbergljung>net config rdr
```

```
Computer name          \\MBERGLJUNG-PC
Full Computer name     mbergljung-PC
User name              mbergljung
```

Workstation active on

```
NetBT_Tcpip_{969D7610-993F-4D3A-AD39-5DC0A0671FD4} (002170D01F98)
NetBT_Tcpip_{9B01A15C-6B63-4C9F-8F47-EBF898E6FDF0} (00215DCA7076)
NetBT_Tcpip_{0C0862E1-B0AC-4FD6-A43F-2FC8AF956462} (00FF0C0862E1)
```

Here, we can see that my Windows 7 box is using NBT (`NetBT_Tcpip_...`). An instance of `NetBT_Tcpip` is shown for each network adapter that it is bound to. If I run it on a Windows 2003 development server I get this:

```
C:\>net config rdr
```

```
Computer name          \\ALFRESCO2_DEV
Full Computer name     alfresco2_dev.opsera
User name              Administrator
```

Workstation active on

```
NetbiosSmb (000000000000)
NetBT_Tcpip_{D90C6492-0779-4784-9B65-5C8831B03E85} (005056A43270)
```

The `NetbiosSmb` device binding line means that this server also supports the NETBIOS-less protocol where SMB runs directly on top of TCP. `NetbiosSmb` is a global device, and is not bound on a per-adapter basis.



To disable Native SMB on TCP port 445, the File and Printer Sharing service has to be turned off completely. This does not free up port 445, windows still listens on it even though the File and Printer Sharing service is not using it.

To release TCP 445 so other applications can bind to it stop the "Server" Windows service with the description "Supports file, print, and named-pipe sharing over the network for this computer. ..." and restart the computer.

If I issue the server command on my Windows 7 box, I get the following result:

```
C:\Users\mbergljung>net config srv
Server Name                \\MBERGLJUNG-PC
Server Comment

Software version           Windows 7 Ultimate
Server is active on
    NetbiosSmb (MBERGLJUNG-PC)
    NetBT_Tcpip_{9B01A15C-6B63-4C9F-8F47-EBF898E6FDF0} (MBERGLJUNG-
PC)
    NetBT_Tcpip_{0C0862E1-B0AC-4FD6-A43F-2FC8AF956462} (MBERGLJUNG-
PC)
    NetBT_Tcpip_{969D7610-993F-4D3A-AD39-5DC0A0671FD4} (MBERGLJUNG-
PC)
```

So my workstation's CIFS server is listening on native SMB connections on TCP 445. And NBT transport is also bound to all adapters. I get a similar result when doing this on the Windows 2003 server:

```
C:\>net config srv
Server Name                \\ALFRESCO2_DEV
Server Comment

Software version           Microsoft Windows Server 2003 R
Server is active on
    NetbiosSmb (000000000000)
    NetBT_Tcpip_{D90C6492-0779-4784-9B65-5C8831B03E85} (005056a43270)
```

A Windows system with both SMB transports active tries to connect to 445/TCP and 139/TCP at the same time. If the connection to 445/TCP is accepted, the connection to port 139 is closed.

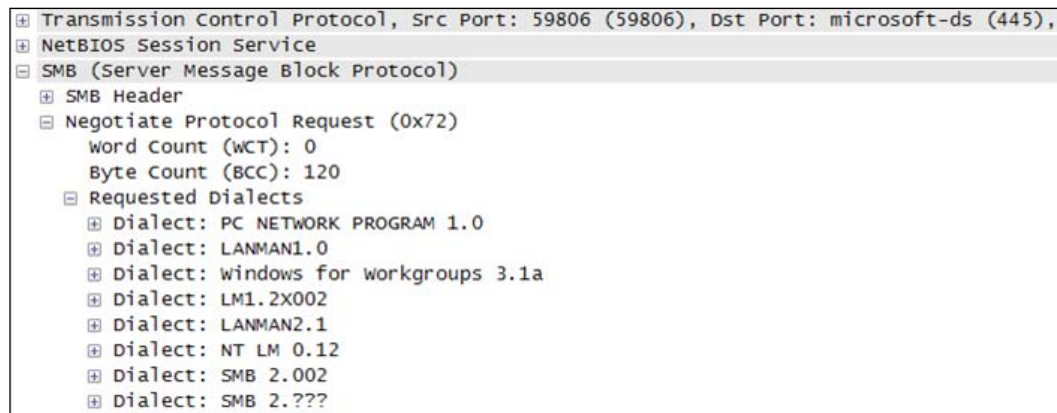
CIFS dialect negotiation

Since the SMB/CIFS protocol's inception, it has been extended from time to time with new commands, creating new versions of the protocol. Each new version is compatible with older versions of the protocol. The following list contains some of the major versions of the SMB/CIFS protocol:

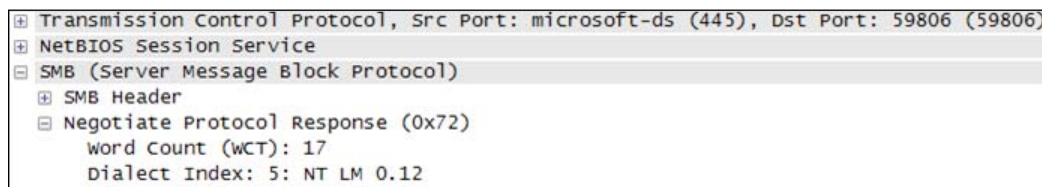
- NT LAN Manager 1.0—used by Windows NT 4.0 (ID = NT LM 0.12)
- Samba's NT LM 0.12—used by Samba (ID = Samba)

- Common Internet File System—used by Windows 2000/XP (ID = CIFS 1.0)
- SMB2—used by Windows Vista and later (ID = SMB 2.0)

The first thing that happens when a CIFS client and a server wants to talk to each other is to negotiate what protocol version to use. On my Windows 7 workstation, a message as follows is sent for a protocol negotiation request, captured with the Wireshark product:



So my workstation is telling the server that it can support a whole lot of SMB dialects and which one would it like to use. The server responds:



So, in this case, the NT LM 0.12 dialect will be used.



A tip when capturing network traffic with the Wireshark product is to use a filter that captures only TCP and SMB traffic between the CIFS client and the CIFS server. A filter like that has the following format: `((ip.src == 192.168.17.83 && ip.dst == 192.168.17.149) || (ip.src == 192.168.17.149 && ip.dst == 192.168.17.83)) && (smb || tcp)`. To also capture NetBIOS name service add `nbns` as protocol.

CIFS authentication and security

The CIFS protocol has two levels of security:

- **User:** A user must authenticate with the CIFS server during the initial connection. The supplied username and password determine what resources the user can access.
- **Share:** This level of security operates on an individual shared resource. The resource has a single password. Anyone with access to the password can access the resource.

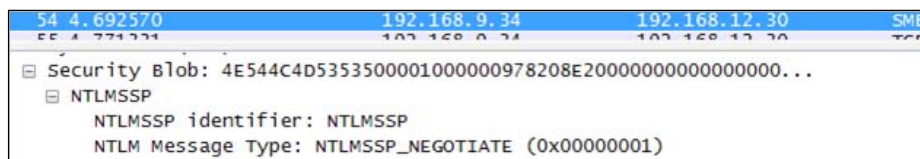
CIFS uses NTLM for authentication and NTLM is a suite of authentication and session security protocols used in various Microsoft network protocol implementations and supported by the **NTLM Security Support Provider (NTLMSSP)**.

The NTLMSSP provides authentication, integrity, and confidentiality services within the Windows **Security Support Provider Interface (SSPI)** framework. NTLM has been largely supplanted by Kerberos as the authentication protocol of choice for Windows domain-based scenarios. However, Kerberos is more complicated and relies on a trusted third party.

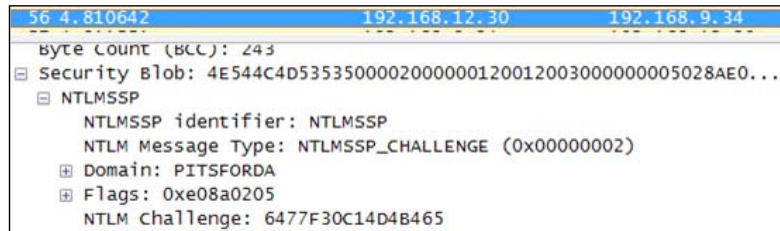
Under both the User and Share security levels, the password is encrypted before it is sent to the server. NTLMv1 (that is MD4) and the older **LAN Manager (LM)** encryption are supported by Microsoft CIFS protocol. However, the newer Windows operating systems such as Vista, 7, and 2008 server will try and authenticate with the newer, more secure NTLMv2 that uses stronger encryption (that is MD5).

The encryption methods use *challenge-response authentication*, where the server sends the client a random string and the client returns a computed response string that proves the client has sufficient credentials for access.

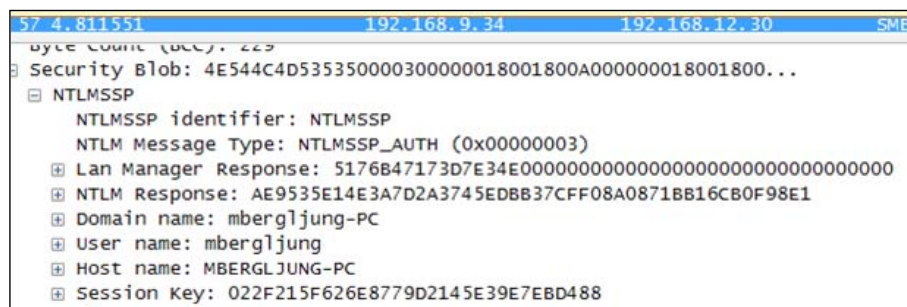
A typical NTLM two stage Session Setup negotiation looks like the following network captures that was taken with the Wireshark product on a Windows 7 box. The client starts by sending a negotiate message:



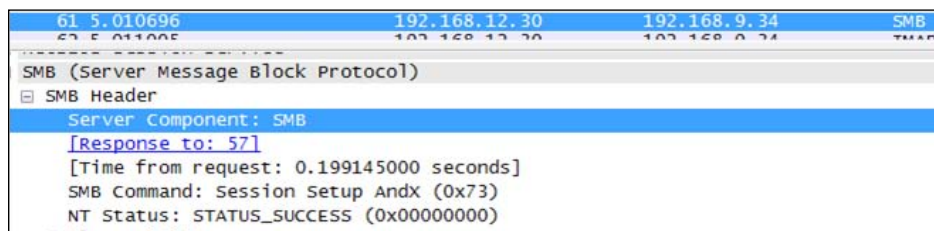
Then the server responds with a challenge message:



And the client sends an authentication message:



And the server responds with a success status message:



Next generation CIFS—SMB2

Microsoft introduced a new version of the Server Message Block (SMB) protocol (SMB 2.0 or SMB2) with Windows Vista in 2006. SMB2 is proprietary but its specification has been published to allow other systems to interoperate with Microsoft operating systems that use the new protocol.

SMB2 reduces the chattiness of the protocol by reducing the number of commands and subcommands from over a hundred to just nineteen. It has mechanisms for pipelining, that is, sending additional requests before the response to a previous request arrives.

The notion of durable file handles is introduced, which allow a connection to an SMB2 server to survive brief network outages, such as with a wireless network, without having to construct a new session. SMB2 also implements support for symbolic links.

SMB2 uses TCP port 445 for communication and does not need NBT for communication. The service is in practice same as the NBT Session Service but without the additional NBT protocol around the SMB Session.

Alfresco CIFS server

Now we know that Windows workstations and servers come preinstalled with CIFS servers and clients for file and printer sharing. So how does the Alfresco server fit into all this? The Alfresco platform comes with its own CIFS server built in. Alfresco acquired the JLAN technology back in 2005 and it is the only available CIFS server implementation in Java.

JLAN provides a number of enterprise features:

- Only pure Java client and server implementation of CIFS, NFS, and FTP.
- High performance which is similar to the native filesystem.
- Enterprise authentication—NTLM, NTLMSSP, SPNEGO, Kerberos.
- Real-time access—no copy to local disk and conflict resolution issues.
- Offline Access—integration to Microsoft Briefcase

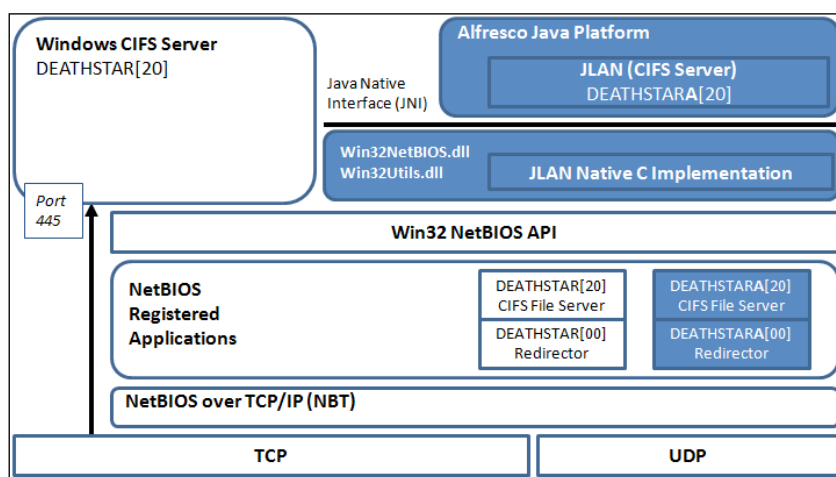
The Alfresco JLAN project supports not only CIFS but also FTP and NFS. This chapter is focusing on the CIFS Server.

The Alfresco CIFS server can be run in the following configurations:

Configuration	Default configuration for	Supported Server OS	Windows File Server can run alongside	Supports Native SMB (port 445)	Supports NBT (port 139)
Java socket-based implementation	Linux, Solaris, Mac	Any	No	Yes, Windows 2000 or later clients	Yes, all Windows clients
Windows Socket based implementation via Win32 NetBIOS API (JNI)	Windows	Windows	Yes	No, Windows Native CIFS Server has it open	Yes, all Windows clients

Alfresco CIFS server on Windows

When the Alfresco CIFS Server is running on a Windows platform it has to coexist with the native Windows CIFS Server for easy installation and use. It does this by using the Win32 NetBIOS API with Windows Sockets instead of Java Sockets:



We can see that there are two new NetBIOS name registrations for the Alfresco CIFS server DEATHSTAR, one for the Computer and one for the File Sharing service. On Windows installations, it is common to use the computer name and append an A to it to compile the NetBIOS name for the Alfresco CIFS service, where A stands for Alfresco.

If we use the Windows utility `nbtstat` on a Windows computer running Alfresco, from a standard installation and configuration, we should see something like this:

```
C:\>nbtstat -a alfresco2_dev
```

Local Area Connection:

Node IpAddress: [192.168.15.22] Scope Id: []

NetBIOS Remote Machine Name Table

Name	Type	Status
ALFRESCO2_DEV <00>	UNIQUE	Registered
ALFRESCO2_DEV <20>	UNIQUE	Registered
ALFRESCO2_DEVA <20>	UNIQUE	Registered
ALFRESCO2_DEVA <00>	UNIQUE	Registered

To verify that the SMB service is listening on the TCP SMB ports, use the following command:

```
C:\>netstat -an | find /i "listening"
...
TCP    0.0.0.0:445          0.0.0.0:0          LISTENING
...
TCP    192.168.15.22:139  0.0.0.0:0          LISTENING
```

Here the Windows CIFS server is listening on port 445 and the Alfresco CIFS server is listening on port 139.

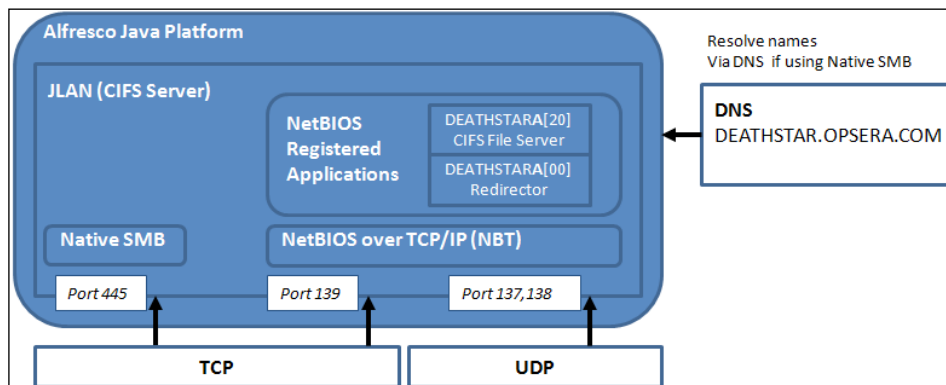
If we are running Alfresco in an external Tomcat (Alfresco usually comes bundled with Tomcat) and want to use CIFS, we also need to copy the `Win32Utils.dll` and `Win32NetBIOS.dll` Alfresco DLLs into the `TOMCAT_HOME/bin`, they can be found in the `<alfrescoinstallldir>\bin` directory.



If Windows 2000 or later clients are trying to connect via TCP 445 they will never reach the Alfresco CIFS server. If we want to use native SMB for the Alfresco CIFS server then the Windows File and Printer sharing service has to be disabled. To do this, we might also have to disable any dependent services such as the Windows Backup service. Make sure to disable the services, and not just stop them and set up manual start, as that will sometimes start the services anyway.

Alfresco CIFS server on Linux

When the Alfresco CIFS Server is running on a Linux/Unix-based platform it does not have to worry that another native CIFS server is already running. Because of this the Java socket-based CIFS server implementation is used by default on Linux:



Here, we can see that Alfresco CIFS Server is responsible for opening Java Sockets and listening to Native SMB or NBT session attempts. This configuration is probably the least error prone as we do not have to worry whether Native SMB (port 445) connection attempts are picked up by the Windows CIFS Server.

The configuration here requires the Alfresco server to be run via the `root` user account as otherwise privileged ports like 445 cannot be used. Normally, we would configure Alfresco to startup with ports 1445, 1139, 1138, and 1137 instead, and configure firewall rules to forward requests, such as for example, TCP port 445 -> 1445.

To check if the CIFS server started correctly and that the Java Sockets are listening to the SMB ports, we can run the following command:

```
alfresco:/home/mbergljung# netstat -anp | egrep -i "(445.*LISTEN) | (139.*LISTEN) "
```

tcp6	0	0	:::1445	:::*
LISTEN			20500/jsvc	
tcp6	0	0	:::1139	:::*
LISTEN			20500/jsvc	

When my Windows 7 workstation connects via CIFS to the Linux server the established connection can be seen as follows:

```
alfresco:/home/mbergljung# netstat -anp | egrep -i "(445.*ESTABLISHED) | (139.*ESTABLISHED) "
```

tcp6	0	0	192.168.12.30:1445	192.168.9.34:58361
ESTABLISHED			20500/jsvc	

Note that a Windows 7 workstation will connect via Native SMB (port 445) so it would not work out of the box against a Windows installation, but works without problem against a Linux installation.

Alfresco CIFS server configuration

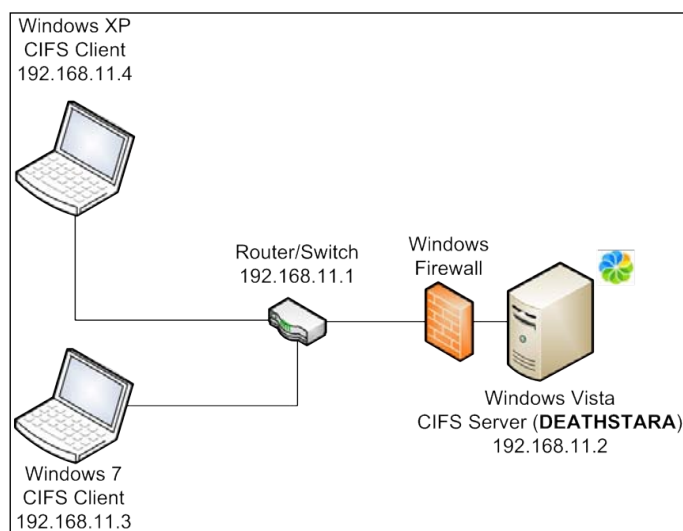
Configuring the Alfresco CIFS Server is normally not a big problem. However, in some setups with firewalls, routers, different clients, and Active Directory authentication it can be a little bit tricky. The following section takes you through how to setup the Alfresco CIFS Server on Windows and Linux in different setups, and it also gives some ideas on how to troubleshoot problems.

Alfresco file server subsystem

As with most other major components in the Alfresco platform, the CIFS server is configured and run as a subsystem. This subsystem is called the File Server subsystem and the default configuration for it can be found in the `file-servers.properties` configuration file located in the `tomcat/webapps/alfresco/WEB-INF/classes/alfresco/subsystems/fileServers/default` directory.

Windows Vista server, Windows 7, and XP clients configuration

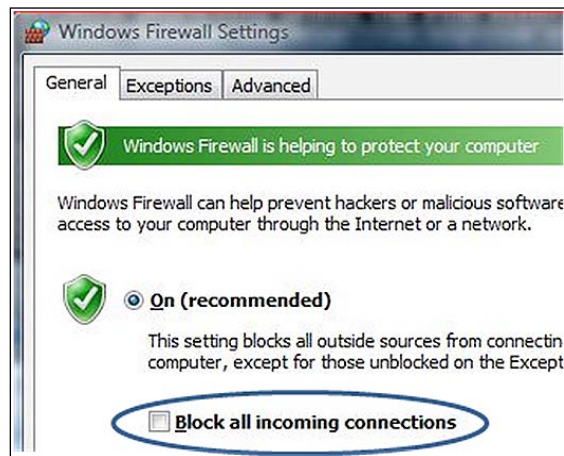
Let's start with a simple configuration where a Windows Vista box runs as the Alfresco CIFS server and a Windows 7 box and a Windows XP box run as the CIFS clients on the same subnet. Here is how that looks:



The default Alfresco configuration is used and the CIFS server will do all authentications against the internal Alfresco database (which supports CIFS authentication). When working with an installation on Windows there are not really that many CIFS configuration properties that we have to deal with, usually none at all.

Now, if we try connecting through the Windows XP CIFS client using the `\\DEATHSTARA\Alfresco` NetBIOS name and folder it will not work. This is because the Windows Vista internal firewall blocks all incoming connection attempts on port 139 and 445. We cannot even connect through browser and Alfresco Explorer.

To solve this, open up the Windows Firewall so it lets through File Sharing traffic, which means it will allow NetBIOS session connection attempts. First, uncheck the **Block all incoming connections** setting, as shown in the following screenshot:



Then make sure **File and Printer Sharing** is allowed as an exception:



You can double check that port 139 and 445 are really reachable from the Windows XP client by using `telnet`, as follows:

```
telnet 192.168.11.2 139
```

If port 139 is open we should get a blank window, if not we should get a message that connection could not be established. To make sure the NetBIOS name DEATHSTARA is available on the server, run the following on the command line:

```
C:\Users\mbergljung>nbtstat -A 192.168.11.2
```

Wireless Network Connection:

```
Node IpAddress: [192.168.11.4] Scope Id: []
```

NetBIOS Remote Machine Name Table

Name	Type	Status

...		
DEATHSTARA	<20> UNIQUE	Registered
DEATHSTARA	<00> UNIQUE	Registered

Then make sure the NetBIOS name is resolvable from the client:

```
C:\Users\mbergljung>ping DEATHSTARA
```

```
Pinging DEATHSTARA [192.168.11.2] with 32 bytes of data:
```

```
Reply from 192.168.11.2: bytes=32 time=162ms TTL=127
```

```
Reply from 192.168.11.2: bytes=32 time=190ms TTL=127
```

So now things are looking up and we can try setting up the CIFS connection again. The Windows XP client can now successfully connect via CIFS to the Windows Vista CIFS Server.

Now try the Windows 7 client and see if this also connects. There is no response when trying to connect. Let's turn on logging and see if we get any clues as to what is going on. Set the following lines to debug in the `log4j.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes` directory:

```
log4j.logger.org.alfresco.smb.protocol=debug
```

```
log4j.logger.org.alfresco.smb.protocol.auth=debug
```

```
log4j.logger.org.alfresco.fileserver=debug
```

And add the following line in `alfresco-global.properties`:

```
cifs.sessionDebug=NEGOTIATE, SOCKET
```

The `NEGOTIATE` parameter will enable logging of CIFS dialect negotiation and the `SOCKET` parameter will enable logging of NetBIOS socket setup.

What we should see now in the `alfresco.log` is the host announcer announcing the NetBIOS name every now and then:

```
19:53:21,933  DEBUG [org.alfresco.fileserver] HostAnnouncer: Announced
host DEATHSTARA
```

```
19:54:42,128  DEBUG [org.alfresco.fileserver] HostAnnouncer: Announced
host DEATHSTARA
```

If I try and login now via CIFS, I still see absolutely nothing happening in the logs. Doing a Wireshark network traffic capture shows that the Windows 7 client is setting up a connection to `microsoft-ds` (that is Direct SMB on port 445):

...

```
Transmission Control Protocol, Src Port: 55379 (55379), Dst Port:
microsoft-ds (445), Seq: 0, Len: 0
```

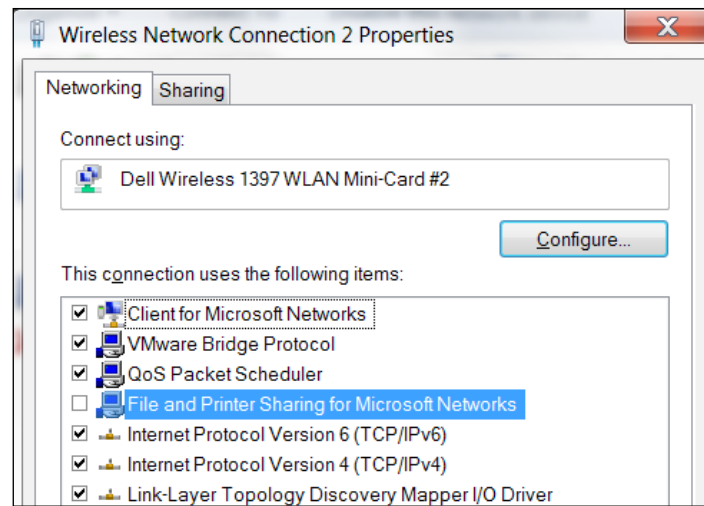
This means that the Windows 7 client will never be able to connect to the Alfresco CIFS server as it does not have control of the Native SMB port 445, which is used by the native Windows File Sharing services.

Looking further in the Wireshark output, we can see that SMB2 is eventually negotiated as the protocol to use (the Alfresco CIFS Server does not support this protocol):

```
41      14.849795      192.168.11.3 192.168.11.2 SMB      Negotiate Protocol
Request
42      14.857771      192.168.11.2 192.168.11.3 SMB2     NegotiateProtocol
Response
```

What we have to do now is either turn off Native SMB in the Windows 7 client or turn off the native File and Printer Sharing in Windows Vista, so the Alfresco CIFS Server can also listen on port 445 for native SMB connection attempts.

It is better to fix this on the server side so we do not have to go around and fix every workstation in Best Money's organization. Open up the Network and Sharing Center and turn off **File and Printer Sharing for Microsoft Network**, as shown in the following screenshot:

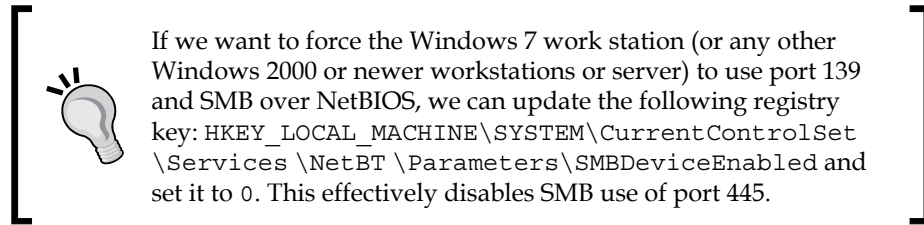


Now it works to connect from the Windows 7 workstation. The logs should show something like the following when XP or W7 workstations log on to the CIFS server:

```
08:49:25,457 DEBUG [org.alfresco.fileserver] [SMB] Winsock NetBIOS
session request received, caller=[VBERGLJUNG-LTXP:WorkStation,Unique,]
08:49:25,535 DEBUG [org.alfresco.fileserver] [SMB] Waiting for Win32
NetBIOS session request (Winsock) ...
08:49:25,535 DEBUG [org.alfresco.fileserver] [WSNB0] Server session
started
08:49:25,535 DEBUG [org.alfresco.fileserver] [WSNB0] Negotiated SMB
dialect - NT LM 0.12
08:49:25,581 DEBUG [org.alfresco.fileserver] [WSNB0] Assigned protocol
handler - org.alfresco.jlan.smb.server.NTProtocolHandler
08:49:40,671 DEBUG [org.alfresco.smb.protocol.auth] NT Session setup
NTLMSSP, MID=8, UID=0, PID=65279
08:49:40,672 DEBUG [org.alfresco.smb.protocol.auth] Using Write
transaction
08:49:41,100 DEBUG [org.alfresco.smb.protocol.auth] Logged on using
NTLMSSP/NTLMv2SessKey
08:49:41,116 DEBUG [org.alfresco.smb.protocol.auth] User mbergljung
logged on (type Normal)
08:49:41,163 DEBUG [org.alfresco.smb.protocol.auth] Using Write
transaction
08:49:41,210 DEBUG [org.alfresco.smb.protocol.auth] Allocated UID=0 for
VC=[0:0,[mbergljung:null,Windows 2002 Service Pack 3 2600,Windows 2002
5.1],Tree=0,Searches=0]
```

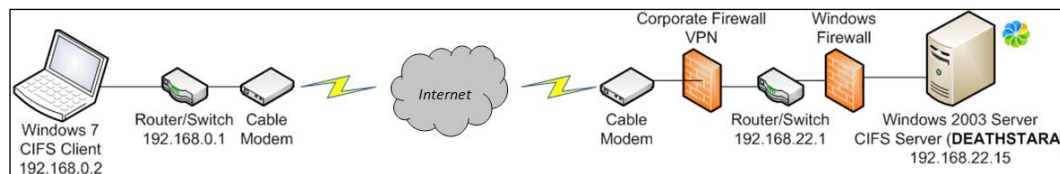

Here, we can see that a Windows Socket Session is set up for workstation VBERGLJUNG-LTXP, which is the Windows XP box.

So far we have seen that most of the problems with getting CIFS to work have to do with network configuration and native File and Printer sharing on Windows. We have not set any property for the CIFS server so far, just used the default settings from the installation.



Windows 2003 Server and Windows 7 client configuration

In this example, the CIFS server runs on a Windows 2003 Server installed on a separate subnet to the CIFS client, which is a Windows 7 workstation running on another subnet. The Alfresco server does all authentications via its internal database:



In this scenario, there are a couple of things we need to fix before we start trying to connect to the CIFS server:

- The first thing we need to do here is to make sure all the necessary ports are open in the company firewall, so it will not stop NetBIOS and Direct SMB traffic. So this means that ports 139 (TCP), 445 (TCP), 137 (UDP), and 138 (UDP) need to be open.
- Then we need to make sure that the Windows 7 workstation can resolve the DEATHSTARA NetBIOS name. We cannot rely on broadcasting the NetBIOS name as a broadcast message will not be allowed to pass through the routers.
- And we also want to make sure there is no Windows Firewall running on the Windows 2003 server that will also stop NetBIOS and Direct SMB traffic (as in the Windows Vista example).

After fixing the firewalls so they are letting through CIFS traffic test it by using telnet from the Windows 7 box as follows:

```
telnet 192.168.22.15 139
```

We should also make sure to test port 445 for direct SMB. If these ports are open in the firewall then we should get a blank window, if not we should get a message that the connection could not be established.

To make sure the NetBIOS name DEATHSTARA is available on the server, run the following on the command line:

```
C:\Users\mbergljung>nbtstat -A 192.168.22.15
```

Wireless Network Connection:

Node IpAddress: [192.168.0.2] Scope Id: []

NetBIOS Remote Machine Name Table

Name	Type	Status

...		
DEATHSTARA	<20> UNIQUE	Registered
DEATHSTARA	<00> UNIQUE	Registered

Then make sure the NetBIOS name is resolvable from the client:

```
C:\Users\mbergljung>ping DEATHSTARA
```

Ping request could not find host DEATHSTARA. Please check the name and try again.

Because there is no broadcasting of the DEATHSTARA NetBIOS name, or the name registered with a WINS server, it will not be possible for the client to resolve it. To get around this we have to hook up the Windows 2003 server to a WINS server and tell Alfresco to use it.



By default the CIFS server will try and auto detect any available WINS servers. The default configuration is setup as follows:

```
cifs.WINS.autoDetectEnabled=true
```

```
cifs.WINS.primary=1.2.3.4
```

```
cifs.WINS.secondary=5.6.7.8
```

If that does not work then hardcode the addresses for the WINS servers and turn off auto detection in `alfresco-global.properties`.

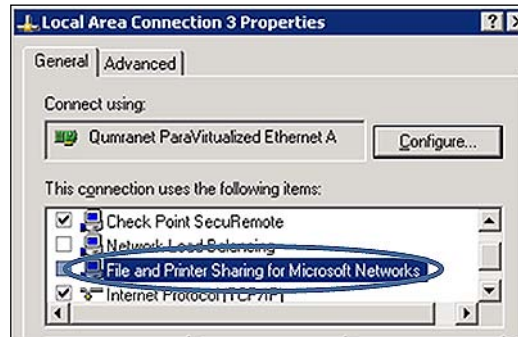
We can also temporarily put the name in the `hosts` file or the `lmhosts` file, which will also be looked at during NetBIOS name resolution. These files are located in the `C:\Windows\System32\drivers\etc` directory.

Add an entry as follows to the `hosts` file:

```
192.168.22.15 DEATHSTARA
```

Now the name resolution will work as expected, so we can move on and get the CIFS access to work. Unfortunately, the Windows 7 box will—as in the first example—try and setup a Direct SMB connection via port 445, which will not work as Windows CIFS server is using it.

So we need to turn off **File and Printer Sharing for Microsoft Networks** on the Windows 2003 Server for this to work.



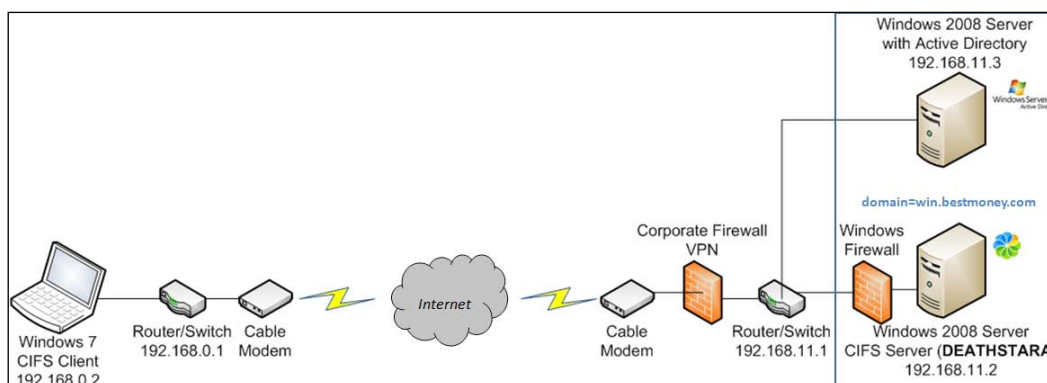
Now when we connect, everything works as expected. However, it would be nice to connect the server to a WINS server, so that naming resolution could be done automatically without the end users having to enter the NetBIOS name -> IP mapping in the local `hosts` or `lmhosts` file.

Also all authentications are done locally with the Alfresco, which is not usually the case. The next example will show how to setup the Alfresco server to authenticate with Active Directory and how to use a WINS server for naming resolution.

Windows 2008 Server, Active Directory, and Windows 7 client configuration

In this example, the CIFS server runs on a Windows 2008 server installed on a separate subnet to the CIFS client, which is a Windows 7 workstation running on another subnet. The Alfresco server does all authentications via a Domain Controller that runs Active directory.

The Domain Controller also works as a WINS server for the NetBIOS name resolution. This is the setup that Best Money is using:



The first thing we are going to do here is to make sure that the authentication is setup correctly. Start by configuring authentication against Active Directory (more information about this is available in the previous chapter) as follows in `alfresco-global.properties`:

```
authentication.chain=alfrescoNtlm1:alfrescoNtlm,bestmoneyAD:ldap-ad,
bestmoneyADPassthru:passthru
ldap.authentication.java.naming.provider.url=ldap://ad.bestmoney.
com:389
ldap.authentication.userNameFormat=%s@win.bestmoney.com
ldap.synchronization.active=false
passthru.authentication.servers=win.bestmoney.com \\ad.bestmoney.
com,ad.bestmoney.com
```

Here, we define an authentication chain that will first try and authenticate against Alfresco's internal database, then against Active Directory via LDAP simple bind, and finally Alfresco will try and authenticate with NTLMv1 against Active Directory, which is known as passthru authentication.

The `bestmoneyADpassthru` authenticator has been configured with what AD server to use when authenticating.



The `bestmoneyADpassthru` authenticator can also be configured to lookup domain controllers for authentication dynamically. Specify the domain with the `passthru.authentication.domain` property. Make sure that you use the Windows NetBIOS domain name, not the forest name. The network broadcast does not work in all network configurations, such as when there are routers between the Alfresco server and the domain controller.

The `passthru.authentication.domain`, `passthru.authentication.servers`, and `passthru.authentication.useLocalServer` properties are mutually exclusive, so no need to specify more than one of them.

Alfresco will only allow one of the built-in authenticators to handle CIFS authentication, so if the first one fails (that is `alfrescoNtlm1`), Alfresco will not let the second one (that is `bestmoneyADPassthru`) try and authenticate. Because of this we turn off CIFS authentication for the `alfrescoNtlm1` authenticator, so Alfresco will go directly to the `bestmoneyADPassthru` authenticator for that:

```
alfresco.authentication.authenticateCIFS=false
```

We also want to be able to use different usernames and passwords to login via Alfresco Explorer. As it stands now, with current configuration, we would be logged in automatically with the Windows credentials when using Alfresco Explorer. Single Sign-on needs to be disabled, so we can try different usernames and passwords:

```
ntlm.authentication.sso.enabled=false
```

Now verify that the AD user that will be used when connecting via CIFS can be used to login via Alfresco Explorer. If that works then we should be able to move on to get the CIFS authentication working (when using Alfresco Explorer, authentication will be done via the `bestmoneyAD` authenticator or the `alfrescoNtlm1` authenticator).

The CIFS server is running in a Windows domain and sometimes it cannot detect the domain by itself, so we need to configure it. Use the following property for that:

```
cifs.domain=win.bestmoney.com
```

We should also configure CIFS logging, so we can see what is going on. See the previous section on how to do that. When the Alfresco server has been started, verify that the following type of log can be seen:

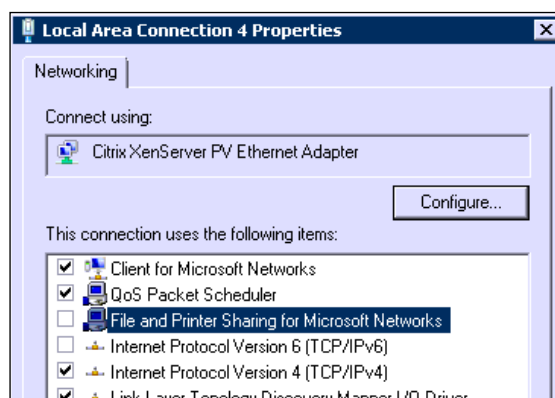
```
09:22:07,346  DEBUG [org.alfresco.fileserver] Passthru server online,  
[win.bestmoney.com\ad.bestmoney.com:192.168.17.2:Online:0,0]
```

This shows that Alfresco has contact with the Passthru authentication server and that it is online and serving domain `win.bestmoney.com`.

Now if we try and setup a CIFS connection to the Alfresco server locally on the Windows 2008 server and use the same username and password that was used when testing login via Alfresco Explorer, it will not work.

This is because the Windows 2008 server will try and use Direct SMB in the same way as Windows 7 does. And there will be nothing in the logs about it. So there is no use in trying to connect from the remote Windows 7 box until we fix this.

The Alfresco CIFS server needs to have control over the port 445 and the Windows File and Printer Sharing needs to be turned off. To do this in Windows 2008, go to **Control Panel | Network and Internet | Network and Sharing Center** and click on the **Local Area Connection** link and then on the **Properties** button:



Uncheck the **File and Printer Sharing for Microsoft Networks** and restart the Windows server and Alfresco. Now try the CIFS connection again locally on the Windows 2008 server. It should work and you should see the log printing something like this:

```
10:36:36,090 DEBUG [org.alfresco.fileserver] [SMB] Winsock NetBIOS
session request received, caller=[DEATHSTARA:WorkStation,Unique,]
10:36:36,277 DEBUG [org.alfresco.fileserver] [SMB] Waiting for Win32
NetBIOS session request (Winsock) ...
10:36:36,277 DEBUG [org.alfresco.fileserver] [WSNB0] Server session
started
10:36:36,277 DEBUG [org.alfresco.fileserver] [WSNB0] Negotiated SMB
dialect - NT LM 0.12
10:36:37,240 DEBUG [org.alfresco.fileserver] [WSNB0] Assigned protocol
handler - org.alfresco.jlan.smb.server.NTProtocolHandler
```

```
10:36:37,240 DEBUG [org.alfresco.smb.protocol.auth] Mapped client null to
domain null
10:36:37,240 DEBUG [org.alfresco.fileserver] Open authenticate session to
[ad.bestmoney.com:192.168.11.3:Online:0,0]
10:36:37,472 DEBUG [org.alfresco.smb.protocol.auth] Passthru sessId=3,
auth ctx=[NTLM,Challenge=5a9bedbfdb955a34]
10:36:37,488 DEBUG [org.alfresco.fileserver] [SMB] NT Session setup from
user=mbergljung, password=d26a4e3cec703df4a1c6f9165b39933f929543220b76489
b, ANSIPwd=d26a4e3cec703df4a1c6f9165b39933f929543220b76489b, domain=WIN.
BESTMONEY.COM, os=Windows Server 2008 R2, VC=0, maxBuf=61440, maxMpx=4,
authCtx=[NTLM,Challenge=5a9bedbfdb955a34]
10:36:37,488 DEBUG [org.alfresco.fileserver] [SMB] MID=8, UID=0,
PID=65279
10:36:37,488 DEBUG [org.alfresco.smb.protocol.auth] Using Write
transaction
10:36:37,566 DEBUG [org.alfresco.smb.protocol.auth] Setting current user
using person mbergljung (username mbergljung)
10:36:37,566 DEBUG [org.alfresco.smb.protocol.auth] Passthru authenticate
user=mbergljung, FULL
10:36:37,566 DEBUG [org.alfresco.fileserver] [SMB] User mbergljung logged
on (type Normal)
10:36:37,581 DEBUG [org.alfresco.fileserver] [SMB] Allocated UID=0 for
VC=[0:0, [mbergljung: [B@1e9db60, WIN.BESTMONEY.COM, Windows Server 2008
R2], Tree=0, Searches=0]
```



If it still does not work that means that the client is trying to use Ntlmv2, which is not supported in a passthru authentication scenario. Ntlmv2 protects against man-in-the-middle attacks, which is just what the Alfresco CIFS server is in this case.

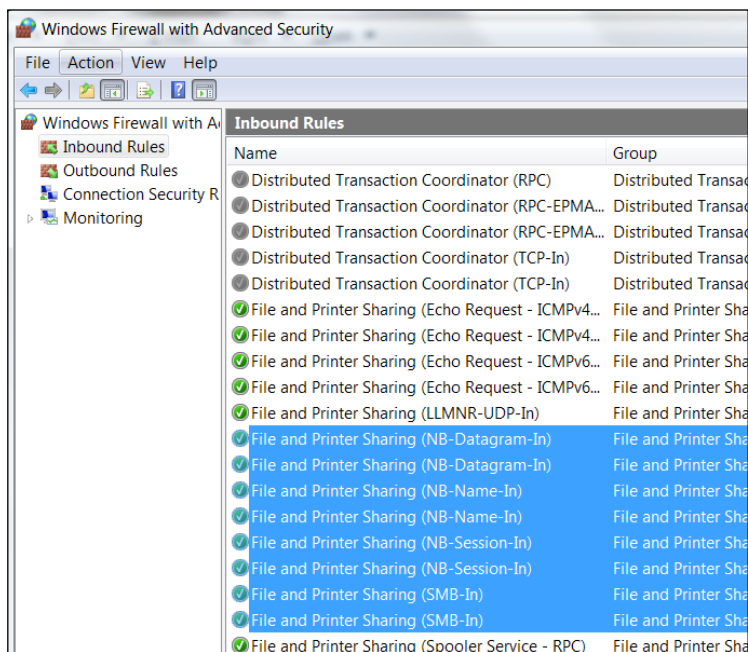
We can get around this by setting the Windows Vista, 7, or 2008 server to use Ntlmv1 by default. Set the following Registry key "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\LMCompatibilityLevel" to "1".

If the key does not exist, create it as a DWORD.

If there are a lot of workstations running Windows Vista and 7, we might be better off using Kerberos instead of NTLM.

We have now got CIFS working locally and authenticating successfully against the Domain Controller with Active Directory. It is time to get this working for remote clients.

As with the other examples, we need to make sure that the NetBIOS and Direct SMB ports are open. In Windows 2008 server, we can open up the NetBIOS ports in the internal Windows Firewall by enabling incoming rules for this:



Test with `telnet` to verify that the server can be reached. Finally, fix so the WINS server is used for NetBIOS name registration and resolution. Alfresco auto-detects available WINS servers on Windows, so no need to set any variables.

To see what WINS servers are available, we can use Windows utility `ipconfig /all` to find out if there is a WINS server:

```
Primary WINS Server . . . . . : 192.168.11.3
```


In some cases, we might see logs like the following when trying to authenticate via CIFS:

```
09:28:00,535 DEBUG [smb.protocol.auth] Mapped client  
null to domain null
```

...



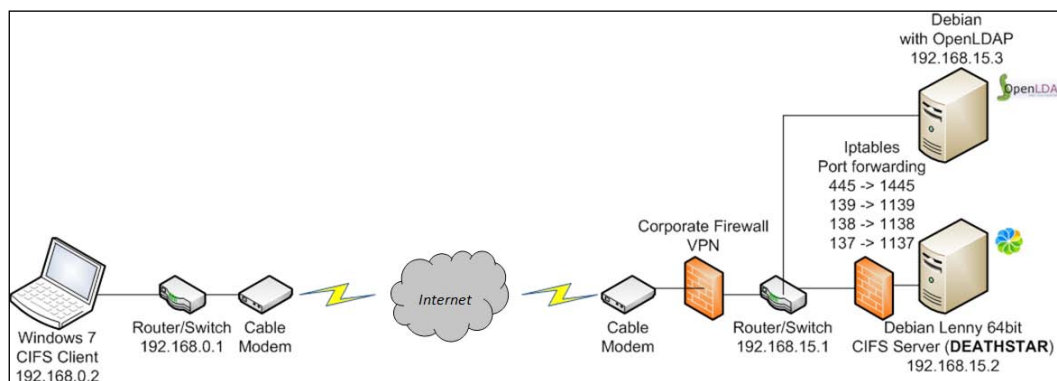
This means that Alfresco was not able to map the user to a domain during login. We can fix this by adding a domain mapping configuration for users not associated with a domain. Add the following properties to the `alfresco-global.properties`:

```
filesystem.domainMappings=WIN  
filesystem.domainMappings.value.WIN.  
subnet=192.168.17.0  
filesystem.domainMappings.value.WIN.  
mask=192.168.17.255
```

Linux server and Windows 7 client configuration

In this example, the CIFS server runs on a Debian Lenny 64-bit Server installed on a separate subnet to the CIFS client, which is a Windows 7 workstation running on another subnet.

The Alfresco server does all authentications via an LDAP server and a custom CIFS LDAP authenticator (see previous chapter for more information about this authenticator):



Normally, an Alfresco Linux installation uses non-privileged ports for CIFS. However, let's make sure that is the case by first finding out what ports the CIFS server is using, non-privileged (for example, 1445) or privileged (for example, 445):

```
alfresco:/home/mbergljung# netstat -anp | egrep -i "(445.*LISTEN) | (139.*LISTEN) "
```

tcp6	0	0	:::1445	:::*
LISTEN			20500/jsvc	
tcp6	0	0	:::1139	:::*
LISTEN			20500/jsvc	

In this case, the Alfresco CIFS server has been configured to use non-privileged ports as follows (This is not the default setting so you would have to put this in `alfresco-global.properties`, if Alfresco is not running as root):

```
cifs.tcpiPSMB.port=1445
cifs.netBIOSSMB.namePort=1137
cifs.netBIOSSMB.datagramPort=1138
cifs.netBIOSSMB.sessionPort=1139
```

So, we need to make sure firewall rules have been setup to forward incoming request on privileged ports to non-privileged ports. Use the `iptables` utility to list current settings:

```
alfresco:/home/mbergljung# iptables --list -t nat
```

Chain PREROUTING (policy ACCEPT)

target	prot	opt	source	destination	
DNAT	tcp	--	anywhere	anywhere	tcp dpt:
netbios-ssn			to:192.168.12.30:1139		
DNAT	tcp	--	anywhere	anywhere	tcp dpt:
microsoft-ds			to:192.168.12.30:1445		

Now, we can test this from the Windows 7 client with `telnet`:

```
C:\Users\mbergljung>telnet alfresco.opsera.com 445
<blank screen>
C:\Users\mbergljung>telnet alfresco.opsera.com 139
<blank screen>
```

If the ports are open all the way, we should get a blank screen as response. If the connection is cancelled, verify with the System Administrator that the ports 137 (UDP), 138 (UDP), 139 (TCP), and 445 (TCP) are open in the company firewall.

Now, we should be able to setup the CIFS connection from the Windows 7 client. This is all there is to it on Linux, which is not that difficult to get to work with CIFS as there is no native CIFS server running. And the CIFS server name is not looked up via NetBIOS name resolving but via a DNS query.



If the Samba file sharing daemon (smbd) is running then that can cause port conflicts with Alfresco's built-in CIFS server. These conflicts may be eliminated by assigning multiple IP addresses and binding the Samba CIFS server and the Alfresco CIFS server to different IPs.

Alfresco WebDAV

We have now covered most things around CIFS and you might think that it should be enough for most deployments and installations. However, CIFS is a very chatty protocol and when we have clients connecting remotely they might experience problems uploading files and browsing through the repository. Also, in installations with thousands of users Alfresco CIFS can experience performance problems and occasional interruptions. Further on, if you are using a virtualized application environment like Citrix XenApp, it can be a configuration challenge to get CIFS working.

The solution in these cases is to turn to WebDAV, which communicates with Alfresco via the familiar HTTP protocol and is much easier to get up and running, and to test that it is working.

Using WebDAV instead of CIFS does have a disadvantage—in that we will have to install client software on each PC, which could be a maintenance/support problem in many organizations, unless we are using Windows XP all around or a virtualized application environment like Citrix XenApp.

We can access Alfresco's WebDAV interface via the `http://localhost:8080/alfresco/webdav` URL. Try it from a browser to see that it works in your installation. You should see something like the following folder listing:

Directory listing for /			
Name	Size	Type	Modified Date
Data Dictionary			Mon, 01 Nov 2010 18:40:16 GMT
Guest Home			Mon, 01 Nov 2010 19:03:10 GMT
User Homes			Mon, 01 Nov 2010 18:40:17 GMT
Sites			Mon, 01 Nov 2010 18:40:27 GMT
Web Projects			Mon, 01 Nov 2010 18:40:40 GMT
Web Deployed			Mon, 01 Nov 2010 18:40:41 GMT

Alfresco WebDAV support is enabled by default and the URL we just used is mapped to a Servlet in the `web.xml` file located in the `/tomcat/webapps/alfresco/WEB-INF` directory. The following entries are relevant:

```
<servlet>
  <servlet-name>WebDAV</servlet-name>
  <servlet-class>org.alfresco.repo.webdav.WebDAVServlet
</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>WebDAV</servlet-name>
  <url-pattern>/webdav/*</url-pattern>
</servlet-mapping>
```

WebDAV authentication is configured with the following Filter configuration:

```
<filter>
<filter-name>WebDAV Authentication Filter</filter-name>
  <filter-class>
    org.alfresco.repo.web.filter.beans.BeanProxyFilter
  </filter-class>
  <init-param>
    <param-name>beanName</param-name>
    <param-value>WebDavAuthenticationFilter</param-value>
  </init-param>
</filter>

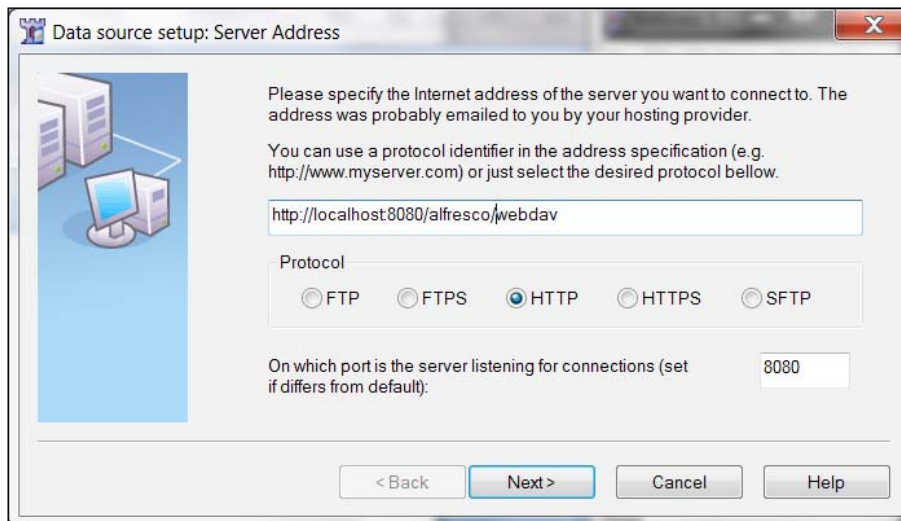
<filter-mapping>
  <filter-name>WebDAV Authentication Filter</filter-name>
  <url-pattern>/webdav/*</url-pattern>
</filter-mapping>
```

When we access Alfresco via WebDAV and a web browser as mentioned earlier, this only gives us read-only access to the Alfresco repository. This is usually not enough and most of the time we want to use a client that enables us to create, update, and delete content. And ultimately, we want to map a drive to an Alfresco resource so we can work with a familiar interface like Windows Explorer.

WebDAV clients

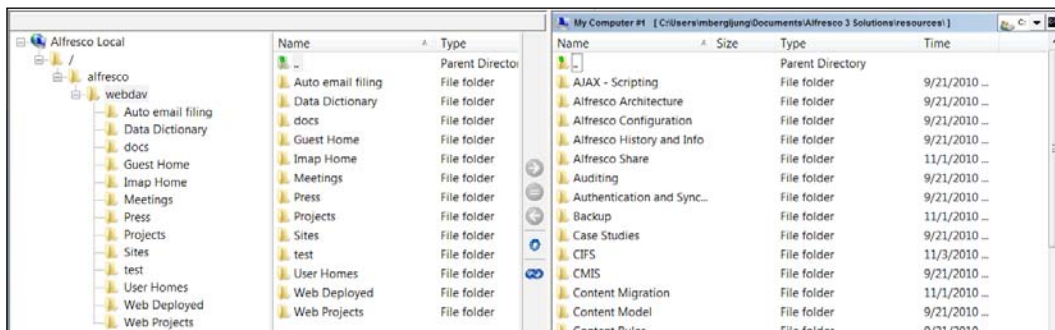
There are several tools out there that we can use on Windows to talk to Alfresco via WebDAV. One such free tool is BitKinex that can be downloaded from <http://www.bitkinex.com/webdavclient>.

During installation specify the address to Alfresco WebDAV, as shown in the following screenshot:



In the next screen, enter the username and password for an Alfresco account. Make sure to specify the `/alfresco/webdav` path as a **Directory (WebDAV)**. If it is setup as a **file** then it will not work to connect.

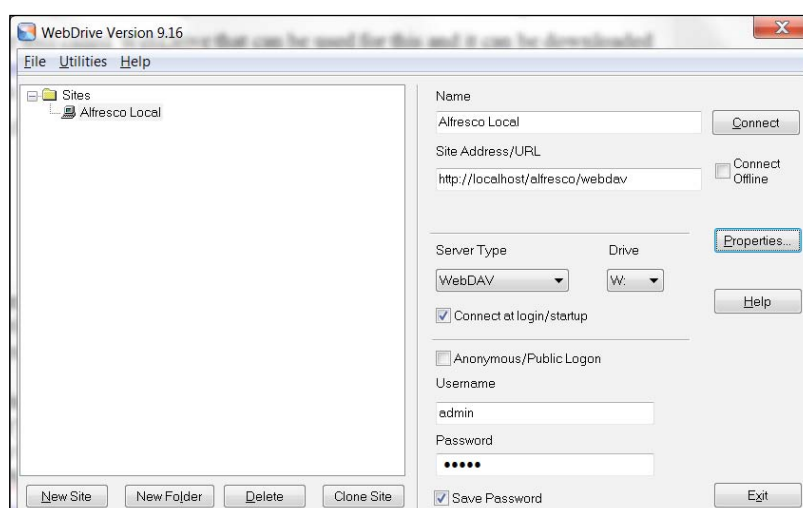
When you are connected you will see a screen with split windows where the left pane shows Alfresco content and the right pane shows local content:



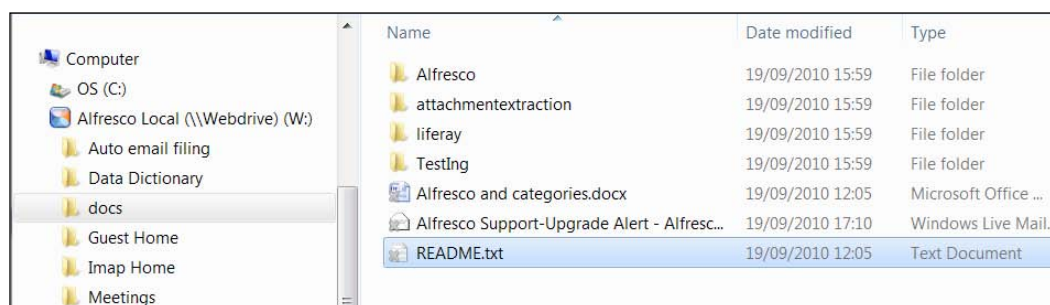
Through this interface we can do most things like browsing the repository, downloading files, and drag-and-drop local files into Alfresco. What we cannot do however is edit a file directly through the interface by double-clicking on it and then update and save it. It would be good if we could map a drive and update files like we do via Windows Explorer.

There is another tool called WebDrive that can be used for this and it can be downloaded from <http://www.webdrive.com/products/webdrive/index.html>. It is a commercial tool but you can download a trial to test it out.

In the **Sites** configuration screen, enter properties for an Alfresco WebDAV connection as follows:



During configuration, we separately specify port 8080 or any other port that we use that is not port 80. If we now **Connect** to this site, we will see a new drive letter (configurable) popup in Windows Explorer (W: in this case):



This WebDrive solution is probably better than the BitKinex solution for end users as it integrates very well with Windows Explorer.

Windows built-in WebDAV clients

Microsoft Windows provides two different WebDAV clients—Web Folders and WebDAV mini Redirector. These clients are integrated into windows and come preinstalled with Windows.

Web Folders (XP only)

Web Folders is the first generation of Windows WebDAV clients and it works only with Windows XP. It allows us to drag-and-drop files between a remote Alfresco WebDAV server and our local computer.

To connect to Alfresco via Web Folders do the following:

1. Go to **My Network Places**, and click on **Add Network Place** at the top of the left-sidebar.
2. The window that pops up is the Add Network Place Wizard. Click **Next**.
3. On the next page, enter the `http://<server:port>/alfresco/webdav` URL of the WebDAV folder in the box named Internet or network address and click **Next**.
4. A window asking for your username and password will pop up at this point. Enter username and password for your Alfresco account, and click **OK**.
5. On the next page, enter a name for this share—this is the name that will show up in the **My Network Places** listing.
6. Click **Finish** on the next page.

WebDAV Mini Redirector (XP, Vista, and Win7)

These are the next generation Windows WebDAV clients that allow us to map a drive to an Alfresco WebDAV location in much the same way we did with CIFS and WebDrive.

WebDAV Mini Redirector limitations:

- No support for HTTPS, that is, no support for secure connections.
- Your WebDAV server must be using port 80, the default port. This makes it impossible to connect directly to Alfresco tomcat, which usually runs on port 8080.

- Can sometimes fail when transferring large files.
- Gets confused if the user does not have access to read and/or write to a file or folder.

A workaround for the port problem could be to install a Web Server like Apache HTTP server in front of Alfresco Tomcat and connect them via the AJP protocol.

This is how you can connect to Alfresco with this client:

1. Right-click on **My Computer** and select **Map Network Drive**.
2. In the Folder "entry field", enter the `http://<server>/alfresco/webdav` URL, and click **Finish**.
3. Enter your Alfresco username and password in the authentication box that appears.

Troubleshooting Alfresco CIFS

The following sections can be used to track down any issues with the CIFS installation.

General

Some general tips.

Nothing happens in Alfresco when trying to log in via CIFS

There are Windows 2000 or later clients that will try and login via Native SMB and that does not work when running Alfresco CIFS server in parallel with Windows CIFS server. Only one server can listen to port 445 at a time and that is the Windows CIFS Server in this case.

Close down Windows CIFS Server to get around the problem. Usually, you have to stop Printer and File Sharing.

Also, make sure there is no firewall blocking the connection, try and telnet to the server and port.

Server says NTLMv2 is not valid for authentication

This can happen if the CIFS client or HTTP client is trying to use NTLMv2 authentication when the Alfresco server is setup for passthru authentication with Microsoft Active Directory.

If, for example, the Alfresco Explorer client is trying to do passthru authentication via NTLMv2, as follows:

```
09:22:48,901 DEBUG [org.alfresco.fileserver] Open authenticate session to
[opsera-w2k8dc01:192.168.17.2:Online:0,0]
09:22:51,729 ERROR [org.alfresco.web.app.servlet.
NTLMAuthenticationFilter] Client MBERGLJUNG using NTLMv2 logon, not valid
with passthru authentication
```

Then this means that the workstation running the browser and Alfresco Explorer is using NTLMv2 by default. Windows Vista, 7, and 2008 server will use Ntlmv2 by default. Passthru authentication does not work with NTLMv2 as it protects against man-in-the-middle attacks, and in this case Alfresco is the man in the middle.

To get around this, we can configure the Windows workstation to use NTLMv1 as follows. Set the following Registry key "HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\LMCompatibilityLevel" to "1". If the key does not exist add it.

When a CIFS client tries to authenticate with NTLMv2 in this scenario, we might not see an NTLMv2 warning log. Instead, the logs might show something like this:

```
17:15:49,463 ERROR [org.alfresco.smb.protocol.auth] org.alfresco.jlan.
smb.SMBException: Invalid parameter
17:15:49,478 DEBUG [org.alfresco.fileserver] [SMB] User opsera_test,
access denied
```

Which actually might mean that the client tried to use NTLMv2, and not that the client specified the wrong password or username.

You might want to consider using Kerberos in a case like this, if there are many Windows Vista and 7 clients.

SMBException: invalid parameter and access denied

See the previous tips.

NetBIOS DLL is not accessible

You will see an error message like the following in the log:

```
09:08:16,358 ERROR [org.alfresco.fileserver] Error accessing Win32
NetBIOS, check DLL is on the path
```

Which means that you need to copy the `Win32NetBIOS.dll` and `Win32Utils.dll` into the `TOMCAT_HOME/bin` directory. These DLLs can be found in the `<alfrescoinstallldir>\bin` directory. [Troubleshooting strategies](#)

Do the following in sequence when you have problems getting CIFS to work. But first, get yourself a network packet capture tool so you can really see what is going on. In this chapter, **Wireshark** was used on Windows. On Linux, use for example, `tcpdump`.

Turning on debug logging for SMB

Turn on logging and see if we get any clues as to what is going on. Set the following lines to debug in the `log4j.properties` file located in the `tomcat/webapps/alfresco/WEB-INF/classes` directory:

```
log4j.logger.org.alfresco.smb.protocol=debug
log4j.logger.org.alfresco.smb.protocol.auth=debug
log4j.logger.org.alfresco.fileserver=debug
```

And add the following line in `alfresco-global.properties`:

```
cifs.sessionDebug=NEGOTIATE, SOCKET
```

The `NEGOTIATE` parameter will enable logging of CIFS dialect negotiation and the `SOCKET` parameter will enable logging of NetBIOS socket setup.

Checking ports from server

On the server side, check that the CIFS ports are active and sockets listening on them:

On Windows do:

```
C:\>netstat -an | find /i "listening"
...
    TCP        0.0.0.0:445           0.0.0.0:0           LISTENING
...
    TCP        192.168.15.22:139    0.0.0.0:0           LISTENING
```

On Linux do:

```
alfresco:/home/mbergljung# netstat -anp | egrep -i "(445.*LISTEN) | (139.*LISTEN) "
```

tcp6	0	0	:::1445	:::*
LISTEN			20500/jsvc	
tcp6	0	0	:::1139	:::*
LISTEN			20500/jsvc	

Checking ports from client

From the client verify that you can reach the 2 TCP ports (the UDP ports are connectionless, so you cannot use `telnet` to verify that they are open):

```
C:\Alfresco3.2rE\bin>telnet alfresco2_dev 445
C:\Alfresco3.2rE\bin>telnet alfresco2_dev 139
```

Checking that CIFS server NetBIOS name is ok

From the client verify that the server has registered NetBIOS name for Alfresco CIFS server:

```
C:\Alfresco3.2rE\bin>nbtstat -a alfresco2_dev
```

Local Area Connection:

Node IpAddress: [192.168.17.61] Scope Id: []

NetBIOS Remote Machine Name Table

Name		Type	Status
ALFRESCO2_DEV	<00>	UNIQUE	Registered
ALFRESCO2_DEV	<20>	UNIQUE	Registered
ALFRESCO2_DEVA	<20>	UNIQUE	Registered
ALFRESCO2_DEVA	<00>	UNIQUE	Registered

Here, we can see that there are two names registered for the File Server Service (that is <20>), the Windows CIFS server (ALFRESCO2_DEV) and the Alfresco CIFS server (ALFRESCO2_DEVA). This will cause problems for clients with Windows 2000 or newer OS. An XP client will connect fine over TCP port 139, but a Windows 7 client will not be able to connect to the Alfresco CIFS server as it will try TCP port 445, which the Windows CIFS server is listening to.

This can be fixed by disabling the Windows File and Printer Sharing and then restarting the computer. Check that nothing is listening on TCP port 445 and then start Alfresco. You should now see the following NetBIOS status:

```
ALFRESCO2_DEV  <00>  UNIQUE      Registered
ALFRESCO2_DEVA <20>  UNIQUE      Registered
ALFRESCO2_DEVA <00>  UNIQUE      Registered
```

On Linux, the DNS entry would have to be checked.

Checking that CIFS server NetBIOS name is resolvable from client

From the client verify that the CIFS server NetBIOS name is resolvable:

```
C:\Users\mbergljung>ping DEATHSTARA
```

```
Pinging DEATHSTARA [192.168.11.2] with 32 bytes of data:
Reply from 192.168.11.2: bytes=32 time=162ms TTL=127
```

On Linux, the DNS entry would have to be checked.

Does any debug logging show up during connection attempts?

If no debug logging shows up during connection attempts then the client is probably a Windows 2000 client or later. They will use Native SMB on port 445, which is controlled by the Windows File and Printer Sharing service. See previous examples in this chapter for how to turn off Windows File and Printer Sharing.

Does the client use the correct authentication method?

Alfresco CIFS server only accepts the NTLMv1 authentication mechanism and any client that tries to authenticate with NTLMv2 will fail. Windows Vista, 7, and 2008 server will use NTLMv2 by default. To downgrade to NTLMv1, do the following registry key update: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\LMCompatibilityLevel to 1. If this property does not exist, create it.

Are you running in a Citrix environment?

If you are running in a Citrix XenApp virtualization environment it might not work to connect to Alfresco CIFS from a virtualized Windows session. Only the first session will work to connect to `\\serverA\Alfresco` and the next session using this address will break the first connection.

To work around this we can use the host file on the Citrix server and configure it as follows with an entry per user (in this case the Alfresco server's IP is 192.168.10.10):

```
192.168.10.10    user1
192.168.10.10    user2
...
192.168.10.10    usern
```

Then each user session maps the CIFS drive with its username such as for example, `\\user1\Alfresco` and so on.

Summary

In this chapter, we have gone through the SMB/CIFS technology quite extensively and seen that to get it to work properly is more of a network issue than an Alfresco configuration issue.

We have learned what NetBIOS is and that it has a Naming Service similar to DNS for registering and resolving computer services, a Session Service for establishing a client-server session between the CIFS client and the CIFS server. NetBIOS is usually transported over TCP/IP and is then called NBT with session setup done on TCP port 139.

CIFS can be used without NetBIOS and is then called Native SMB or Direct SMB with session setup done on TCP port 445.

In a default Windows installation, the Alfresco CIFS server runs alongside Windows CIFS server and only the Windows CIFS server is managing the Direct SMB connections on port 445. This is usually one of the main problems and can be solved by stopping Windows CIFS server.

In a default Linux installation, we do not have any problems with Alfresco CIFS server and it listens to both TCP port 139 (NBT) and TCP port 445 (Direct SMB). However, if the `smbd` is running, we need to stop it as it will interfere with the Alfresco CIFS server.

CIFS is working fine in most situations except in some cases when we have a lot of clients connecting to Alfresco remotely or we are using an application virtualization environment like Citrix. Then we might experience interruptions, problems uploading files, and so on. In these cases, we can instead turn to using the WebDAV interface that Alfresco provides. We saw examples of how to use the BitKinex and the WebDrive WebDAV clients.

We also went through a strategy for how to troubleshoot an Alfresco CIFS server when things are not working as they should. Usually it is all about testing that the Alfresco CIFS server is reachable and is listening to both the 139 and 445 ports.

In the next chapter, we will finally dig into how to design and implement document management (DM) in Alfresco. We will also have a look at records management (RM).

6

Document and Records Management Solutions

In this chapter, we will look at **Document Management (DM)** solutions and introduce the Records Management module. Managing documents is usually the main reason for companies to invest in an ECM solution. Like Best Money, most companies often have a gigantic network drive with hundreds of thousands of documents that are getting more and more difficult to manage.

So what is a DM solution in the context of an ECM system? When we implement a new DM solution we are often asked the question *'Why do we need this, we have the network drive and that works just fine for managing and sharing files?'*. This is a really important and relevant question, and we need to be able answer it with confidence.

Some of the reasons for moving from a network drive solution to a document management solution are:

- **Extended search:** A network drive provides limited file name search whereas a DM solution provides search on filename, **full text search (FTS)**, and property search.
- **Classification:** A DM solution will allow us to associate files with properties that can be used for classification, searching, executing rules, and so on. A network drive does not provide this functionality.
- **Versioning:** A DM solution will allow us to manage versions of a file automatically, so no more `mydoc_v1.doc`, `my_doc_v2.doc`, and so on. A network drive does not have this feature.
- **Auditing:** A document management solution will allow us to keep an audit log of changes to a document, so we can view who changed what and when. A network drive does not have this feature.

- **Applying business rules:** Sometimes we might want to apply business rules in the form of, for example, a script to be run whenever a file is added to a folder, this can be achieved with a DM solution. A network drive does not provide this functionality.
- **Process automation:** By using a DM solution, we can take advantage of the built-in workflow engine and kick off, for example, review processes. This is not available with a network drive.
- **Automatic transformations:** Sometimes it is useful to be able to transform one file format into another when a file is added to a folder. For example, we might want to transform an MS Word document into a PDF. This feature is not available with a network drive.
- **Fine grain permissions:** With a document management solution, we can set up quite sophisticated permission structures. We can control who has the right to create folders, update files, create content, delete content, read content, and we can do it either by role or by individual user. The permission structure can also be integrated with existing directory servers, for example, OpenLDAP and Microsoft Active Directory. This makes it easier to have a central system where you control permissions for groups/roles.
- **3rd party application integration:** Many organizations have more than one user interface where they present information from the ECM system. For example, in the case of Best Money they want to create a custom mobile application for BlackBerry and other smart phones that can display document information. They also want to display some information from the ECM system in their portal.

So, now we can see that we have to think about a lot more things than just files and directories when designing and implementing a DM solution, compared to when setting up a network drive solution.

Document management solutions are also closely related to records management solutions, so we need to think about both of them at the same time during the design phase. Whenever we hear about things such as review periods and a need to save a document for a number of years because of a policy, it's time to have a look at records management and what it can do for us.

In this chapter, you will learn:

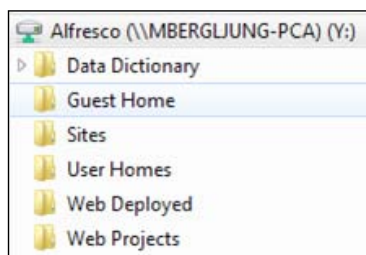
- About the out of the box folder hierarchy
- Using a template when designing folders and space templates
- Setting up a library of rulesets

- Using scripts to create users and groups
- Importing groups based on content in an Excel file
- Implementing simple document review periods
- What Alfresco RM module is?

Out of the box folder hierarchy

Alfresco comes pre-configured with a folder hierarchy and it is important to get familiar with it before designing a domain-specific folder hierarchy for the company or organization that we are working with.

As soon as we have installed Alfresco and viewed the repository from any of the clients, we will see the following top folders:



These top folders have the following meanings:

- **Data Dictionary:** This contains all scripts, metadata model definitions, folder templates, e-mail templates, form definitions, and so on. Most DM solutions have some kind of data dictionary where all definitions and logic that makes the DM solution smarter is stored.
- **Guest Home:** When anonymous access/guest access is enabled, which it is by default, this folder is displayed when a user accesses the system.
- **Sites:** This is the top folder for the Alfresco Share client. The Alfresco Share client is a collaboration environment where users create the so-called 'sites' in which they share information. Every time a new collaboration site is created a new folder is created under this top folder.
- **User Homes:** Each logged in user will have his or her own folder, called home folder, under this folder.

- **Web Deployed:** Web content can be deployed into an application server independently of Alfresco or it can be deployed into an Alfresco instance. If the web content is deployed into an Alfresco instance from an Alfresco WCM authoring environment, then it ends up in this top folder.
- **Web Projects:** Every web project that is managed with the Alfresco WCM module will have its own folder under this top folder.

The Data Dictionary top folder

Now let's look closer at the **Data Dictionary** top folder as we will come in contact with several of its subfolders when building the document management implementation. These subfolders have the following meaning:

- **Email Actions:** This is used by the IMAP subsystem.
- **Email Templates:** This contains templates for e-mails sent out by Alfresco in situations such as when a user or group of users are invited to a folder. These templates are usually written in the FreeMarker language.
- **Imap Configs:** The IMAP interface to Alfresco can display e-mails that wrap document metadata and these e-mails have FreeMarker templates stored in this folder.
- **Messages:** These are custom domain messages for new document management implementations.
- **Models:** These are custom domain models for new document management implementations.
- **Presentation Templates:** Any folder can have a custom view and the templates driving these views are stored in this folder. They are usually FreeMarker templates.
- **Records Management:** When the records management module is installed, it creates this folder that contains custom model, e-mail templates, scripts, and so on for the records management functionality. This deviates a little bit from the convention of how to store things. For example, the e-mail templates should really have been stored under Email Templates and the custom model under Models, and so on, but I guess this makes it easy to find things that have to do with the records management functionality.
- **Rendering Actions Space:** This is a folder used by the system to persist rendering actions.
- **RSS Templates:** Modifications to the document repository can be tracked via RSS feeds. A folder can be configured as an RSS feed and the feed can be configured from an RSS template stored in this directory. This is usually a FreeMarker template.

- **Saved Searches:** Whenever we do a search via the Alfresco Explorer client, we can save the search criteria for later use. This folder contains these saved searches.
- **Scripts:** This contains JavaScripts used by rules or for manual document management via the Run Action feature.
- **Space Templates:** These are folder hierarchy templates used by the document management solution. Folders can be created based on these templates via the Advanced Spaced Wizard.
- **Transfers:** This folder is used by the transfer subsystem.
- **Web Client Extension:** This folder contains customization configurations for the Alfresco Explorer web client. Instead of loading the `web-client-config-custom.xml` file via a bootstrapping procedure, when an AMP is installed, we can just store this file in this folder directly.
- **Web Forms:** These are the web forms used by the Alfresco web content management system.
- **Web Scripts:** The REST-based scripts can be stored in this folder. They can also be loaded via a bootstrapping procedure. The web scripts stored in this folder are usually Alfresco product web scripts, custom web scripts should be stored in the Web Scripts Extension folder.
- **Web Scripts Extensions:** This folder is used to deploy domain-specific custom web scripts used by the DM solution that we implement.
- **Workflow Definitions:** The workflow definitions for the JBOSS JBPM workflow engine can be deployed via this folder. They can also be loaded via a bootstrapping procedure or via the Workflow Admin console.

It is important to know that definitions, templates, and scripts can be added to the dictionary in different ways. We can either add them directly via the UI to the folder above, or via a bootstrap procedure. There are advantages and disadvantages to both ways.

	Advantages	Disadvantages
Adding via UI:	They can test definitions, scripts, and templates directly without restarting the Alfresco server. It can be very timesaving when, for example, a JavaScript or Web Script is being developed or we are designing a FreeMarker template.	It is not so easy to manage all scripts, definitions, and templates when different environments should be updated with the latest dictionary content. It also requires a lot of installation documentation for someone not familiar with the customizations when the solution should be delivered.

	Advantages	Disadvantages
Adding via Bootstrapping:	Easy installation and deployment of definitions, scripts, and templates. All related customizations are contained in one installation file. It is easy to deliver and document.	It takes longer than an AMP installation. It is not ideal during development as it takes time to re-deploy an AMP.

The best solution is probably to use a combination of direct deployment via UI and bootstrapping via AMP installation.

Designing document management solutions

When designing the domain-specific document management features for a company or organization, we need some kind of system or method to do that as there are a lot of things to think about. We will use a special *Document Folder Template* as a tool to design the folder structure.

This folder template includes fields that can be used to specify permissions, rules, versioning, metadata, name, description, and so on for the folder. We will design the folder structure together with the Best Money client. They are after all the only ones who know what should be stored, how it should be stored, how it should be processed, and who should have the right to access it in different ways. Our job will be to advise on how to best construct the metadata, set up permissions, implement rules, and so on.

Defining a proper folder hierarchy for the DM solution is really important for the success of the project. When the folder structure for the Alfresco repository is designed, it will become clear how many of the Alfresco document management features we will need to use. When we design the DM solution, we will also indirectly generate the needed data for the content model.

Another important role for the Document Folder Template is to be the bridge between the Business Analysts that we talk to at the client site and the Software Architect that is going to implement the DM solution. The template should be specified in enough detail, so the Software Architect does not have to consult the Business Analyst very frequently, if at all.

Document Folder Template

The following template will be used to define new folders in the Alfresco repository:

Folder	
Name	Folder name including path from Company Home.
Icon	[folder paper disk]: The icon to be used for the folder. Only relevant in Alfresco Explorer UI.
Description	Description of this folder.
Title	(Optional) Title for this folder.
Permissions	[Group Username (Permission),...]
Inherit Permissions?	[yes no]
Rules	Description of the rules that should be active for this folder.
Apply Rules to Subfolders?	[yes no]
Apply Versioning?	[yes no] Include subfolders? [yes no]
Metadata	Extra metadata for the folder.
Processes	Description of any processes that should be active for this folder.
Stored Documents	
Description	Optional document description.
Metadata	Describe any metadata that can/should be applied to documents in this folder.
Processes	Description of any processes that should be active for documents in this folder.

Some of these fields require a bit more explanation, so we will go through them one by one.

Folder name

Folder names are specified in relation to the /Company Home folder, as in the following examples:

Folder name specification	Explanation
Meetings	Create a folder called <code>Meetings</code> under /Company Home.
Meetings/Congress	Create a folder called <code>Congress</code> under /Company Home/ <code>Meetings</code> .
Press/[year]	Create a folder named after current year and located under the <code>Press</code> folder: /Company Home/ <code>Press/2010</code> .

Folder name specification	Explanation
Affiliates/[Countries*] Countries List: {Sweden, England, Germany, Spain, Italy}	Sometimes when the folder template is used to specify a number of very similar folders we might not want to repeat the same specification over and over again. In these cases we can use patterns or pointers to lists in the folder name specification. This example specification refers to a Countries list that looks like this: {Sweden,England,Germany,Spain,Italy} And the following folders should then be created: /Company Home/Affiliates/Sweden /Company Home/Affiliates/England /Company Home/Affiliates/Germany /Company Home/Affiliates/Spain /Company Home/Affiliates/Italy
Affiliates/[Countries* from affiliate-countries.xls]	If we receive a list of folders in an Excel spreadsheet or some other file, then we could just directly refer to it in the folder specification. These long folder lists will probably be created by a script anyway, and the script would read directly from the Excel spreadsheet.
Affiliates/[Countries*]/ Affiliated	This folder name specification means that we should create an Affiliated folder under each country folder. For example: /Company Home/Affiliates/Sweden/Affiliated
London 2010/[Hotel,Venue, Participants, Documents]/ [Correspondence, - Documents]	In some cases, we can specify the lists inline. Here, we have specified the folder list { Hotel, Venue, Participants, Documents} inline. The specification then says that the Correspondence folder should be created as a subfolder for each folder in the inline list, except for the Documents folder. This means that we should create the following folders in this case: London 2010/Hotel/Correspondence London 2010/Venue/Correspondence London 2010/Participants/Correspondence

Folder title

In some cases, it can be useful to use the `Title` property for a short code or something similar. Let's say we have a situation where each folder name is quite long, but there also exists a short code for the folder, then the short code can be specified in the title property.

This gives the end user the possibility to search with the short code or the longer name. For example:

Name	/Meetings/Meetings for UK 2010
Title	MTUK2010

Folder permissions

Permissions are set up based on the groups and users that exist in the system and what access they should have to the different folders. Best is to follow a **role-based access control (RBAC)** setup, as that is easiest to maintain and integrate with external directory systems such as OpenLDAP and Active Directory.

The following pattern is used to define permissions:

```
[Group| Username (Permission),...]
```

Where an Alfresco Group or Username is used with the permission they should have on the folder. Permission is a combination of C = Create document or folder, R = Read document or folder, U = Update document or folder, and D = Delete document or folder.

Depending on what permission has been set, the following roles are used in Alfresco:

- **C**: Contributor
- **R**: Consumer
- **CR**: Contributor
- **U**: Editor
- **CU** or **CRU**: Collaborator
- **D** in any combination: Coordinator

When creating new group names remember to keep the group name at least three characters long as that is a limitation by default. When setting up permissions for everyone/all make sure to use the built-in **EVERYONE** group. Here are some examples:

Permissions	EVERYONE (CR) SYSTEM_ADMINS (CRUD)
Inherit Permissions?	yes

Permissions	MARITIME (CRU) SYSTEM_ADMINS (CRUD)
Inherit Permissions?	no

Permissions	EDUCATION (CRU) MBERGLJUNG (CRU) SYSTEM_ADMINS (CRUD)
Inherit Permissions?	Yes

Make sure to talk to System Administrators at an early stage, so they are aware of the groups that you intend to use when setting up the Alfresco permission structure. Chances are that there are already groups and a permission structure in use on a company wide scale that can be used with not too many changes when we define the Alfresco access control.

And the System Administrators do not usually want to maintain different groups for almost the same thing. So, we do not want to have 30 groups defined and specified all over the folder hierarchy and then have the System Administrators telling us that we have to change the name of all groups or use a different number of groups, and so on.

Rules

Each folder that is specified can have business rules associated with it. When we discuss the document management functionality with the client they will often talk about business rules without specifically referring to the functionality as business rules. It is our job to extract this functionality as business rules and specify it in the Document Folder Template. Here are some examples of business rule specifications:

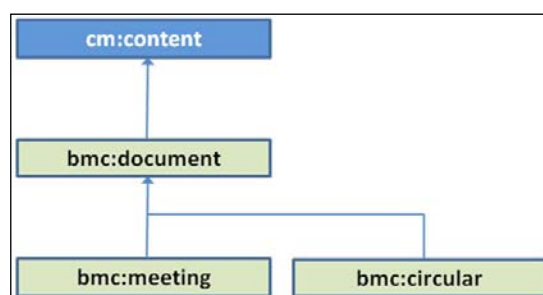
Rules	1. Check Meeting Document Naming convention. 2. Apply Meeting Document type (metadata). 3. Extract Meeting Metadata from filename.
Apply Rules to Subspaces?	1. Yes 2. Yes 3. Yes

Rules	1. Convert MS Word 2003 and 2007 documents into PDFs and save PDFs in the Published folder.
Apply Rules to Subspaces?	1. No

Metadata

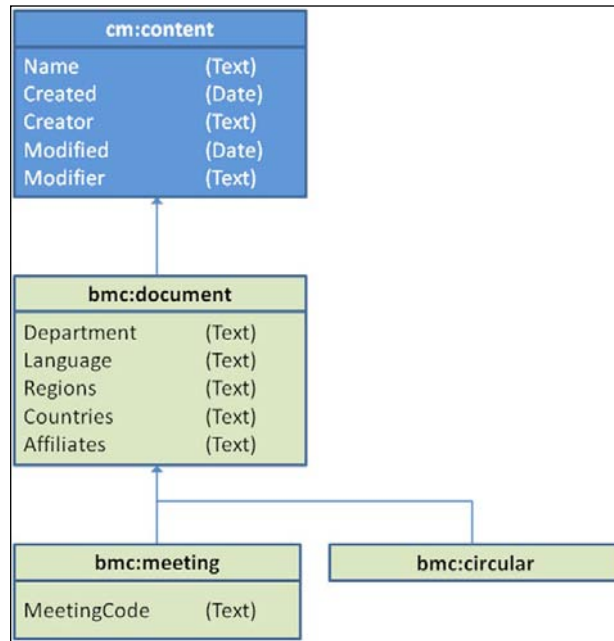
When each folder is specified we need to think about what kind of documents are going to be stored in the folder and how they are going to be classified. One way of doing this is to ask the Business Analyst what data they would like to be able to search on and extract the classification information from that.

It can be helpful to think about the classification in two levels. The first high level classification is just a document type classification. So we would, for example, ask the Best Money Business Analyst what types of documents are going to be stored in the folder and they might say it is going to be meeting documents. So, we got our first type *Meeting*, and so on. We should always have one base type that all other domain-specific types inherit from, as in the following example:



In this example, the **bmc:document** is the base type (for Best Money and *bmc* is the namespace) where we can add generic properties that are common for all other types. The base type extends the Alfresco base content type **cm:content**.

When we have got the type structure sorted, we can continue and talk about what properties should be part of each type. Then we end up with a diagram looking something like this:



The properties need to be specified in more detail to make implementation easier. The following template will be used for that:

Name	Description	Data type	Constraint	Mandatory	Multi	Example	Advanced Search

And here is a property example definition:

Name	Description	Data type	Constraint	Mandatory	Multi	Example	Advanced Search
Department	The Best Money department that created the document.	Text	See department codes in BestMoneyDoc Codes.doc, plus ""	False	No	HR	Yes

The fields have the following meaning:

- **Name:** This is the name of the property (`<property name=>`).
- **Description:** This is the description of the property, so the Software Architect knows what it is used for and how it fits into the whole picture. (`<title>`).
- **Data Type:** This is the data type that should be used when specifying this property in the Alfresco custom content model (`<type>`).
- **Constraint:** This is an optional specification of a constraint for this property. Here we can refer to an external document with lists and codes. Add any other constraint that should be added to the list such as an "" empty selection. (`<constraints>`).
- **Mandatory:** This property is mandatory and should appear with a star in the UI (`<mandatory>`).
- **Multi:** It should be possible to select more than one value for this property (`<multiple>`).
- **Example:** This is an example of a value for this property.
- **Advanced Search:** This property should be available in the Advanced Search form in Alfresco Explorer.

Document versioning

To specify that versioning should be enabled for documents in the folder we use the **Apply Versioning** field. If document versioning should be applied to all subfolders then we also specify yes for the **Include Subfolders** field.

Versioning is also implemented via rules but it is a common enough feature to have its own field. It is also easier to remember to think about versioning during the document management design phase when it has its own field.

Processes

When a document is uploaded to the folder there is sometimes a business process associated with it, such as a review and approval process. This process should be described in this field. The best way to describe the business process is with a Swim Lane diagram, unless it only consists of a couple of steps.

For more information about business process design with Swim Lane diagrams see *Chapter 9, Business Process Design Solutions*.

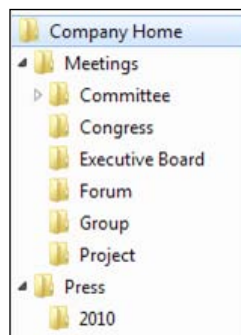
Designing the Best Money document management solution

We now have some useful templates and ideas on how to go about designing document management solutions. So let's take Best Money as an example and define part of their folder hierarchy.

Meetings and Press folder hierarchy

We will define the **Press** and **Meetings** folder hierarchy and the associated Meeting space template. Defining the complete folder hierarchy for the Best Money client is out of scope for this chapter. It would be far too long a chapter and once we get the hang of it, it is quite the same job whether we define 20 top folders or just a couple as in this case.

After talking to Best Money's Business Analyst, we have found out that the **Press** and **Meetings** folder structure should look something like the following screenshot:



The Meetings folder hierarchy

So let's start by defining the Meetings top folder as follows:

Folder	
Name	Meetings
Icon	Folder
Description	All arrangements and documentation around meetings, conferences, committees, forums, projects, and seminars, and so on.
Title	--

Folder			
Permissions	EVERYONE (CR)		
	SYSTEM_ADMINS (CRUD)		
Inherit Permissions?	yes		
Rules	1. Apply standard Best Money Meeting Document metadata via Form, nothing is mandatory.		
Apply Rules to Subfolder?	1. Yes		
Apply Versioning?	Yes	Include subfolders?	Yes
Metadata	--		
Processes	--		
Stored Documents			
Description	--		
Metadata	Meeting metadata:		
	Department (constraint by department list)		
	Language (constraint by language list)		
	Country (constraint by country list)		
	Meeting Code		
Processes	Keep permanently.		

Here we have specified a top folder called `Meetings` that should have the folder icon (that is, default icon). We have set up permissions so everyone can access the folder, create subfolders, and upload documents into this folder. The System Administrators can do anything but they are not the same administrators as the Alfresco administrators, so a new group called `SYSTEM_ADMINS` needs to be created. The folder inherits permissions from the `Company Home` folder, so everyone with access to Alfresco will be able to read the `Meetings` folder.

When something is uploaded into the `Meetings` folder, we have specified a rule that will apply a new type to the document. This type should be called `Meeting` and have properties such as `Department`, `Language`, `Country`, and `Meeting Code`. This rule should also be active for any subfolders.

We also want any meeting document to be versioned, so we can go back and see previous versions and comments around updates. The documents in this folder or subfolders should be kept permanently, so there are no rules or processes regarding review periods, retention policies, and so on.

Next follow a couple of subfolders for different kinds of meetings:

Folder			
Name	Meetings/Committee		
Icon	Paper		
Description	All arrangements and documentation around committee meetings.		
Title	--		
Permissions	SECTION_COMMITTEE (CRU) STEERING_COMMITTEE (CRU) EXECUTIVE_COMMITTEE (CRU)		
Inherit Permissions?	yes		
Rules	Inherited		
Apply Rules to Subfolder?	--		
Apply Versioning?	Yes – inherited	Include subfolders?	--
Metadata	--		
Processes	--		
Stored Documents			
Description	Minutes, Agendas, and Presentations.		
Metadata	Inherited		
Processes	--		

The Committee meetings subfolder should contain all documentation for committee meetings and it should have the paper icon with a pen. The permissions from the Meetings top folder are inherited, so we only need to add permissions for the committee members that should have the right to do anything except delete documents. Three groups SECTION_COMMITTEE, STERING_COMMITTEE, and EXECUTIVE_COMMITTEE have been set up with the Collaborators role (that is, CRU). We have to make sure they are synchronized properly from the directory system that Best Money has.

Rules regarding versioning and metadata are inherited from the top folder, so not much else to specify for this subfolder.

Folder	
Name	Meetings/Executive Board
Icon	Paper
Description	All arrangements and documentation around executive board meetings.

Folder			
Title	--		
Permissions	EXECUTIVE_BOARD (CRU)		
	SYSTEM_ADMINS (CRUD)		
Inherit Permissions?	False		
Rules	Inherited		
Apply Rules to Subfolders?	--		
Apply Versioning?	Yes — inherited	Include subfolders?	--
Metadata	--		
Processes	--		
Stored Documents			
Description	--		
Metadata	Inherited		
Processes	--		

The `Executive Board` subfolder is a little bit different from the other subfolders as it does not inherit permissions from the top `Meetings` folder. Because of this, we have to redefine the `SYSTEM_ADMINS` permission. Rules are inherited as for the other subfolder, so metadata and versioning will be set.

The other `Meetings` subfolders such as `Congress`, `Forum`, `Project`, and `Group` are very similar to either the `Committee` or `Executive Board` folder specifications, so we are not going to specify them here, but they would be done in a similar way as the other two subfolders.

When we are done with the folder specifications, we should also define the metadata in detail with the special template discussed before. For the Meeting metadata type it would look like this:

Name	Description	Data type	Constraint	Mandatory	Multi	Example	Advanced Search
Department	The Best Money department that created the document.	Text	See department codes in BestMoneyDoc Codes.doc, plus ""	False	No	HR	Yes
Language	Language that the document is written in.	Text	Languages Plus ""	False	No	En	Yes

Name	Description	Data type	Constraint	Mandatory	Multi	Example	Advanced Search
Country	The country/ countries relevant to the document.	Text	Countries plus ""	False	Yes	UK, France	Yes
MeetingCode	The unique code for this meeting.	Text		False	No		Yes

The Press folder hierarchy

The Press top folder specification looks like this:

Folder	
Name	Press
Icon	Folder
Description	All press releases
Title	--
Permissions	PRESSTEAM (CRU) SYSTEM_ADMINS (CRUD)
Inherit Permissions?	yes
Rules	1. Apply standard Best Money Document metadata via Form, nothing is mandatory.
Apply Rules to Subfolders?	1. Yes
Apply Versioning?	Yes
Include subfolders?	Yes
Metadata	Reviewable: Review Period in Years (Set review period to five years) Include Subfolders Flag (Set to true for this folder)
Processes	At the end of each month, scan all folders that are reviewable and check last modified date against review period. If the review is up then send an e-mail to the new group called DOC_REVIEWERS. Finally, set the modified date to today's date so the review is not repeated next month.

Folder	
Stored Documents	
Description	--
Metadata	Press release metadata: Department (constraint by department list) Language (constraint by language list) Country (constraint by country list)
Processes	--

The Press top folder inherits permissions from parent folder /Company Home, so all users will be able to read this folder. A new group is also defined called PRESSTEAM with members that have the rights to upload and create content in this folder.

A rule is defined to apply the general document metadata department, language, and country to any document uploaded or created in this folder. Documents in this folder should also be reviewed after a certain number of years so we describe a review process and a new reviewable type (this type could be applied to any folder in the future).

All press releases for a year are saved in a separate folder defined like this:

Folder			
Name	Press/[year]		
Icon	Disc		
Description	All press releases for [year]		
Title	--		
Permissions	Inherited		
Inherit Permissions?	yes		
Rules	Inherited		
Apply Rules to Subspaces?	--		
Apply Versioning?	Yes – inherited	Include subfolders?	--
Metadata	Inherited		
Processes	Inherited		
Stored Documents			
Description	--		
Metadata	Inherited		
Processes	--		

This folder inherits pretty much all settings from the Press top folder except the folder icon, which should be a disk.

Meeting folder/space hierarchy template

When it is time to upload documentation for a meeting for the first time, the user has to decide into which of the Meeting subfolders (that is, Committee, Congress, and so on) the documentation should go. Then a new Meeting folder structure needs to be created. It is a good idea to base the Meeting folder structure on a template as then users will recognize it from one meeting to another.

Folder Templates, or Space Templates as they are called in Alfresco, are defined in the same way as folder hierarchies, using the same template with some minor additions. The **Meeting** folder hierarchy looks like this:



And in this case, the meeting name is **Finance 2010-May London**. Here we have quite a lot of subfolders, so we want to use as many short codes as possible when defining each folder specification so we do not end up with too many specifications.

For the Meeting space template we start by defining the top folder for a meeting:

Folder Template	Meeting
Applied to Folder Name	Meetings/{Meeting Name} /[Hotel,Venue,Participants, Documents]/[Correspondence, - Documents]
Icon	Paper
Description	Documents relating to hotel bookings, venue, participants for Best Money meetings.
Title	--
Permissions	--
Inherit Permissions?	True
Rules	Inherited + 1. Check Naming Convention, abort upload if not correct 2. Parse and Set Language and Department Metadata from Filename Example: 10En-FM.02_3_annex1.doc • 10 = last 2 digits of the year • En = language • FM = department code • 02 = sequence number • 3 = Agenda item • annex1 = annex for that agenda item RegExp: <pre>^\d{2}(En Fr Ge Sp Sw Ru Jp Po Ar Ch)- (A HR FM FS FU IT M L)\.\d{2}_\d{1,3}_annex.*</pre> A = Audit, HR = Human Resources, FM = Financial Markets, FS = Financial Services, FU = Funds Management, IT = Information Technology, M = Marketing, L = Legal
Apply Rules to Subfolders?	1. Yes 2. Yes
Apply Versioning?	Yes – inherited
Include subfolders?	--
Metadata	--
Processes	--
Stored Documents	
Description	--
Metadata	Inherited
Processes	--

The template that we use to specify a folder that is part of a space template is almost exactly the same as the one used for a normal folder specification. The only difference is one extra field at the beginning called **Applied To Folder**. This field specifies what parent folder the folder specification should be applied to.

The name of the parent folder is unknown and therefore specified as {Meeting Name}. It is specified by the user when a new folder hierarchy is created based on the space template that we are defining.

The folder name is specified as / [Hotel, Venue, Participants, Documents] / [Correspondence, -Documents] and this means that this specification is the same for all these folders. We should create the Hotel, Venue, Participant, and Documents subfolders and for each of these subfolders a Correspondence subfolder should be created, except for the Documents folder. This saves us from creating a lot of redundant folder specifications, as they are the same in this case.

There are no new permissions specified and they are instead inherited from the parent folders. A new rule has been added to check the naming convention of all documents uploaded to this folder and its subfolders. If the document has an incorrect filename then it should not be saved and an error message should be displayed.

It is important to go through an example of the naming convention rule so it is completely clear how it should work and be applied. Nothing should be ambiguous when the Software Architect is about to implement the rule. That is also why we try and specify what regular expression could be used to enforce the rule. The more we can do upfront, the more communication we save later on.

Next, we specify the subfolder for the Participants folder as follows:

Folder Template	Meeting
Applied to Folder	Meetings/{Meeting Name}/Participants
Name	/[Lists,Interpreters]
Icon	Paper
Description	Lists = Lists of participants attending meetings attended or hosted by the Best Money. Interpreters = Documents relating to interpreters at meetings attended or hosted by the Best Money.
Title	--
Permissions	--
Inherit Permissions?	True
Rules	Inherited

Folder Template		Meeting	
Apply Rules to Subfolders?	--		
Apply Versioning?	Yes – inherited	Include subfolders?	--
Metadata	--		
Processes	--		
Stored Documents			
Description	--		
Metadata	Inherited		
Processes	-		

And finally we define the subfolders for the Documents folder:

Folder Template		Meeting	
Applied to Folder	Meetings/{Meeting Name}/Documents		
Name	/[Agenda,Briefing,Minutes,Reports]		
Icon	Paper		
Description	Documents relating to meetings.		
Title	--		
Permissions	--		
Inherit Permissions?	True		
Rules	Inherited		
Apply Rules to Subfolders?	--		
Apply Versioning?	Yes – inherited	Include subfolders?	--
Metadata	-		
Processes	-		
Stored Documents			
Description	--		
Metadata	Inherited		
Processes	--		

Defining folder specifications is something we, as Software Architects, can do together with the client's Business Analyst, or they can do most of it themselves after some training in how to use the templates.

Usually, we would create a separate document for the folder specifications and folder template specifications. This is the document that the Software Architect will use as a base for the implementation.

Implementing the Best Money document management solutions

When the folder specification has been completed by the Business Analyst and the Software Architect, it is time to implement the DM solution. It is usually a good idea to start by setting up the groups as they are going to be needed when setting up the folder hierarchy. We are also going to need some test users belonging to the different groups, so we can test the permission settings.

Setting up users and groups

In the beginning of the project, it is highly unlikely that we will be able to connect to Best Money's Active Directory where the groups and users are managed. So we need a temporary solution in the development and testing environment before we have everything hooked up to the directory and the groups and users in sync with the directory.

When we start the implementation, we usually just want to create the groups and users locally in our development Alfresco instance. If there are only a few groups then we can just add them manually via the Alfresco Explorer User Interface but otherwise it might be good to use a JavaScript for that.



Setting up the users and groups manually or via scripts is a temporary solution, so it is really important that everyone has approved the folder specification and the groups in it. So the System Administrator does not come along when the implementation is almost done and point out that the groups you guys have used do not at all match the groups we use in the directory, you will have to update.

Using a script to set up users and groups

Sometimes it is useful to be able to automatically populate the repository with users and groups for testing or training purposes. In this case, a script can come in handy, so we do not have to do this manually every time we want to set up a new Alfresco instance with the Best Money environment.

The following JavaScript shows how to create all the Best Money groups that we have used in the folder specification and also how to create 20 test users that are members of one or more of the groups:

```

logger.log("Start creating users and groups");

var bestMoneyParentGroupName = "BEST_MONEY";
var sysAdminsGroupName = "SYSTEM_ADMINS";
var sectionCommitteeGroupName = "SECTION_COMMITTEE";
var steeringCommitteeGroupName = "STEERING_COMMITTEE";
var executiveCommitteeGroupName = "EXECUTIVE_COMMITTEE";
var executiveBoardGroupName = "EXECUTIVE_BOARD";
var pressTeamGroupName = "PRESSTEAM";

var baseUserName = "bmuser";
var firstName = "BM";
var baseLastName = "USER";
var startEmailAddress = baseUserName;
var endEmailAddress = "@bestmoney.com";
var password = "1234";
var startUserNumber = 1;
var endUserNumber = 20;

// Create the Best Money groups that we need
var parentGroup = groups.getGroup(bestMoneyParentGroupName);
if (parentGroup != null) {
    logger.log("Root group " + bestMoneyParentGroupName + " already
        exists, aborting user and group creation.");
} else {
    var parentGroup = groups.createRootGroup(bestMoneyParentGroupName,
        bestMoneyParentGroupName);
    var sysAdmins = parentGroup.createGroup(sysAdminsGroupName,
        sysAdminsGroupName);
    var sectionCommittee =
        parentGroup.createGroup(sectionCommitteeGroupName,
            sectionCommitteeGroupName);
    var steeringCommittee =
        parentGroup.createGroup(steeringCommitteeGroupName,
            steeringCommitteeGroupName);
    var executiveCommittee =
        parentGroup.createGroup(executiveCommitteeGroupName,
            executiveCommitteeGroupName);
    var executiveBoard =
        parentGroup.createGroup(executiveBoardGroupName,
            executiveBoardGroupName);

```



```
var pressTeam = parentGroup.createGroup(pressTeamGroupName,
    pressTeamGroupName);

// Loop and create test users
for (var userNumber = startUserNumber; userNumber <=
    endUserNumber; userNumber++) {
    var userName = baseUserName + userNumber;
    var lastName = baseLastName + " " + userNumber;
    var emailAddress = startEmailAddress + userNumber +
        endEmailAddress;

    // Create new user and
    // enable account so user can login and so admin can edit user
    var enableAccount = true;
    var newUser = people.createPerson(userName, firstName,
        lastName, emailAddress, password, enableAccount);

    // Make sure home folders can be seen only by the owner
    var homeFolder = newUser.properties["cm:homeFolder"];
    homeFolder.setInheritsPermissions(false);

    // Populate all groups with some users
    var username = newUser.properties["cm:userName"];
    if (userNumber == 1) {
        sysAdmins.addAuthority(username);
        logger.log("Created new system admin user: " + userName + " and
            added to group: " + sysAdminsGroupName);
    } else if (userNumber > 1 && userNumber <= 5) {
        executiveCommittee.addAuthority(username);
        executiveBoard.addAuthority(username);
        logger.log("Created new user: " + userName + " and added to
            groups: " + executiveCommitteeGroupName + ", " +
            executiveBoardGroupName);
    } else if (userNumber > 5 && userNumber <= 10) {
        sectionCommittee.addAuthority(username);
        logger.log("Created new user: " + userName + " and added to
            group: " + sectionCommitteeGroupName);
    } else if (userNumber > 10 && userNumber <= 15) {
        steeringCommittee.addAuthority(username);
        logger.log("Created new user: " + userName + " and added to
            group: " + steeringCommitteeGroupName);
    } else if (userNumber > 15 && userNumber <= 20) {
        pressTeam.addAuthority(username);
    }
}
```

```

        logger.log("Created new user: " + userName + " and added to
            group: " + pressTeamGroupName);
    }
}

logger.log("Users and groups created");
}

```

To create the groups and the users we use the special root script objects `groups` and `people`. The groups are organized under the `BEST_MONEY` root group, which makes it easy to find them if there are other group hierarchies.

The first user that is created is the Best Money system administrator and then five users are added to the different groups, so it is easy to test the permissions that will be set up. Each user's home folder has been set up so only the owner can view it.

There is no need to add the users to the **EVERYONE** group as that is a special pseudo system group that will automatically contain all users. You can see it when you invite users to a space but you cannot see it when managing groups via the Alfresco Explorer UI and you cannot add users to it.

To use this script log in as admin and just add it to the `/Company Home/Data Dictionary/Scripts` folder and then it can be run as an action from any folder.

 To execute a script action, select the **More Actions** link and then the **View Details** link followed by the **Run Action** link. Now select the **Execute script** action and then as a value select the uploaded JavaScript.

Setting up the folder hierarchy

The folder hierarchy that we have defined in the specification for Best Money can be set up in different ways. We can just create it manually via the Alfresco Explorer UI, Alfresco Share UI, or CIFS. We can also use a script to create the folder hierarchy. A combination of manual set up and scripting is probably a good way to go in many cases.

 Remember to be logged in as admin when setting up the folder hierarchy as otherwise you might not have permission to set up folders and permissions.

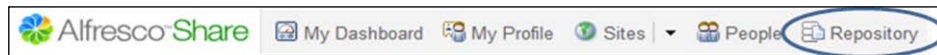
Using CIFS to set up folders

The CIFS interface is a good starting point when creating folders as it is easy and very fast to create folders this way. We can also easily copy similar folder hierarchies via this interface. However, as soon as we want to set metadata for folders, configure permissions, set up rules, change icon, and so on we need to use one of the Alfresco UIs.

Using the Alfresco user interfaces to set up folders

We can set up folders from both the Alfresco Explorer UI and the Alfresco Share UI. However, the Share UI is nicer, easier, and more intuitive than the Explorer UI. The Share UI will eventually replace the Explorer UI so we will use the Share UI at all times, except when the functionality is only available in the Explorer UI.

The Alfresco Share UI was originally designed as a sharing and collaboration environment where users worked with content belonging to a so-called *site*. These days a lot of the Alfresco Explorer document management functionality is also available in the Share UI. We can access it by clicking on the **Repository** icon in the top toolbar:



To start creating the folders that we have specified, click on the **New Folder** button above the folder list. Then we fill in the properties for the folder, as shown in the following screenshot:

A screenshot of the 'New Folder' dialog box. The title bar says 'New Folder' with a close button. The main section is titled 'New Folder Details'. It contains three input fields: 'Name:' with a red asterisk, containing the text 'Meetings'; 'Title:' which is empty; and 'Description:' containing the text 'All arrangements and documentation around meetings, conferences, committees, forums, projects, and seminars etc.'. At the bottom are 'Submit' and 'Cancel' buttons.

Currently, it is not possible to set a different folder icon via the Alfresco Share UI and it will also not display any other icon than the standard folder/directory icon, even if a different icon is set from the Alfresco Explorer UI.

The rest of the folders from the folder template specification are created in the same way.

Using scripts to set up folders

There are situations when using any of the Alfresco clients to set up folders manually is just too time consuming and some kind of bulk upload/update functionality is needed. We might have many folders, and maybe also documents, that should have the same or very similar title, descriptions, or other metadata. A customer could supply a list of folders (maybe in the thousands) in, for example, an Excel spreadsheet and we have the task to set them up in the repository.

Alfresco does not really provide a bulk change functionality to, for example, set the description of 20 nodes at the same time. So it can be quite time consuming to start doing all these updates manually. Some things we can do quickly via CIFS, but most things that have to do with updating the metadata we cannot.

A good solution for this is to use JavaScripts as helpers. They can easily apply changes to multiple repository nodes at the same time.

Updating folder icons

For example, to update the folder icon for all subfolders in a particular folder, we would use a script as follows:

```
// Get all children under current space
var children = space.children;

// Loop through and set icon for sub-folders
for (var i = 0; i < children.length; i++) {
    if (children[i].isContainer) {
        children[i].properties["app:icon"] = "space-icon-doc";
        children[i].save();
    }
}
```

This script gets all children, including folders and documents, under the current folder and then updates the icon for any child that is a folder (that is, the `isContainer` method checks if it is a folder, uses `isDocument` to check if it is a file). Current folder is the folder from which the script was run.

To find out what properties you can update, and what values can be set for them, it is usually a good idea to do it manually first for one node and then check out what property and value was set via the Alfresco Node Browser.

Adding groups of folders to other folders

Another common scenario is that you have a group of folders with special permission settings and rules that you want to copy to several other folders. To start doing this via the Alfresco UI will take you a lot of time or be impractical, if this needs to be done for hundreds of folders.

Also, when you copy folders in the Alfresco Explorer UI, the permissions are not copied with the folders. Doing it from CIFS does not work either, as neither the permissions nor the custom folder properties are copied.

To get around these problems and limitations we will create a "copy set of folders" script. This script will copy all subfolders from the /Company Home/CopyFrom folder to all the subfolders of the folder from where the script is executed.

The script looks like this:

```
// Get Set of subfolders to copy
var tempFolder = companyhome.childByNamePath("CopyFrom");
var foldersToCopy = tempFolder.children;

// Get all children under current space
var children = space.children;

// Loop thru subfolders and add set of folders to each one
for each (child in children) {
    if (child.isContainer) {
        for each (folderToCopy in foldersToCopy) {
            // Copy folder and its properties
            var copy = folderToCopy.copy(child);

            // Copy local folder permissions
            var permissions = folderToCopy.directPermissions;
            if (permissions != undefined) {
                for each (permission in permissions) {
                    if (permission != undefined) {
                        var permissionTokens = permission.split(";");
                        var authorityId = permissionTokens[1];
                        var permissionName = permissionTokens[2];
                        copy.setPermission(permissionName, authorityId);
                    }
                }
            }
        }
    }
}
```

```
        }  
    }  
    copy.setInheritsPermissions(folderToCopy.inheritsPermissions());  
  
    // Rules are copied automatically  
    }  
}  
}
```

To use the script mentioned earlier, set up the folders you want to copy including rules and permissions under the `/Company Home/CopyFrom` folder and then navigate to the top folder that has all the subfolders that should have all folders in `CopyFrom` copied to them.

The script first copies the folder and its properties (that is, metadata) from `CopyFrom` to the destination subfolder. Then the folder's permissions are copied; only the local permissions for the folder are copied. If we also wanted to copy the inherited permissions then we can change the following line:

```
var permissions = folderToCopy.directPermissions;
```

to:

```
var permissions = folderToCopy.permissions;
```

These methods return a list of permissions in the following format:

```
" [ALLOWED | DENIED] ; [USERNAME | GROUPNAME] ; PERMISSION"
```

Such as for example:

```
"ALLOWED;SYSTEM_ADMINS;Coordinator"
```

Before running this script make sure to test it first on some temporary folders, so you are aware of exactly what will be copied and also the end result.

Importing folders from an Excel file

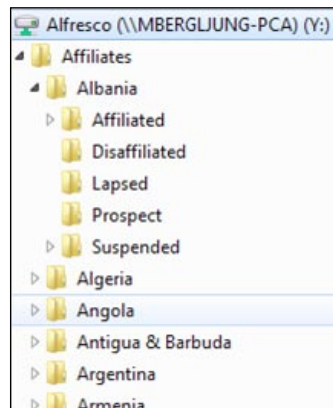
Another thing that is often needed is to create folders from a Microsoft Excel file. Let's say you are on a project and the client tells you, here is an Excel file with a list of folders that should be created including related metadata. And when you open the file there are hundreds of folders, maybe even thousands, with extra metadata. What to do? In these cases, we need to be able to automatically create the folders via a script.

In the following example, we assume that Best Money has lots of affiliates around the world and we have been given an Excel file with all these affiliates and we are supposed to create folders for them under correct country and correct affiliate status. The Excel file looks something like this:

	A	B	C	D	E	F
1	Primary Name	Short Name	Affiliate No	Affiliate Status	Country Short Name	Aff Region Name
2	Congo affiliate long name	CA1	1001	Prospect	Congo DR	Africa / Arab World
3	Belgium affiliate long name	BA1	1002	Affiliated	Belgium	Europe
4	India affiliate long name	IA1	1003	Affiliated	India	Asia Pacific

The **Primary Name** plus the **Affiliate No** should be used as `cm:name` and the **Short Name** should be used as `cm:title`. The **Country** and **Affiliate Status** fields indicate under which folders the affiliate should be organized. The `cm:description` property should be set to "The top folder for this affiliate" for all affiliates.

The existing folder structure under which these affiliate folders should be created looks like this:



The Excel file needs to be uploaded to some folder in Alfresco for the script to be able to read it. Before uploading it, we should convert it to a CSV file so it is easy to read from the script.

The script looks like this:

```
// Read content from current document and split it into separate lines
var affiliateLines=document.properties.content.content.split("\r\n");

if (affiliateLines == null) {
    logger.log("No text lines available");
} else {
```

```

        logger.log(affiliateLines.length + " lines available for
        processing");
    }

    var primaryName = null;
    var shortName = null;
    var number = null;
    var status = null;
    var country = null;

    if (affiliateLines != null) {
        for each (affiliateLine in affiliateLines) {
            // Extract the different affiliate fields
            var affiliateTokens = affiliateLine.split(",");
            primaryName = affiliateTokens[0].trim();
            shortName = affiliateTokens[1].trim();
            number = affiliateTokens[2];
            status = affiliateTokens[3].trim();
            country = affiliateTokens[4].trim();

            logger.log("Processing : " + primaryName + ", " + shortName + ", "
                + number + ", " + status + ", " + country);

            // Setup Affiliate folder name
            var numberString = "";
            if (number != null && number != "") {
                numberString = " - " + number;
            }
            var affiliateFolderName = primaryName + numberString;
            // Replace stuff that is not valid in a filename
            affiliateFolderName = affiliateFolderName.replaceAll("/", "-");
            affiliateFolderName = affiliateFolderName.replaceAll("\\", "");

            // Find folder to create Affiliate folder under
            // /Company Home/Affiliates/[Countries*]/[Affiliate Status*]
            var affiliateParentFolderPath = "Affiliates/" + country + "/" +
                status;
            affiliateParentFolder =
                companyhome.childByNamePath(affiliateParentFolderPath);

            if (affiliateParentFolder != null) {
                // Check if folder already exists
                var affiliateFolder =
                    companyhome.childByNamePath(affiliateParentFolderPath + "/" +
                        affiliateFolderName);
                if (affiliateFolder == null) {
                    // Create Affiliate folder

```



```
affiliateFolder =
    affiliateParentFolder.createFolder(affiliateFolderName);
if (affiliateFolder != null) {
    affiliateFolder.properties["app:icon"] = "space-icon-doc";
    affiliateFolder.properties["cm:title"] = shortName;
    affiliateFolder.properties["cm:description"] = "The top
        folder for this affiliate.";
    affiliateFolder.save();

    affiliateFolder.setPermission("Coordinator",
        "SYSTEM_ADMINS");
} else {
    logger.log("Affiliate folder: " + affiliateFolderName + "
        could not be created");
}
} else {
    logger.log("Affiliate folder: " + affiliateFolderName + "
        already exists, no need to create it");
}
} else {
    logger.log("Could not find affiliate parent folder: " +
        affiliateParentFolderPath + ", affiliate folder " +
        affiliateFolderName + " has not been created");
}
}
}
```

So for each line in the CSV file, we extract the value of each property and then we locate the parent folder for the affiliate. If the parent folder is found, then we create the affiliate folder and set its metadata. In this example, we also set permissions for the new affiliate folder, but it is probably better to set up common permissions on parent folders. Otherwise, it will be quite time consuming later on if we ever need to change these permissions.



When uploading a CSV file into the Alfresco repository, we need to be careful to set correct character encoding. The encoding might be in, for example, Windows-1252 but Alfresco will sometimes set it to, for example, UTF-8 as default, which will make characters from many different languages not display correctly.

Cleanup folders

Now, let's say you created all these new affiliate folders but discovered that they weren't created correctly for some reason and you need to change the script and run it again. How do we rollback the created folders? We can create a cleanup script such as the following:

```

function removeAllChildren(folder) {
    var nodes = folder.children;
    for each (node in nodes) {
        node.remove();
    }
}

var countries = space.children;
for each (country in countries) {
    var affiliated = country.childByNamePath("Affiliated");
    var disaffiliated = country.childByNamePath("Disaffiliated");
    var lapsed = country.childByNamePath("Lapsed");
    var prospect = country.childByNamePath("Prospect");
    var suspended = country.childByNamePath("Suspended");

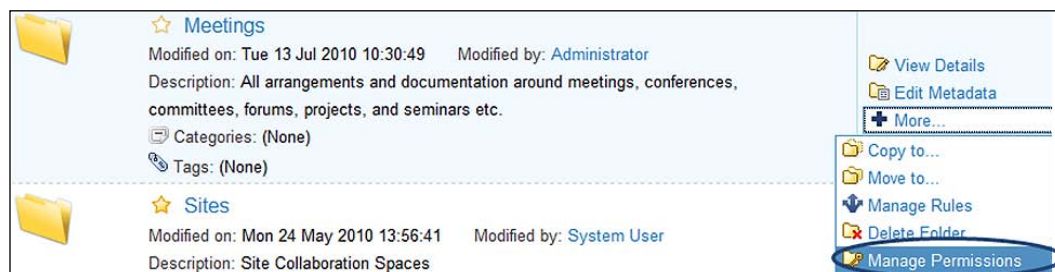
    removeAllChildren(affiliated);
    removeAllChildren(disaffiliated);
    removeAllChildren(lapsed);
    removeAllChildren(prospect);
    removeAllChildren(suspended);
}

```

This JavaScript can be run from the top *Affiliates* folder and it will remove all the affiliates folders created before.

Setting up folder permissions

To set up folder permissions, we will use the Alfresco Share UI as it provides a nicer and easier-to-use interface. The Share interface also offers more functionality such as displaying inherited permissions. To set up permissions for a folder, click on the **More...** menu item in the folder's menu on the right side. Then in the drop-down menu, select **Manage Permissions**:



When the SYSTEM_ADMINS group has been set up with Coordinator role for the /Company Home/Meetings folder, it looks like this:

The screenshot shows the 'Repository Browser' interface. At the top, there's a navigation bar with 'Admin Console', 'Help', 'Logout', and a search bar. Below this, the breadcrumb 'Location: Repository > Meetings' is displayed. The main section is titled 'Manage Permissions: Meetings' and includes a green checkmark icon next to 'Inherit Permissions' and a button labeled 'Add User/Group'. There are two tables: 'Inherited Permissions' and 'Locally Set Permissions'. The 'Inherited Permissions' table has two columns: 'User and Groups' and 'Role'. It shows 'EVERYONE' with the role 'Consumer'. The 'Locally Set Permissions' table has three columns: 'User and Groups', 'Role', and 'Actions'. It shows 'SYSTEM_ADMINS' with the role 'Coordinator' (indicated by a dropdown arrow) and an empty 'Actions' column. At the bottom, there are 'Save' and 'Cancel' buttons.

Inherited Permissions	
User and Groups	Role
EVERYONE	Consumer

Locally Set Permissions		
User and Groups	Role	Actions
SYSTEM_ADMINS	Coordinator ▼	

To add more permissions, click on the **Add User/Group** button in the upper-right corner. Then type in at least three characters of the username or group name and hit *Enter*. If a username or group matched, then we can select it by clicking on the **Add** button. The role that the group or username should have is set via the screen displayed next that we can see in the earlier screenshot.

Setting up business rules for folders

The folder specifications contain some business rules that we need to set up. The following rules need to be implemented, note that the Apply Versioning field is also implemented as a rule:

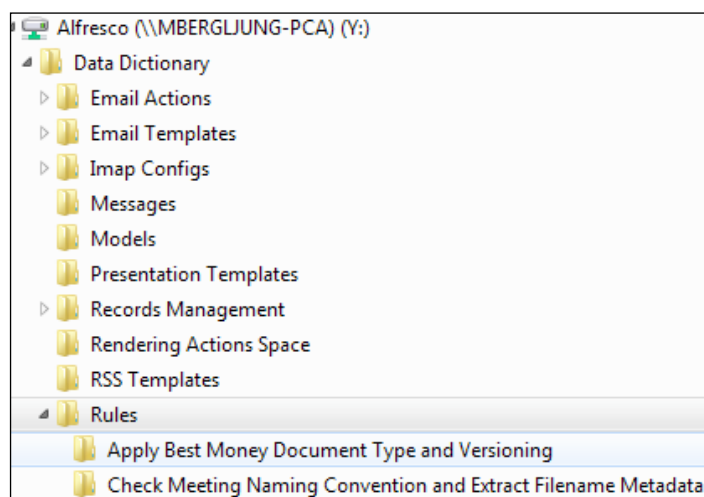
- Apply standard Best Money Meeting Document metadata via Form, nothing is mandatory
- Apply Versioning
- Check Naming Convention, abort upload if not correct
- Parse and Set Language and Department Metadata from Filename

Rules can be defined directly on a folder or we can link to a ruleset in another folder. If we have a set of rules, or just one rule, that are used over and over again in the folder hierarchy it might be useful to define them in one place and then link to them from all other places where they are used. This will make it easier to maintain/update the rules, if they ever change.

This can be done by first defining the ruleset for one folder and then link to this ruleset from all other folders that use the same rule. The rules cannot be defined in a library, so we have to create some kind of rules folder hierarchy for the project.

The `Meetings` top folder specification has two rules specified (the first two rules from the previous list) and the top folder specification for the `Meeting` folder template has two rules specified (that is, the last two rules of the previous list).

So it makes sense to define two reusable rulesets as follows for the Best Money project:



The ruleset's folder hierarchy can be created anywhere in the repository but it makes more sense to put it under the `/Company Home/Data Dictionary` folder as this is where all other document management functionality is located. This folder is usually not visible to end users, so they will also not be confused by the rule folders if we put them in the dictionary.



When looking at this, initially, it might make more sense to define one reusable rule per folder, making it possible to select and link to any number of rules when setting up folders. This is not possible as we can only link to one ruleset from a folder.

Defining the Apply Best Money Document Type rule

Let's start by defining the Apply Best Money Document Type rule. It requires the `bmc:document` custom type to be available. Creating the custom content model is the topic for the next chapter, but we will define a minimal type here so we can go ahead and create the rule. Open up the `content-model.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\model` directory and add the following type:

```
<type name="bmc:document">
  <title>Best Money Document</title>
  <parent>cm:content</parent>
  <properties>
    <property name="bmc:department">
      <title>The Best Money Department that created the
        document</title>
      <type>d:text</type>
      <multiple>true</multiple>
    </property>
    <property name="bmc:language">
      <title>Language that the document is written in</title>
      <type>d:text</type>
    </property>
  </properties>
</type>
```

In order for this new type to show up in Action wizards when we define rules, some custom web client configurations are needed. For the Alfresco Share web client, update the `share-config-custom.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_share\config\alfresco\web-extension` directory as follows:

```
<config evaluator="string-compare" condition="DocumentLibrary"
  replace="true">
  <types>
    <type name="cm:content">
      <subtype name="bmc:document" />
    </type>
  </types>
</config>
```

For the new type to show up in Alfresco Explorer, open up the `web-client-custom-config.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\ui` directory and add the following configuration:

```
<alfresco-config>
  <config evaluator="string-compare" condition="Action Wizards">
    <subtypes>
      <type name="bmc:document"/>
    </subtypes>
    <specialise-types>
      <type name="bmc:document"/>
    </specialise-types>
  </config>
</alfresco-config>
```

Now stop Alfresco and run the `deploy-alfresco-amp` ant target available in the `build.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk` directory. This updates the Alfresco Explorer client and adds the new content type. Run also the `deploy-share-jar` ant target to update the Alfresco Share web client. Start Alfresco again and the new type should be available together with the UI customizations.



If the Repository icon has disappeared from the Alfresco Share top toolbar when you restart Alfresco then make sure the following configuration is in the `share-config-custom.xml` file:

```
<config evaluator="string-compare" condition="RepositoryLibrary" replace="true">
  <visible>true</visible> . . .
```

To set up the rule we will use the Alfresco Share UI as it provides a nicer and easier-to-use interface. The Share interface also offers the rule linking functionality that is not available in Alfresco Explorer.

Navigate to the /Company Home/Data Dictionary/Rules folder and click on the **More...** menu item in the Apply Best Money Document Type and Versioning folder's menu on the right side. Then in the drop-down menu, select **Manage Rules**:



To create the rule, click on the **Create Rules** link on the initial page. Then fill in the following data for the Apply Versioning rule:

General	
Name: *	Apply Best Money Document Type
Description:	Change the default cm:content type for uploaded document to bmc:document.
Define Rule	
When:	
Items are created or enter this folder	
▼	
☒ If all criteria are met:	
Content of type or sub-type	is Content
☐ Unless all criteria are met:	
▼	
Perform Action:	
Specialise type	Type: Best Money Document
Other Options	
☐ Disable rule	☐ Run rule in background
☒ Rule applies to subfolders	If errors occur run script Select...

After the **Create** button is clicked, the following page will be displayed with the rules for the current folder:

Location: Repository > Data Dictionary > Rules > Apply Best Money Document Type and Versioning

Apply Best Money Document Type and Versioning: Rules

Rules from Apply Best Money Document Type and Versioning

1 ☒

Apply Best Money Document Type
Change the default cm:content type for uploaded document to bmc:document.

Apply Best Money Document Type
Description: Change the default cm:content type for uploaded document to bmc:document.

Defining the Apply Versioning rule

To set up the versioning rule, we can click on the **New Rule** button directly as this rule should be part of the same ruleset. Then fill in the following data for the Apply Versioning rule:

General

Name: *
Apply Versioning

Description:
Apply the Versionable aspect to all documents dopped into this folder or any of its subfolders.

Define Rule

When:
Items are created or enter this folder

☒ If all criteria are met:
Content of type or sub-type is Content

☐ Unless all criteria are met:

Perform Action:
Add an aspect Versionable

Other Options

☐ Disable rule ☐ Run rule in background

☒ Rule applies to subfolders ☐ If errors occur run script Select...

Click the **Create** button so the rule is created and saved.



Make sure to only apply versioning to content and not folders by setting Content of type or subtype = Content.

Folders are not versionable unless the WCM module is used, which supports a more advanced versioning management (AVM) system.

Defining the Check Naming Convention rule

The filenames for documents that are stored in any of the meeting folders have to follow a naming convention. We will enforce the naming convention with a rule that is implemented with a JavaScript.

The script looks like this:

```
// Regulars Expression Definition
var re = new RegExp("^\\d{2} (Ar|Ch|En|Fr|Ge|In|Jp|Po|Ru|Sp|Sw|Ta|Tu) - (A|HR|FM|FS|FU|IT|M|L)\\.\\.\\d{2}_\\d{1,3}_annex.*");

logger.log("Check Meeting Naming Convention RegExp = " + re);

if (re.test(document.name) == false) {
    var exampleNamingConvention = "06En-FM.02_3_annex1.doc";
    var errorMsg = "<<ERROR: Filename " + document.name + " does not follow naming convention for this folder. Example of naming convention: " + exampleNamingConvention + ". Regular Expression used: " + re + ">>";
    logger.log(errorMsg);

    // Cancel Transaction so document is not stored
    throw errorMsg;
}
```

Store this script as `checkMeetingNamingConvention.js` in the `/Company Home/Data Dictionary/Scripts` folder. Make sure the `Mimetype` is set to `Java Script` otherwise the script will not show up when selecting script to execute when defining a rule.

Navigate to the /Company Home/Data Dictionary/Rules folder and click on the **More...** menu item in the Check Meeting Naming Convention and Extract Filename Metadata folder's menu on the right side. Then in the drop-down menu, select **Manage Rules** followed by **Create Rules** on the following page. Then fill in the following data for the rule as follows:

General

Name: *

Check Meeting Naming Convention

Description:

Checks the meeting document filenames so they follow a naming convention specified in the checkMeetingNamingConvention.js script.

Define Rule

When:

Items are created or enter this folder

If all criteria are met:

Content of type or sub-type is Content

Unless all criteria are met:

Perform Action:

Execute script checkMeetingNamingCc

Other Options

☐ Disable rule
 ☐ Run rule in background

☒ Rule applies to subfolders
 If errors occur run script Select...

The rule is applied to all content added to the folder or subfolders. Note that if a folder is created, this rule will not be executed as we have specified Content of type or sub-type = Content. The rule action is specified as the checkMeetingNamingConvention.js script that we uploaded earlier. Click on the **Create** button, so the rule is created and saved.

Defining the Extract Meeting Filename Metadata rule

This rule is closely related to the Check Meeting Naming Convention rule. It is defined to be run just after that rule in some folder specifications. It should extract some metadata that can be found in the filename. The metadata will be extracted with a JavaScript.

The script looks like this:

```
if (document.hasAspect("bmc:document_data")) {
    var language = document.name.substring(2, document.name.indexOf("-"));
    var department = document.name.substring(document.name.indexOf("-") + 1);
    department = department.substring(0, department.indexOf("."));

    logger.log("Language = " + language);
    logger.log("Department = " + department);

    document.properties["bmc:language"] = language;
    document.properties["bmc:department"] = department;
    document.save();
} else {
    logger.log("Aspect bmc:document_data is not set for document " + document.name);
}
```

Store this script as `extractMeetingFilenameMetadata.js` in the `/Company Home/Data Dictionary/Scripts` folder. Make sure the `Mimetype` is set to `Java Script` otherwise the script will not show up when selecting script to execute when defining a rule.

Navigate to the `/Company Home/Data Dictionary/Rules` folder and click on the **More...** menu item in the `Check Meeting Naming Convention` and `Extract Filename Metadata` folder's menu on the right side. Then in the drop-down menu, select **Manage Rules** followed by **Create Rules** on the following page. Then fill in the following data for the rule as follows:

General

Name: *
Extract Meeting Filename Metadata

Description:
Extracts Best Money custom metadata from filenames of meeting documents.

Define Rule

When:
Items are created or enter this folder

☒ If all criteria are met:
Content of type or sub-type is Content

☐ Unless all criteria are met:

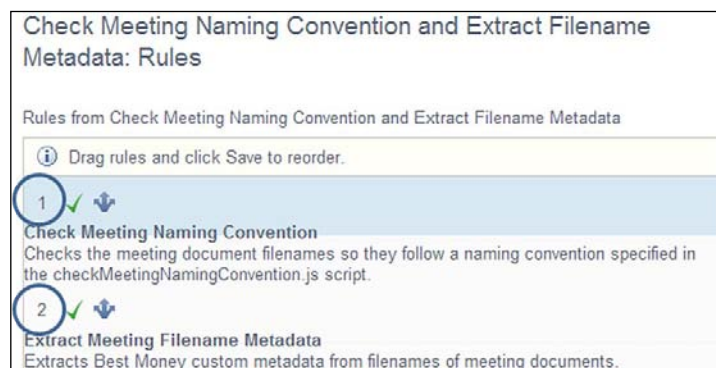
Perform Action:
Execute script extractMeetingFilename

Other Options

☐ Disable rule ☐ Run rule in background
☒ Rule applies to subfolders If errors occur run script Select...

The rule is applied to all content added to the folder or subfolders. Note that if a folder is created, this rule will not be executed as we have specified Content of type or sub-type = Content. The rule action is specified as the `extractMeetingFilenameMetadata.js` script that we uploaded earlier. Click on the **Create** button so the rule is saved.

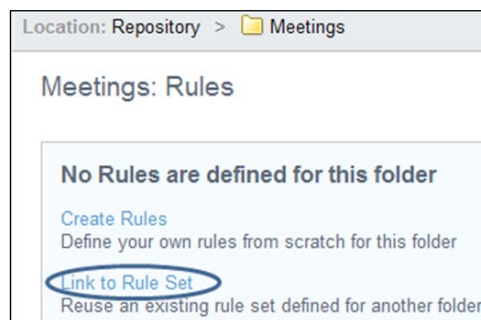
In this case, the execution order of the rules is important, so make sure the order is as shown in the following screenshot:



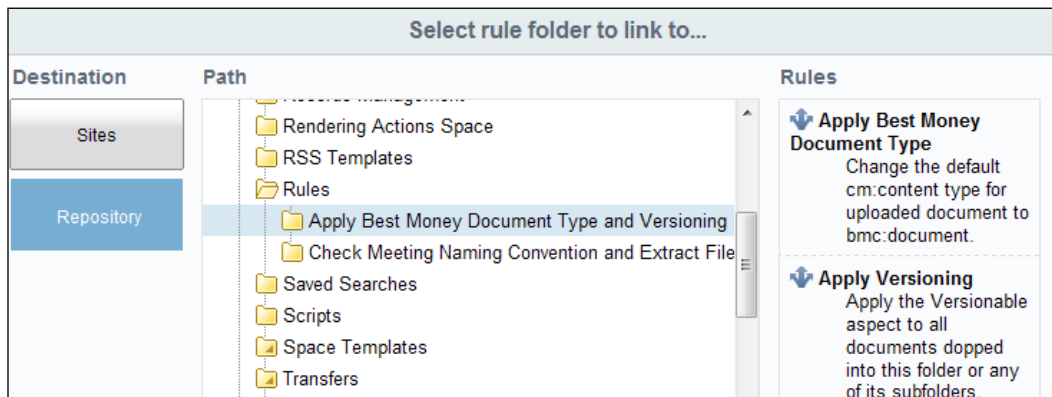
Linking to the rules

Now, we have a library of rulesets that we can reuse throughout the document management implementation. To demonstrate how to link to a ruleset we will implement the rules for the Meeting folder specification that requires the Apply Best Money Document Type and Apply Versioning rules.

Start by navigating to the **/Company Home** folder (that is, this folder is just called **Repository** in Alfresco Share) and click on the **More...** menu item in the **Meetings** folder's menu on the right side. Then in the drop-down menu, select **Manage Rules**. The following page will be displayed where we should click on the **Link to Rule Set** link:



Then click on the **Repository** Destination and then navigate to the **Rules** folder and select the **Apply Best Money Document Type and Versioning** ruleset:



Create the link by clicking on the **Link** button.

Error handling

When a rule is executed, there might be an error state as a result or we might, for example, throw an exception to abort a document upload. In these cases, it would be good to be able to run a script to display some useful information.

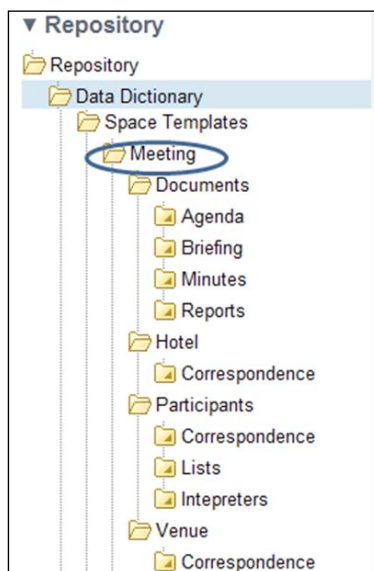
This is possible to do when the rule is defined; we can then specify what script that should be run in, in case of an error state.

This is only available if the rule is set up via Alfresco Share.

Setting up space templates

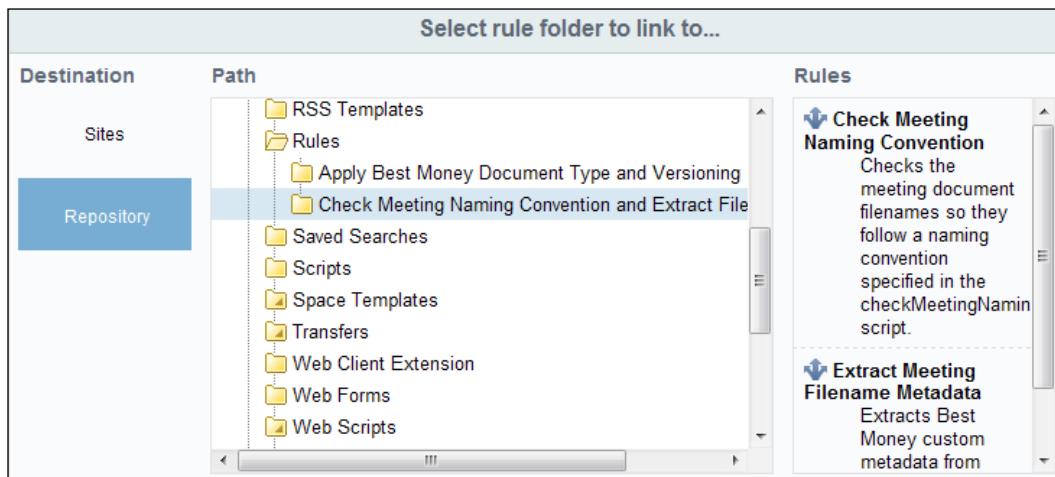
The Best Money folder specification has one space template defined for meetings called **Meeting**. For a folder hierarchy to be recognized as a space template, it needs to be located under the /Company Home/Data Dictionary/Space Templates folder. Creating the folders that are part of the space template is no different from creating folders somewhere else in the Repository.

Create the following folder hierarchy according to the Meeting specification:



There are two rules defined in the specification for the `Meeting` folder template and they correspond to the Check Meeting Naming Convention and Extract Filename Metadata ruleset. To link to this ruleset start by navigating to the `/Company Home/Space Templates` folder and click on the **More...** menu item in the **Meeting** folder's menu on the right side. Then in the drop-down menu, select **Manage Rules**. The following page is displayed where we should click on the **Link to Rule Set** link.

Now click on the **Repository** Destination and then navigate to the **Rules** folder and select the **Check Meeting Naming Convention and Extract Filename Metadata**:



Create the link by clicking on the **Link** button. The Meeting hierarchy folder template is now ready to be used.

To create a new folder based on this template in Alfresco Explorer, we first navigate to the **Meetings** subfolder that fits our meeting type. For example, if it is a committee-related meeting then we would navigate to `/Company Home/Meetings/Committee` and then click on the **Create** link. Then in the drop-down menu, select **Advanced Space Wizard**. In step One, choose to create the space **Using a template**. Then select the **Meeting** template.

In Alfresco Share it is not, at the time of writing, possible to create a folder hierarchy based on a space template. We would have to use Alfresco Explorer for this. Until Alfresco Share has been extended to support all features available in Alfresco Explorer, we are likely to be using both UIs for some time.

Configuring details list view for folder and file display

By default, the Alfresco Explorer UI client displays folders and files in an icon view. Most customers like Best Money are going to want to change that to a details list view instead to get more information on each page.

To change from icon view to details list view open up the `web-client-custom-config.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\ui` directory and add the following configuration:

```
<config evaluator="string-compare" condition="Views">
  <views>
    <view-defaults>
      <browse>
        <view>details</view>
        <page-size>
          <list>10</list>
          <details>25</details>
          <icons>10</icons>
        </page-size>
      </browse>
    </view-defaults>
  </views>
</config>
```

At the same time we also change the number of items per page to 25, which is more suitable for a list view.

Now, stop Alfresco and run the `deploy-alfresco-amp` ant target available in the `build.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk` directory. This updates the Alfresco Explorer client and adds the new content type.

In Alfresco Share, we can also manually switch between different list views (that is, Simple View and List View) that show more or less data for each folder or document. However, it is not possible to permanently set this via `share-config-custom.xml`.

Configuring Google-Like search

When we search via Alfresco Explorer, and specify multiple search terms, an implicit OR is by default used between search terms. It would be better if it was an implicit AND as that is how it works in Google and most people are used to that.

To configure this, open up the `web-client-custom-config.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\ui` directory and add the following configuration:

```
<config>
  <client>
    <search-and-terms>true</search-and-terms>
  </client>
</config>
```

Setting up document review periods

The Best Money folder specification specifies that documents in the `Press` folder should be reviewed every fifth year. This is closely related to Records Management but it is not complex enough to require the use of the records management module. Instead, we can use the built-in Scheduler in Alfresco to set up a review process based on review period metadata that has been set on the folder.

A JavaScript will be used to check if a document is up for review and this script will also take care of notifying the members of the `DOC_REVIEWERS` group as required.

Adding the reviewable aspect

Let's first start with defining the document review properties that should be applied to the folder. As these properties can be applied to any folder, and they are not related to what type of folder it is, we will define these properties in an Aspect as follows:

```
<aspect name="bmc:reviewable">
  <title>Reviewable Folder</title>
  <properties>
    <property name="bmc:reviewPeriod">
      <title>The number of years until documents in this folder
        should be reviewed</title>
      <type>d:int</type>
      <mandatory>true</mandatory>
      <default>5</default>
    </property>
    <property name="bmc:includeSubFolders">
      <title>Should sub folders also be affected by this review
        period</title>
      <type>d:boolean</type>
      <mandatory>true</mandatory>
    </property>
  </properties>
</aspect>
```

```

        <default>true</default>
    </property>
</properties>
</aspect>

```

The first property `reviewPeriod` will contain the number of years between reviews and is used when checking the modified date for a document. The second property `includeSubFolders` tells us if the folders' subfolders should also be reviewable folders with the same review period.

Add the above aspect definition to the `content-model.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\model` directory.

In order for this new aspect to show up in the UI, some custom web client configurations are needed. For the Alfresco Share web client, update the `share-config-custom.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_share\config\alfresco\web-extension` directory as follows and add the `bmc:reviewable` aspect:

```

<config evaluator="string-compare" condition="DocumentLibrary"
replace="true">
  <aspects>
    <!-- Aspects that a user can see -->
    <visible>
      <aspect name="cm:generalclassifiable" />
      <aspect name="cm:complianceable" />
      <aspect name="cm:dublincore" />
      <aspect name="cm:effectivity" />
      <aspect name="cm:summarizable" />
      <aspect name="cm:versionable" />
      <aspect name="cm:templatable" />
      <aspect name="cm:emailed" />
      <aspect name="emailserver:aliasable" />
      <aspect name="cm:taggable" />
      <aspect name="app:inlineeditable" />
      <aspect name="bmc:reviewable" />
    </visible>
    <!-- Aspects that a user can add. Same as "visible" if left empty
    -->
    <addable></addable>
  </aspects>
</config>

```

```
<!-- Aspects that a user can remove. Same as "visible" if left
empty -->
<removeable></removeable>
</aspects>
<types>
  <type name="cm:content">
    <subtype name="bmc:document" />
  </type>
</types>
</config>
```

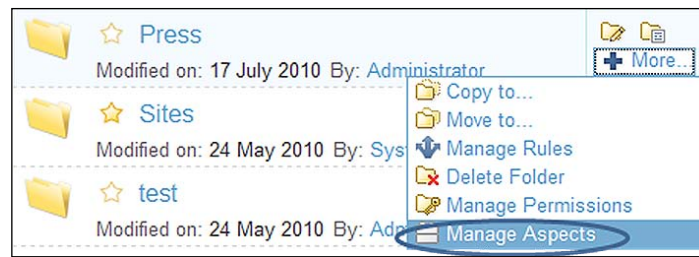
For the new aspect to show up in Alfresco Explorer, open up the `web-client-custom-config.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\ui` directory and add the following configuration:

```
<alfresco-config>
  <config evaluator="string-compare" condition="Action Wizards">
    <aspects>
      <aspect name="bmc:reviewable"/>
    </aspects>
    <subtypes>
      <type name="bmc:document"/>
    </subtypes>
    <specialise-types>
      <type name="bmc:document"/>
    </specialise-types>
  </config>
</alfresco-config>
```

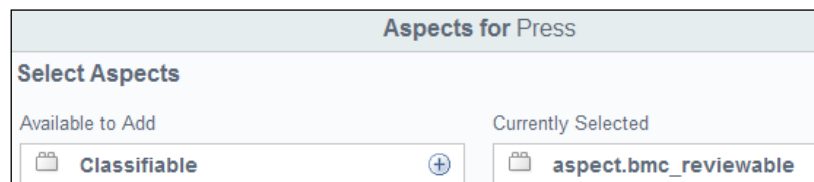
Now, stop Alfresco and run the `deploy-alfresco-amp` ant target available in the `build.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk` directory. This updates the Alfresco Explorer client and adds the new content type. Also run the `deploy-share-jar` ant target to update the Alfresco Share web client. Start Alfresco again and the new type should be available together with the UI customizations.

Setting a review period for a folder

To set up the five year review period for the `Press` folder, do the following. Click on **/Company Home** (that is, **Repository**) so the top folders are listed. Then click on the **More...** menu item for the **Press** folder and then select the **Manage Aspects** menu item in the pop-up menu:



In the next dialog, click the plus icon after the `bmc_reviewable` aspect:



We have not yet defined any i18n properties for the aspect, so it is displayed with the `aspect.bmc_reveiwable` name (we will see how to do this in the next chapter). Click **Apply changes** so the aspect is applied to the `Press` folder. The next thing we need to do is set up a script that uses this aspect when it looks at the documents stored in this folder or any of its subfolders.

Creating script to check folder review periods

To check the review period settings for folders we will create a JavaScript. This script will start searching from `/Company Home` for folders with the `bmc:reviewable` aspect and then inspect the contained documents and decide if they are up for review or not. The first part of the script that searches for all the reviewable folders looks like this:

```
var today = new Date();
var docReviewList = new Array();
var store = "workspace://SpacesStore";
var query = "+PATH:\"/app:company_home/*\"
            +ASPECT:\"bmc:reviewable\"";
var reviewableFolders = search.luceneSearch(store, query);
```

When we have got all the reviewable folders, we loop through them and extract the `bmc:reviewPeriod` property, which we will use to calculate the latest date when documents should be reviewed. This last review date will be compared with the modified date for the documents that are contained in the folder. We also extract the `bmc:includeSubFolders` property, so we know if we should search for documents in subfolders:

```
for each (reviewableFolder in reviewableFolders) {
  var reviewPeriod =
    reviewableFolder.properties["bmc:reviewPeriod"];
  var lastReviewDate = new Date(new Date(today).setYear(1900 +
    today.getYear() - reviewPeriod));
  var includeSubfolders =
    reviewableFolder.properties["bmc:includeSubFolders"];
```

Next, we construct the documents query and search for all documents in the folder and subfolders. Then we loop through the documents and extract the `cm:modified` date, so we can compare that to the last review date for the folder. If the documents modified date is less than or equal to the last review date then this document is added to a list of documents that are up for review. To protect against the document being added to the review list next month also, we update the modified date for the document to today's date:

```
var wildcard = includeSubfolders ? "/*" : "/";
query = "+PATH:\""+reviewableFolder.qnamePath + wildcard + "\"";
var nodes = search.luceneSearch(store, query);
for each (node in nodes) {
  if (node.isDocument) {
    var modifiedDate = node.properties["cm:modified"];
    if (modifiedDate <= lastReviewDate) {
      var docToReview = node.name + "
        (" + reviewableFolder.displayPath + "/" +
        reviewableFolder.name + ")";
      docReviewList.push(docToReview);
      node.properties["cm:modified"] = today;
      node.save();
    }
  }
}
```

Now when we got the list of documents that should be reviewed, we compile an e-mail with this list and send it to the members of the DOC_REVIEWERS group:

```
// Setup email body text
var emailBodyText = "The following documents need to be reviewed:\n\r\n\r";
for each (docToReview in docReviewList) {
    emailBodyText = emailBodyText + docToReview + "\n\r";
}

// Create an array with all users and groups that the email should be sent to
var reveiwersGroupName = "GROUP_DOC_REVIEWERS";
var reviewerGroups = new Array(reveiwersGroupName);

// Create mail action and send emails
var mail = actions.create("mail");
mail.parameters.to_many=reviewerGroups;
mail.parameters.subject = "Documents For Reviewing";
mail.parameters.from = "do-not-reply@bestmoney.com";
mail.parameters.text = emailBodyText;
// execute action against current space
mail.execute(space);
```

This script is now ready to be tested:

- Upload a couple of documents.
- Set date on the development machine to more than five years in the future.
- Upload the script to the /Company Home/Data Dictionary/Scripts directory and call it checkReviewPeriodAndSendEmail.js.
- In Alfresco Explorer, click on the **More Action** link for some folder.
- Click **View Details**.
- Then **Run Action**.
- Followed by **Execute Script**.
- As script value select checkReviewPeriodAndSendEmail.js.

Setting up a scheduler that runs review folder content script

When the script has been tested and verified to work as expected, we need to set up a scheduled action that executes this script at the end of each month. We can do this by using the Quartz Scheduler that Alfresco embeds and a Cron job.

We will start by configuring an action template that points out the script that the Cron job will run. Open up the `services-context.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\context` directory and add the following configuration:

```
<bean id="com.bestmoney.dm.runCheckReviewPeriodScriptAction"
class="org.alfresco.repo.action.scheduled.
SimpleTemplateActionDefinition">
  <property name="actionName">
    <value>script</value>
  </property>
  <property name="parameterTemplates">
    <map>
      <entry>
        <key>
          <value>script-ref</value>
        </key>
        <!-- element and value has to be on the same line -->
        <value>selectSingleNode('workspace://SpacesStore', 'lucene',
'PATH: "/app:company_home/app:dictionary/app:scripts/
cm:checkReviewPeriodAndSendEmail.js"')</value>
      </entry>
    </map>
  </property>
  <property name="templateActionModelFactory">
    <ref bean="templateActionModelFactory"/>
  </property>
  <property name="dictionaryService">
    <ref bean="DictionaryService"/>
  </property>
  <property name="actionService">
    <ref bean="ActionService"/>
  </property>
  <property name="templateService">
    <ref bean="TemplateService"/>
  </property>
</bean>
```

Then set up the Cron Job to run the action we just defined, every month:

```
<bean id="com.bestmoney.dm.invokeCheckReviewPeriodScriptActionEveryMonth"
class="org.alfresco.repo.action.scheduled.
CronScheduledQueryBasedTemplateActionDefinition">
  <property name="transactionMode">
    <value>UNTIL_FIRST_FAILURE </value></property>
  <property name="compensatingActionMode">
    <value>IGNORE</value></property>
  <property name="searchService">
    <ref bean="SearchService"/></property>
  <property name="templateService">
    <ref bean="TemplateService"/></property>
  <property name="queryLanguage"><value>lucene</value>
</property>
  <property name="stores">
    <list><value>workspace://SpacesStore</value></list>
  </property>
  <!-- A query is not being used, just set the path to company home,
    which results in one hit that we will run the script against -->
  <property name="queryTemplate">
    <value>PATH:"/app:company_home"</value>
  </property>
  <!-- Execute the check review period action at the end of every
    month. -->
  <property name="cronExpression">
    <value> 0 0 22 L * ?</value></property>
  <property name="jobName">
    <value>checkReviewPeriodForFolders</value></property>
  <property name="jobGroup"><value>BestMoney</value>
  </property>
  <property name="triggerName">
    <value>lastDayOfMonthTrigger</value></property>
  <property name="triggerGroup">
    <value>BestMoneyTriggerGroup</value></property>
  <property name="scheduler">
    <ref bean="schedulerFactory"/></property>
  <property name="actionService">
    <ref bean="ActionService"/></property>
  <property name="templateActionModelFactory">
    <ref bean="templateActionModelFactory"/></property>
  <property name="templateActionDefinition"><ref
    bean="com.bestmoney.dm.runCheckReviewPeriodScriptAction"/>
```



```
</property>
<property name="transactionService">
  <ref bean="TransactionService"/></property>
<property name="runAsUser">
  <value>System</value></property>
</bean>
```

To test and verify that the script is actually run by the Cron job, we can set the Cron expression so the script is executed every three minutes, use the expression `0 0/3 * * * ?` in this case.

The last Spring we need to define is the following:

```
<bean id="templateActionModelFactory" class="org.alfresco.repo.action.
scheduled.FreeMarkerWithLuceneExtensionsModelFactory">
  <property name="serviceRegistry">
    <ref bean="ServiceRegistry"/>
  </property>
</bean>
```

Now, stop Alfresco and run the `deploy-alfresco-amp` ant target available in the `build.xml` file located in the `chapter_6_Code\bestmoney\alf_extensions\trunk` directory. This will install the scheduled Cron job.

Exporting and importing folders, users, and groups

Sooner or later we are going to need a convenient way of copying folders and users, and so on between Alfresco installations. Here are a couple of tips.

Copying folder hierarchies between Alfresco boxes

One good way to backup folders and space templates during development and deployment is to use **Alfresco Content Packages (ACP)**. We can export folders or space templates into ACP files from the Alfresco Explorer UI.

It is a good idea to have some kind of naming convention for the ACP files so everybody knows what they contain.

We can, for example, use the following naming convention for top folders (it is usually a good idea to split the ACP files into one per top folder):

```
<company>_<top folder>_FOLDER_HIERARCHY.acp
```

For the Meetings top folder, the file would then be called:

```
BM_MEETINGS_FOLDER_HIERARCHY.acp
```

And for space templates, we can use a similar naming convention (use one ACP file per space template):

```
<company>_<top folder>_FOLDER_TEMPLATE.acp
```

For the Meeting space template the ACP file would then be called:

```
BM_MEETING_FOLDER_TEMPLATE.acp
```

These ACP files can then be backed up and they can also be used when deploying the system in the different development, test, and production environments.

Copying users and groups between Alfresco boxes

Use a JavaScript for this, as explained earlier in this chapter.

Another way of doing this is to keep all the users and groups in a directory. Then we just need to hook Alfresco up to this directory and sync users and groups. This is probably a very good way of doing it if there are a lot of developers and environments, so we are sure everyone is using the same data.

Introduction to Records Management

During the design and implementation of the document management solution, we might get requirements from clients such as "keep the final version of this file for six years then destroy it", "retain this file for three years and then review", and so on. When requirements like this start popping up, we are entering the area of records management and need to think about if we should use a proper records management system, or if some customizations to the document management system would be enough to cover these sorts of requirements.

So what is records management then? It is a solution that provides a process for managing documents both in electronic form and in paper-based form. The process covers the whole lifecycle of the documents from creation to disposal and it provides controlled retention and destruction.

But this sounds like something that could be implemented with the DM solution, what is really the difference? The DM solution does not come with these kind of document management processes. We would have to create them from scratch like we did with the Review Period aspect that was applied to folders and then the Review Script that was kicked off at the end of each month by a Cron job.

The external requirements that usually drive records management implementations are legal and regulatory compliance. A record can be explained as an electronic or paper-based file or group of files that is maintained as evidence by an organization in pursue of legal obligations.

Examples of business records are meeting minutes, e-mails, employment contracts, accounting documents, business plans. Most business records have retention periods based on legal or regulatory requirements or internal company policies.

To understand why it might be a good idea to use an "off-the-shelf" product like the Alfresco RM module, we should have a look at the things involved in implementing an RM solution. Implementing an RM solution is not just about putting in place software that supports this. There is a lot of planning and information gathering that needs to take place before we start configuring stuff in the RM software.

We will have to:

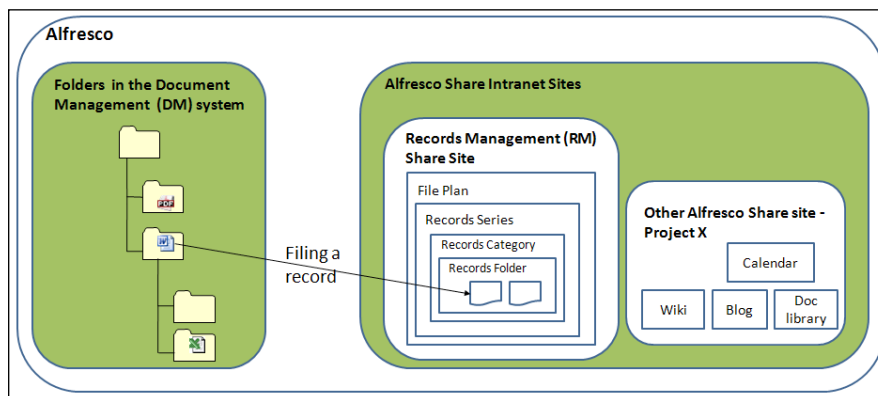
- Identify records requiring capture
- Find out who should have access to what records inside and outside the organization
- Develop a records storage plan/file plan

So an RM solution needs to support:

- Management of Filing Plans
- Execution of retention policies
- Searching for records

Alfresco records management

The Alfresco records management functionality is implemented as an Alfresco Share site. This is how it relates to the Alfresco document management functionality:



Here, we can see that the RM module is implemented as an Alfresco Share site. The main function is the File Plan Management. The structure of the File Plan hierarchy reflects business functions and comprises the following predefined levels:

Record series	A container that holds filing categories. Comparable to the filing cabinet when filing paper based records. Example: Legal
Record category	Created under a record series and contains the retention and disposition instructions for its contained filing folders. All filing folders and records under a category have the same retention period and disposition schedule. Comparable to a drawer in a filing cabinet. Example: Shareholder
Record folder	A record folder is created within a record category and it inherits the attributes of the record category. The record folder is also considered to be under the control of the record category. Once the record folder is created, and a disposition schedule is defined, restrictions apply. A record folder can be open or closed. A closed record folder cannot accept records for filing. Comparable to a folder in a drawer in a filing cabinet. Example: 2010Q2Public
Record	A record is a document under the control of records management, which is filed in a record folder.
Vital record	A vital record is considered to be essential to the operation of an organization. A vital record must be reviewed on a periodic basis, which is defined in the review schedule. The review schedule is defined on the record category or folder. Just as record folders appear to exist even though they are really no more than aggregations of records, so higher levels of the File Plan hierarchy seem to exist, though they are no more than aggregations of record folders and/or higher levels.

Each user is given a role that may or may not grant them permission to create the elements of the File Plan structure. You can file records and create the structure within the File Plan level in which you have permission.

To manage the File Plan, you need to:

- Design the File Plan structure using the record series, record category, and record folder hierarchy
- Upload electronic files and specify the location of non-electronic physical files
- Declare files as records

The records folders and filing hierarchies are actually stored in the DM system, but the underlying Node structure and Type of record nodes are different from the document nodes.

To file/create a new record we have to copy or move the file from the document management folder to the records management folder. The document that should be filed as a record does not actually have to be stored in the DM system; we could just upload it from our local hard disk or from a network drive.

The record does not even have to be an electronic one but could be a paper-based one and in this case, we would just store the location of the record in the RM system.

Summary

In this chapter, we have gone through how to design and implement document management solutions. We saw how a Folder Template could be used to specify/design everything that has to do with a folder such as properties (that is, metadata), permissions, rules, processes, and so on. The template could also be used to specify space templates.

Both the Alfresco Explorer client and the Alfresco Share client were used to implement document management functionality such as rules and permissions. The Alfresco Share client is the preferred choice because of its nicer UI. It's only in some cases that we really need to use Alfresco Explorer, such as when we want to create folder structures based on space templates.

We used JavaScripts to perform a lot of the document management functionality and it is really an excellent tool to use as we do not need to restart the server and we can update and create JavaScript code on the fly.

We also looked at how to manually implement records management functionality, such as review documents after a certain period. And finally, we introduced the Alfresco records management module and explained how it is related to the Alfresco DM functionality.

In the next chapter, we will look at how to define content models and this is closely related to what we did in this chapter, as when we design folders we also think about what domain metadata is needed.

7

Content Model Definition Solutions

One of the main differences between a network drive and a document management system is that the latter provides extra classification features. If we look at a document management system behind the scenes, we can see that everything in the repository is typically a node or an object, depending on what we want to call it. Properties are then set on the nodes so they become folders, files, categories, rules, forums, web pages, e-mails, people, groups, and so on. Basically, the nodes are classified so it is easier to search for them, so we know how to display them in the user interface, what they can be used for, and so on.

The properties that can be used to classify nodes, or if you prefer content, cannot be just any properties, as then the system would not know what each one of these properties represents. Because of this a document management system usually comes preconfigured with properties that can be used to classify content in the repository. These properties are usually organized into groups that are called either *Types* or *Aspects*.

Nodes do not live by themselves in the repository. They are related to one another in different ways. How the nodes are related to each other is defined with so-called *Associations*.

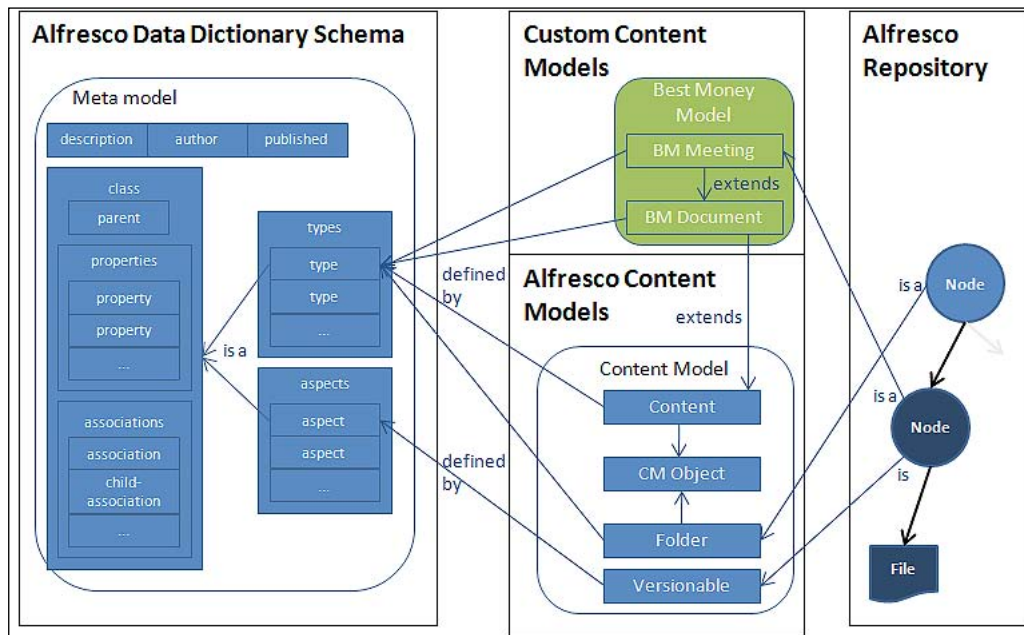
The Types, Aspects, Properties, and Associations are in turn organized into models that we call *Content Models*. Alfresco comes with a number of Content Models out of the box for different things like general folder and file content, workflow-related content, records content, web content, and so on.

It is also useful to be able to create content models related to specific domains such as Finance or Marketing. Alfresco cannot possibly know about all kinds of content that will be stored in the repository, so they just supply some generic standard content models that we can build on to be able to classify the particular domain that we happen to work in for the current project.

When we designed and created the Best Money folder structure, it also became apparent what custom types and properties we would need. There were the Best Money Document type and also the Best Money Meeting type. A meeting type is probably quite common, so why is Alfresco not providing such a type out of the box? This is because the properties that are part of the type will probably differ a lot from one domain/project to another.

So we can define new types and aspects forming new content models, but how do we know how to specify a type and a property with, for example, a data type integer? Alfresco also comes with a *Meta Model*. The Meta model defines what syntax we can use when defining our content models. It will, for example, define the syntax for how a type or an integer property should be defined.

The following diagram gives a simplified overview of the relationship between Alfresco meta model, Alfresco content models, and custom content models:



In the image, we can see two nodes in the repository: one folder node and one Best Money Meeting document node. The Meeting type extends the Best Money Generic Document type, which in turn extends the Alfresco Content type. The Meeting document node also got versioning turned on by having the Versionable aspect applied.

Both the custom content model and the Alfresco content model is defined by the Alfresco Data Dictionary Schema, which is the same as the Meta Model.

The whole thing with content models, types, aspects, and properties are not that far away from object-oriented concepts in programming. If we are familiar with object-oriented modeling, then we should not have any problem with content modeling.

This chapter will show you how to define new content models and what different entities can be used to define a content model.

You will learn:

- Interpreting the XSD Schema/meta model that describes Alfresco's content modeling language
- Using the content models that come out of the box with Alfresco
- Creating content models with some useful design patterns
- Defining a new content model
- Showing new custom types and aspects in Alfresco Explorer and Alfresco Share

Meta Model XML schema

The Alfresco meta model contains the constructs or syntax that can be used to define content models. The meta model is defined in an XML Schema definition file called `modelSchema.xsd` located in the `alfresco-enterprise-sdk/lib/server/config/alfresco/model` directory. It is a good idea to load this file into our programming environment (that is Eclipse, IDEA, and so on) as it will then "help out" with the syntax when we create content models.

model

The `model` definition is the container of all other definitions and it contains the following:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of the content model
description	element (string)	[0..1]	Description of the content model
author	element (string)	[0..1]	The name of the author of this content model
published	element (date)	[0..1]	The date for when this new model was first published

Name	Type	Multiplicity	Description
version	element (string)	[0..1]	Current version number for this model
imports	element	[0..1]	Container for one or more imports of other content models referred to by this model
namespaces	element	[1..1]	Mandatory container for one or more namespace definitions. Every model requires at least one new namespace.
data-types	element	[0..1]	Container for data type definitions
constraints	element	[0..1]	Container for constraints definitions
types	element	[0..1]	Container for type definitions
aspects	element	[0..1]	Container for aspect definitions

Example:

```

<?xml version="1.0" encoding="UTF-8"?>
<model name="cm:contentmodel"
  xmlns="http://www.alfresco.org/model/dictionary/1.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

  <description>Alfresco Content Domain Model</description>
  <author>Alfresco</author>
  <published>2009-06-04</published>
  <version>1.1</version>

  <imports>...</imports>
  <namespaces>...</namespaces>
  <constraints>...</constraints>
  <types>...</types>
  <aspects>...</aspects>
</model>

```



Note that the order in which the elements are specified is important. It will not work to specify them in any other order than in the example. This is because in the meta model they are defined within a `<xsi:sequence>` element. This is true throughout the model schema so be careful to always use the elements in the right order.

model.imports

The `imports` definition is the container for import definitions of other content models referred to by the model being defined:

Name	Type	Multiplicity	Description
import	element	[0..*]	Zero or more import definitions for other content models referred to by this content model.

The `import` definition:

Name	Type	Multiplicity	Description
uri	attribute (string)	required	The unique URI for the content model that should be imported into this model.
prefix	attribute (string)	required	The prefix or short code by which the imported content model will be known in this model. This prefix does not have to be the same as the one defined in the original model, it can be remapped in this model. So, if we are importing the <code>cm</code> namespace, we could actually call it something else like <code>cm2</code>.

Example:

```
<imports>
  <import uri="http://www.alfresco.org/model/dictionary/1.0"
    prefix="d"/>
  <import uri="http://www.alfresco.org/model/system/1.0"
    prefix="sys"/>
</imports>
```

This example imports the data dictionary model with all the data-type definitions and it also imports the system model with types such as `base`, `descriptor`, `container`, `reference` and aspects such as `referenceable` and `temporary`.

model.namespaces

The `namespaces` definition is the container for all new custom namespace definitions that will be used by this model:

Name	Type	Multiplicity	Description
namespace	element	[0..*]	Zero or more import definitions for other content models referred to by this content model

The `namespace` definition:

Name	Type	Multiplicity	Description
uri	attribute (string)	required	The unique URI for the new namespace that will be used by this new content model
prefix	attribute (string)	required	The prefix or short code by which the namespace will be referred to

Example:

```
<namespaces>
  <namespace uri="http://www.alfresco.org/model/content/1.0"
    prefix="cm"/>
</namespaces>
```

This example defines the namespace for the new content model to `cm`. All definitions done in the model will be prefixed with `cm`. We could also define more than one namespace to be used in the model, such as in the following example:

```
<namespaces>
  <namespace uri="http://www.bestmoney.com/model/content/1.0"
    prefix="bmc" />
  <namespace uri="http://www.bestmoney.com/model/workflow/1.0"
    prefix="bmw" />
</namespaces>
```

Here, we have defined one namespace `bmc` that will be used for document content classification and another namespace `bmw` that will be used exclusively for workflow-related content.

A good idea is to use the following format for the URI:

`http://<company website address>/model/[content|workflow|...]/1.0`

Where `content` is for the main domain content model and `workflow` denotes a model that has to do with only workflow definitions.

model.data-types

The `data-types` definition is the container for all new data type definitions to be used by this new content model:

Name	Type	Multiplicity	Description
data-type	element	[0..*]	Zero or more data type definitions to be used by this content model

The data-type definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of the data type
title	element (string)	[0..1]	The title of the data type
description	element (string)	[0..1]	The description of the data type
analyser-class	element	[0..1]	The analyzer class that will build token streams for Lucene. These classes represent a policy for extracting index terms from text.
java-class	element	[0..1]	The Java class that will represent this data type.

Example:

```
<data-types>
  <data-type name="d:text">
    <analyser-class> org.alfresco.repo.search.impl.lucene.analysis.
    AlfrescoStandardAnalyser
    </analyser-class>
    <java-class>java.lang.String</java-class>
  </data-type>

  <data-type name="d:int">
    <analyser-class> org.alfresco.repo.search.impl.lucene.analysis.
    IntegerAnalyser
    </analyser-class>
    <java-class>java.lang.Integer</java-class>
  </data-type>
</data-types>
```

This example shows the definition of two data types `text` and `int` from the out of the box Alfresco Dictionary model `dictionaryModel.xml` located in the `alfresco-enterprise-sdk/lib/server/config/alfresco/model` directory. This model contains most of the data types that we will need when defining new content models. It is not very often that we will have to define new data types.

model.constraints

The `constraints` definition is the container for all new constraints to be used by the model:

Name	Type	Multiplicity	Description
constraint	element	[1..*]	One or more constraints to be used throughout this model or other models

The constraint definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of this constraint
type	attribute (string)	optional	The type of constraint that is being defined. Currently the following types are recognized: LIST = value must match entry in a list REGEXP = value must match regular expression MIN-MAX= value must be numeric within this range LENGTH = value must be string and within min and max length <java class> = the java class that implements the constraint
ref	attribute (string)	optional	Only used when referencing a constraint definition like this from a property definition
parameter	element (string)	[0..*]	One or more constraint values that should be used to restrict the value

The parameter definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of this parameter
value	element (string)	[0..1]	Parameter value
list	element	[0..1]	Container for a list of values

Here, we can use either the `value` element or the `list` element.

The `list` definition:

Name	Type	Multiplicity	Description
value	element (string)	[0..*]	Parameter value

LIST example from the forms development kit content model (`fdk-model.xml`):

```
<constraints>
  <constraint name="fdk:type" type="LIST">
    <parameter name="allowedValues">
      <list>
        <value>Phone</value>
        <value>Audio Visual</value>
        <value>Computer</value>
      </list>
    </parameter>
  </constraint>
```

LENGTH example from the forms development kit content model:

```
<constraint name="fdk:summary" type="LENGTH">
  <parameter name="minLength">
    <value>5</value>
  </parameter>
  <parameter name="maxLength">
    <value>100</value>
  </parameter>
</constraint>
```

MINMAX example from the forms development kit content model:

```
<constraint name="fdk:percentage" type="MINMAX">
  <parameter name="minValue">
    <value>0</value>
  </parameter>
  <parameter name="maxValue">
    <value>100</value>
  </parameter>
</constraint>
```

REGEXP example from the forms development kit content model:

```
<constraint name="fdk:email" type="REGEX">
  <parameter name="expression">
    <value><![CDATA[ [A-Za-z0-9._] +@[A-Za-z0-9.\-] +\.[A-Za-z]{2,4}]]>
  </value>
</parameter>
<parameter name="requiresMatch">
  <value>true</value>
</parameter>
</constraint>
```

Custom Java class constraint from the standard content model (contentModel.xml):

```
<constraint name="cm:userNameConstraint" type=
  "org.alfresco.repo.dictionary.constraint.UserNameConstraint" />
</constraints>
```

model.types

The types definition is the container for all new type definitions:

Name	Type	Multiplicity	Description
type	element	[1..*]	One or more type definitions

The type definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of the type
title	element (string)	[0..1]	The title of the type
description	element (string)	[0..1]	The description of the type
parent	element (string)	[0..1]	Reference to another type that this type extends. Single inheritance is supported.
archive	element (boolean)	[0..1]	true = content with this type should be archived when deleted. This means that it will end up in the Archive Store and it can be restored from the recycle bin. false = content with this type is not archived when deleted and is therefore permanently gone and can never be restored.

Name	Type	Multiplicity	Description
properties	element	[0..1]	Container for all properties that are members of this type class
associations	element	[0..1]	Container for all parent-child and peer associations for this type
overrides	element	[0..1]	Property overrides of super type class properties
mandatory-aspects	element	[0..1]	Mandatory aspects for this type. When content of this type is created these aspects will be created/populated automatically.

model.types.type.properties

The `properties` definition:

Name	Type	Multiplicity	Description
property	element	[0..*]	Zero or more properties

The `property` definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of the property. The name has to be unique between all types and aspects that are defined in the namespace. It is different from object-oriented programming where a member variable of one class can be named the same as in another class.
title	element (string)	[0..1]	The title of the property. Will be used as the property label in the UI in many places, such as in Advanced Search. Note: For proper i18n support use property files.
description	element (string)	[0..1]	The description of the property.
type	element (string)	[1..1]	Mandatory data type reference. This references a <code>data-type</code> definition in the <code>dictionaryModel.xml</code> content model.

Name	Type	Multiplicity	Description
protected	element (boolean)	[0..1]	true = this property cannot be edited after the value has been set. If true then there will be a * before the input field in the UI to indicate that this property is mandatory. false = it is optional and this is the default if this field is not specified.
mandatory	element (boolean)	[0..1]	true = this property has to be specified when this type is created. If true then there will be a * before the input field in the UI to indicate that this property is mandatory. false = it is optional and this is the default if this field is not specified.
multiple	element (boolean)	[0..1]	true = this property can have a list of values. If true then there will be a special UI widget generated to be able to input multiple values for this property. false = only one value can be entered and this is the default if this field is not specified.
default	element (any)	[0..1]	Default value for this property if the user does not specify any value. The UI input field will be pre-populated with this value. false = value should be empty and this is the default if this field is not specified.
index	element	[0..1]	Lucene index configuration
constraints	element	[0..1]	Container for property constraints.



The mandatory aspect can be further configured with a boolean attribute called `enforced`. If set to `true`, content with this type cannot be saved if this property is not specified (this is the default behavior).

This might cause problems for certain non-UI processes uploading content and not being able to set certain properties. Then this attribute can be set to `false`, which means that the mandatory check will be relaxed and if the property value is not provided the content will have the `sys:incomplete` aspect applied.

This way the content can be processed again after being uploaded to fix the missing mandatory properties.

The indexing behavior of each property can be configured. If we do not configure any indexing behavior then the default configuration is:

- To index atomically, which means the indexing is synchronized with the commit of an update content transaction
- The property value is not stored in the index
- The property value is tokenized when it is indexed

This basically means that the default index configuration for properties is as follows:

```
<index enabled="true">
  <atomic>true</atomic>
  <stored>false</stored>
  <tokenised>true</tokenised>
</index>
```

Here is the index definition:

Name	Type	Multiplicity	Description
enabled	attribute (boolean)	required	true = this property will be indexed by Lucene false = this property will not be indexed and hence not searchable.
atomic	element (boolean)	[0..1]	true = if the indexing of this property should be done in the same transaction as for example the document upload. false = the indexing process should be done asynchronously in the background. Indexing of content that requires transformation before being indexed (for example, PDFs) will only obey <code>atomic=true</code> if the transformation takes less time than the value specified for <code>lucene.maxAtomicTransformationTime</code>. Also, properties with data type <code>cm:content</code> are usually indexed in the background.

Name	Type	Multiplicity	Description
stored	element (boolean)	[0..1]	<code>true</code> = the property value is stored in the index and may be obtained via the Lucene low-level query API. <code>false</code> = the property value is not stored in the Lucene index, so cannot be inspected via tools that can read the Lucene index. This can be useful while debugging systems to see exactly what is being indexed, but do not set this to <code>true</code> on production systems.
tokenised	element (string, true,false, both)	[0..1]	<code>true</code> = the property value should be tokenized before indexing. <code>false</code> = the whole property value should be indexed as is, without any tokenization. <code>both</code> = index both the complete value and the tokens.

The constraints definition:

Name	Type	Multiplicity	Description
constraint	element	[1..*]	One or more constraints

The constraint definition:

Name	Type	Multiplicity	Description
name	attribute (string)	optional	Only used when defining constraints
ref	attribute (string)	optional	A reference to the constraint definition
type	attribute (string)	optional	Only used when defining constraints
parameter	element (string)	[0..*]	Only used when defining constraints

model.types.type.associations

The associations definition:

Name	Type	Multiplicity	Description
association	element	[0..*]	List of peer associations
child-association	element	[0..*]	List of child associations

The association definition:

Name	Type	Multiplicity	Description
name	attribute (string)	required	The name of the peer association.
title	element (string)	[0..1]	The title of the peer association.
description	element (string)	[0..1]	The description of the peer association.
source	element	[0..1]	Information about the source node in this peer association, such as what role it is playing. In a peer association, the source node has this type applied (that is the type that defined the association).
target	element	[1..1]	Mandatory information about the target node in this peer association, such as what role it is playing and what class it is of.

The source definition:

Name	Type	Multiplicity	Description
role	element (string)	[0..1]	The role that the source node in this peer association is playing. For example: <code>referencedBy</code> , you can add namespace prefix if you want to: <code>cm:referencedBy</code> However, this is not a reference to a type or an aspect, just a string
mandatory	element (boolean)	[0..1]	<code>true</code> = association is mandatory <code>false</code> = optional association
many	element (boolean)	[0..1]	<code>true</code> = node can be the source in many of these associations <code>false</code> = node can be the source in only one of these associations

The target definition:

Name	Type	Multiplicity	Description
class	element (string)	[1..1]	Mandatory target class (that is type or aspect reference) that we are linking to.
role	element (string)	[0..1]	The role that the target node in this peer association is playing. For example: references, you can add namespace prefix if you want to: cm:references However, this is not a reference to a type or an aspect, just a string.
mandatory	element (boolean)	[0..1]	true = target node has to exist in this association false = target node is optional Note: We can also use the enforced attribute to relax the mandatory aspect
many	element (boolean)	[0..1]	true = node with this association can be linked to many other target nodes false = node with this association can be linked to only one target node

The child-association definition:

Same as the peer association with the following extra properties:

Name	Type	Multiplicity	Description
child-name	element (string)	[0..1]	An optional name of the child association such as for example, "contains"
duplicate	element (boolean)	[0..1]	true = the node names of all target nodes (children) have to be unique false = duplicate target node names are allowed
propagateTimestamps	element (boolean)	[0..1]	true = set parent node cm:modified if any child node cm:modified is changed false = do not propagate changes in children to parent

When setting the *cardinality* for an association, the following example guide can be used:

Source cardinality	Source configuration	Target cardinality	Target configuration
[0..1]	mandatory = false many = false	[0..1]	mandatory = false many = false
[1..1]	mandatory = true many = false	[1..1]	mandatory = true many = false
[0..1]	mandatory = false many = false	[0..*]	mandatory = false many = true
[0..*]	mandatory = false many = true	[1..*]	mandatory = true many = true

Type definition examples

A minimal type definition from the Alfresco Forum Model (`forumModel.xml`):

```
<types>
  <type name="fm:post">
    <parent>cm:content</parent>
  </type>
```

This kind of type definition could be called a content type marker. It tells us that this is a forum post but does not add any properties or associations. We could do a search on only forum posts by selecting to search on content, which has this type applied.

Almost all types that we define that should model domain content will extend the Alfresco base content type `cm:content`. If you wanted to create a type that is not visible in the UI then you can extend the `cm:cobject` instead. Both these types can be found in the `contentModel.xml` file.

To define our own domain folder type we can extend the `cm:folder` type, this example is from the Forum Model:

```
<type name="fm:forums">
  <parent>cm:folder</parent>
</type>
```

This domain type will mark a node as a folder for forums. All types that should be extensions to folders should extend the Alfresco base folder type `cm:folder`.

A simple type definition with some properties from our Best Money project looks like this:

```
<type name="bmc:document">
  <title>Best Money Document</title>
  <parent>cm:content</parent>
  <properties>
    <property name="bmc:department">
      <title>The Best Money Department that created the
        document</title>
      <type>d:text</type>
    </property>
    <property name="bmc:language">
      <title>Language that the document is written in</title>
      <type>d:text</type>
    </property>
  </properties>
</type>
```

Here, we can see two text properties being defined by setting the type to `d:text` (the `d` namespace will have to be imported and it refers to the namespace defined in the `dictionaryModel.xml`).

A more complex type definition with property constraints, index definition, and mandatory aspect is as follows:

```
<type name="cm:cobject">
  <title>Object</title>
  <parent>sys:base</parent>
  <properties>
    <property name="cm:name">
      <title>Name</title>
      <type>d:text</type>
      <mandatory enforced="true">true</mandatory>
      <index enabled="true">
        <atomic>true</atomic>
        <stored>false</stored>
        <tokenised>both</tokenised>
      </index>
      <constraints>
        <constraint ref="cm:filename" />
      </constraints>
    </property>
  </properties>
```

```

    <mandatory-aspects>
      <aspect>cm:auditable</aspect>
    </mandatory-aspects>
  </type>

```

When we define a new domain content type it will also extend `cm:cmbobject` as `cm:content` extends it. This object type defines the `cm:name` property as mandatory and it is enforced as mandatory. So no documents (that is content) can enter the repository without having at least a name, no matter from what protocol or interface the content is uploaded.

The `cm:name` property is constrained by the `cm:filename` constraint, which will make sure the filename is Windows compliant. So you could not use, for example, a `"/` in the filename. The name property is also defined to be indexed, both the complete name and the tokenized name. The indexing will be done within the same transaction as the content upload.

An aspect `cm:auditable` has been set as mandatory for this type and will always be set on content with this type applied.

Let's also look at an example with a type that has an association definition (from the `contentModel.xml`):

```

<type name="cm:folder">
  <title>Folder</title>
  <parent>cm:cmbobject</parent>
  <archive>true</archive>
  <associations>
    <child-association name="cm:contains">
      <source>
        <mandatory>false</mandatory>
        <many>true</many>
      </source>
      <target>
        <class>sys:base</class>
        <mandatory>false</mandatory>
        <many>true</many>
      </target>
      <duplicate>false</duplicate>
      <propagateTimestamps>true</propagateTimestamps>
    </child-association>
  </associations>
</type>

```


This folder type defines one child association called `cm:contains`. This is the association that controls the folder and documents relationship in the user interfaces. The association is many to many and this means that a folder can contain many documents, or none (as `mandatory` is false) and a document can be contained in many different folders.

The child association does not allow a folder to contain two documents with the same name (that is `duplicate` is false). And when a document in a folder is updated and its `cm:modified` property is updated, the `cm:modified` is also updated for the folder (that is `propagateTimestamps` is true).

model.aspects

The definition of aspects does not differ that much from type definitions. In fact, in the schema they are both based on the same `class` definition. The main difference is that an aspect does not usually have a `parent`, it can extend another aspect, but it is more common that they live by themselves. Aspects are used to implement cross-cutting concerns independently of what type of node it is. A type definition can have zero or more mandatory aspects.

Let's have a look at the `cm:auditable` aspect that is a mandatory aspect for the `cm:cmobject` type that we were looking at:

```
<aspect name="cm:auditable">
  <title>Auditable</title>
  <properties>
    <property name="cm:created">
      <title>Created</title>
      <type>d:datetime</type>
      <protected>true</protected>
      <mandatory enforced="true">true</mandatory>
      <index enabled="true">
        <atomic>true</atomic>
        <stored>false</stored>
        <tokenised>both</tokenised>
      </index>
    </property>
```

This aspect defines five well-known properties that are set on all content in the repository, both folders and documents. The first property `cm:created` is `protected` and this is also true for the other properties. This is important to know as after the content has been created, and these properties set, we cannot change them. The rest of the audit properties look like this:

```
<property name="cm:creator">
  <title>Creator</title>
  <type>d:text</type>
  <protected>true</protected>
  <mandatory enforced="true">true</mandatory>
</property>
<property name="cm:modified">
  <title>Modified</title>
  <type>d:datetime</type>
  <protected>true</protected>
  <mandatory enforced="true">true</mandatory>
  <index enabled="true">
    <atomic>true</atomic>
    <stored>false</stored>
    <tokenised>both</tokenised>
  </index>
</property>
<property name="cm:modifier">
  <title>Modifier</title>
  <type>d:text</type>
  <protected>true</protected>
  <mandatory enforced="true">true</mandatory>
</property>
<property name="cm:accessed">
  <title>Accessed</title>
  <type>d:datetime</type>
  <protected>true</protected>
  <index enabled="true">
    <atomic>true</atomic>
    <stored>false</stored>
    <tokenised>both</tokenised>
  </index>
</property>
</properties>
</aspect>
```

These properties are totally controlled by the core repository logic. If we try to change the `cm:created` date, which is always set to current date and time when content is added to the repository, we would not be successful no matter what we do to try and change it. In fact, right now one of the only ways to create content with a different `cm:created` date is to use an ACP file and import content.

Modeling tips and tricks

This section contains some tips and tricks around content modeling.

Not changing the out of the box models

When looking at the out of the box content models it might sometimes be tempting to just make a small change to one of them to get what we want. Avoid this at all costs as it is going to give you a maintenance problem. Every time a new version of Alfresco is released, and we want to use it, the model has to be updated manually. We have to update it manually, as we do not know if something was added to it in the new version.

Also, when someone not familiar with the project is upgrading the system, they might not realize that we have updated an out of the box model and it might take awhile before they realize why some feature is not working as expected.

Starting small

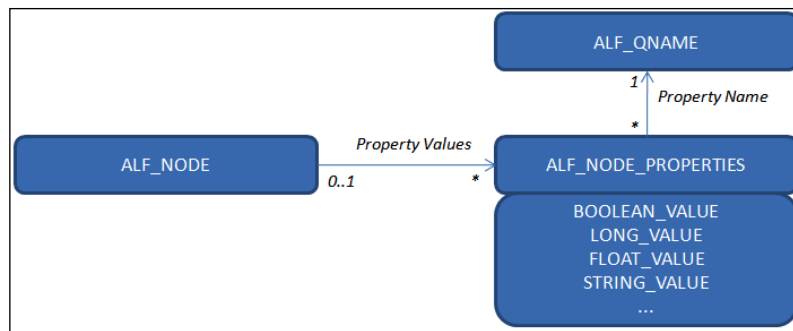
It is often easy to start adding loads of types, aspects, associations, and properties to a new model just to be sure to cover all domain information that the customer might want to associate with a document or a folder. But be careful not to end up with classes and properties that are not actually used. Then why is this important? There are several reasons:

Performance

The more properties we have, the slower it will be to manage content in the repository. This is because of how the properties are stored in the database. Normally, you would have the type or aspect and its properties in the same table as follows:

```
JOB_TYPE {  
    ID INTEGER,  
    NAME TEXT,  
    DESCRIPTION TEXT,  
    . . .  
}
```

And it is then easy to get all properties at once with `SELECT * FROM JOB_TYPE WHERE ID = X`. On the other hand, in Alfresco the main node information is stored in the `ALF_NODE` table, the node property values in the `ALF_NODE_PROPERTIES` table, and the node names in another table called `ALF_QNAME`. The following diagram illustrates:



We can also see that the way the property value is stored is not normalized. For example, a long value is stored in one column and a string value in another. There will be one row per property value. This is all necessary if we want such a flexible and extensive system as Alfresco where properties can be added dynamically.

Manageability

As in all programming and software development it is not advisable to have code and definitions that are not actually used, as this causes confusion for anyone looking at it and trying to understand. As far as possible try and avoid having any type, aspect, association, or property in the definition that is not actually used. Better to add it later on, if needed.

Changeability

The major disadvantage of having more definitions in the model than are used is that as soon as we start using a type or an aspect from the model, we cannot remove it without causing content integrity errors when the system is restarted.

This is because there are going to be database rows created for property names, default values, and so on and if we remove the definitions for these from the model we are breaking the integrity of the data in the repository.

Instead, add properties to types and aspects later on, even if they have been used, as this is handled without problem by Alfresco.

Defining a new custom type for a domain

When the domain is modeled, we will come up with all kinds of domain objects such as Marketing Document, Meeting, Forums, and so on. To define these, we just need to think whether they are containers or contained. In this case, the Marketing Document and the Meeting file are both contained in a folder and the Forums object is a container for Forum objects.

A container is best defined by extending the `cm:folder` container as follows:

```
<types>
  <type name="{namespace:typeName}">
    <title>{description of folder domain type}</title>
    <parent>cm:folder</parent>
    ...
  </type>
</types>
```

If the domain content is a contained file, then the following definition template can be used where we extend the `cm:content` type:

```
<types>
  <type name="{namespace:typeName}">
    <title>{description of file domain type}</title>
    <parent>cm:content</parent>
    ...
  </type>
</types>
```

If the domain content should be stored in the repository but not be visible in any of the user interfaces, we can extend the `cm:cobject` type instead of `cm:content`:

```
<types>
  <type name="{namespace:typeName}">
    <title>{description of file domain type}</title>
    <parent>cm:cobject</parent>
    ...
  </type>
</types>
```

We should also think of what extra domain properties we will define for the new domain type. If we or the client cannot come up with any properties, then we have to ask ourselves the question, what are we going to use the new type for? If the type will be used to narrow down searches then it is a good idea to keep it. It will then act as a kind of type marker and we can select it as a criterion when doing advanced searches.

A marker type might also be useful if we are searching for content programmatically and want to narrow down the searches.

When to use a type and when to use an aspect

A rule of thumb is to look at a description of the domain and write down all nouns, adjectives, and verbs. They would then be converted to types or aspects as follows:

- **Noun** – a type
- **Adjective** – a subtype
- **Verb** – an aspect

Take the following sentence for example:

"Best Money Documents in this folder are finance whitepapers that should be versioned and that should be allowed to be published on the Web, where there should also be a marker for if the document was e-mailed into us or not."

From this sentence, we can extract the following types and aspects:

- **Document** – type
- **Folder** – type
- **Whitepaper** – type
- **Finance** (whitepaper) – subtype of Whitepaper
- **Versioned** – aspect
- **Published** – aspect
- **E-mailed** – aspect

For the Document object we could just settle for using the out of the box type `cm:content`. However, it is often a good idea to define an enterprise-wide root document type. This is what we have done with the Best Money Document Type that we designed in the last chapter. For the Folder object, it usually makes sense to just use the Alfresco `cm:folder` type.

At first, it seems that defining a new type for Whitepaper that extends the Best Money Document Type would be alright. This is not always the case as several different document types might be classified as whitepapers. For example, we might also have a marketing document that is a whitepaper, and it might already extend the marketing document type – and Multiple Inheritance is not supported.

So when defining a type, think about if it can be applied in any way as a cross-cutting concern; can it be applicable to several different types of domain objects? If this is the case, then change it to be defined as an aspect.

The versioned, published, and e-mailed characteristics are perfect as aspects as they can be applied all over the repository, on many different types of nodes.

Design patterns

When new content models are designed there are a few patterns that come back over and over again. A couple of them are described here.

Domain document root type

This is one of the most used patterns in document management solutions as the main use of the CMS repository is to manage and classify documents.

Problem

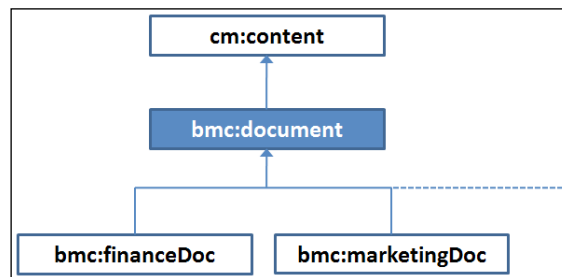
We want to be able to classify generic files that cannot be classified under any specific subtype. We want to be able to search through all documents in the enterprise via a root type. We want a way to add generic properties for all subtypes, usually non-UI related properties.

Solution

Implement a generic base type that all other document types extend.

Diagram

The following figure shows an example of an inheritance hierarchy with a generic enterprise-wide base type for Best Money called `bmc:document`:



Definition example

The following content model definition example code shows how this content type hierarchy can be built up:

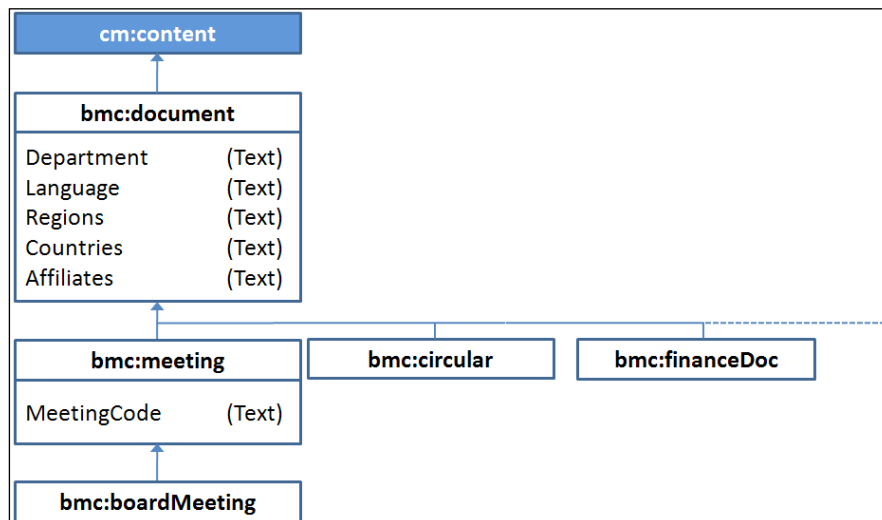
```
<types>
  <type name="bmc:document">
    <parent>cm:content</parent>
  </type>
  <type name="bmc:financeDoc">
    <parent>bmc:document</parent>
  </type>
  <type name="bmc:marketingDoc">
    <parent>bmc:document</parent>
  </type>
</types>
```

Composite type

This pattern is useful when we have a document type hierarchy where the base type has a number of properties that should be displayed in the UI for each subtype. It is also useful when we want to add some properties at different levels in the hierarchy.

Problem

Let's say we have defined a document type hierarchy as follows:

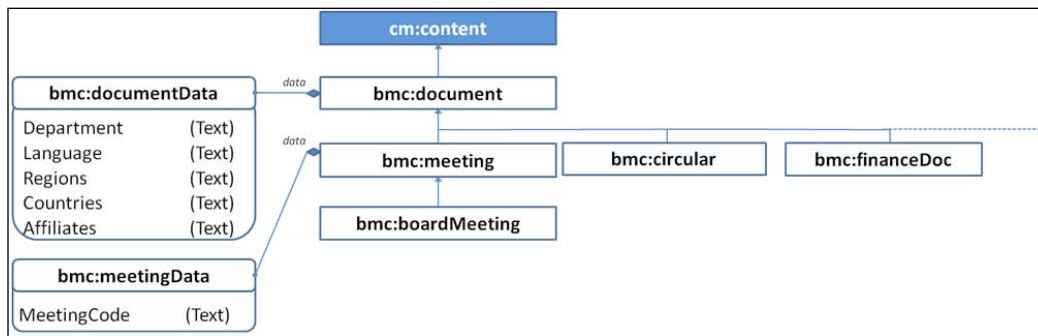


This looks quite alright at first until we start defining property sheets to display the properties. Then we discover that we have to define property sheets for every single type, otherwise the properties will not be displayed. And we do not want to break the *Don't Repeat Yourself (DRY)* rule and create a maintenance problem and a more error prone solution.

Solution

Use the *Composite* design pattern to build the types using aspects.

The following figure shows the document hierarchy re-designed using aspects to keep the property data:



Definition example

The composite type example can be defined like this:

```
<types>
  <type name="bmc:document">
    <parent>cm:content</parent>
    <mandatory-aspects>
      <aspect>bmc:documentData</aspect>
    </mandatory-aspects>
  </type>

  <type name="bmc:circular">
    <parent>bmc:document</parent>
  </type>

  <type name="bmc:meeting">
    <parent>bmc:document</parent>
    <mandatory-aspects>
```

```

    <aspect>bmc:meetingData</aspect>
  </mandatory-aspects>
</type>

<type name="bmc:boardMeeting">
  <parent>bmc:meeting</parent>
</type>

<type name="bmc:financeDoc">
  <parent>bmc:document</parent>
</type>
</types>

<aspects>
  <aspect name="bmc:documentData">
    <properties>
      <property name="bmc:department">
        <type>d:text</type>
      </property>
      <property name="bmc:language">
        <type>d:text</type>
      </property>
      <property name="bmc:countries">
        <type>d:text</type>
      </property>
    </properties>
  </aspect>

  <aspect name="bmc:meetingData">
    <properties>
      <property name="bmc:meetingCode">
        <type>d:text</type>
      </property>
    </properties>
  </aspect>
</aspects>

```

It is then easy to define property sheets for the aspects. When a type is displayed, no matter which one, the custom properties will always be displayed without having any property sheets for the types.

Then we will only need property sheets for the aspects:

```
<config evaluator="aspect-name" condition="bmc:documentData">
  <property-sheet>...</property-sheet>
</config>

<config evaluator="aspect-name" condition="bmc:meetingData">
  <property-sheet>...</property-sheet>
</config>
```

And no type property sheet, like the following, is needed for any of the types:

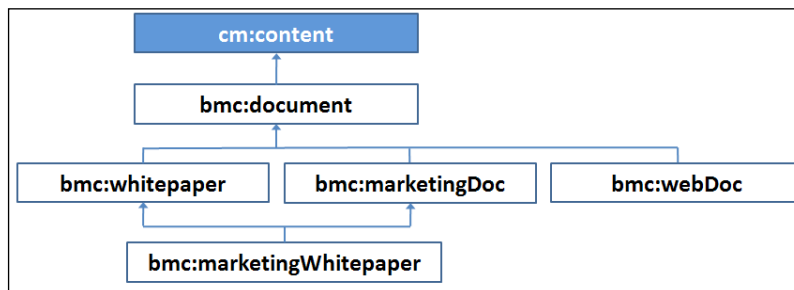
```
<config evaluator="node-type" condition="bmc:document">
  <property-sheet>...</property-sheet>
</config>
```

Multiple types inheritance

Use this pattern when you see the need to extend two or more types.

Problem

Sometimes, during the domain design, we can end up with a type hierarchy that seems to really need multiple inheritances. Take the following example:



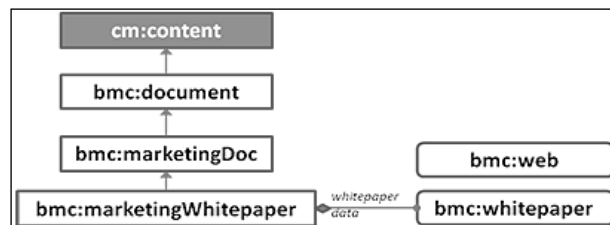
Here, we have defined a marketing document type and a whitepaper document type. Then we extend these two types to create a marketing whitepaper type. This is a quite common scenario but we need to think about each type here and see if it should really be defined as a type. We also have the web document type that denotes a document published on the Web.

When defining a domain type, think if it really represents a domain object. For example, the marketing document can definitely be said to be a domain object that has do with marketing. However, if we look at the whitepaper document type, it does not really represent a domain object, it just classifies a document as a whitepaper. The same goes for the web document that just classifies a document as published on the website.

Solution

Use aspects instead of types when the object is not a domain object and then use the *Composite* design pattern to build the domain types. We can compose a type from as many aspects as we like (similar to multiple inheritance).

The following figure shows the document hierarchy re-designed using aspects for non-domain objects:



We might not even want to have a marketing whitepaper type, but instead just apply the whitepaper and web aspects as necessary.

Definition example

See the previous Composite Type pattern for an example of how to define this.

Configuration object

Use this pattern when you need to manage some configuration data that is not going to be directly displayed in the user interface.

Problem

For example, let's say we have a wizard or an action that creates folders based on data that the user inputs and based on what has previously been created. In this case, we need to store information about what was previously created. How do we do that without having to create new database tables?

Solution

Define a type that is not visible in the user interface and store it under the store root. The type should extend the `sys:base` type as this is a low-level type that has nothing to do with content.

Definition example

This example defines a marketing campaign configuration type that holds the next available sequence number for campaigns:

```
<type name="bmc:marketingCampaignConfig">
  <title>Campaign Config node</title>
  <parent>sys:base</parent>
  <properties>
    <property name="bmc:nextNumber">
      <title>Next Campaign Number</title>
      <type>d:long</type>
    </property>
  </properties>
</type>
```

Code example

To use the example configuration type, we can use the following code. First define some QNames for the model definitions:

```
public static final String CONTENT_NAMESPACE_URI =
    "http://www.bestmoney.com/model/content/1.0";
public static final QName CONTENT_MODEL_NAME =
    QName.createQName(CONTENT_NAMESPACE_URI, "contentModel");
public static final QName TYPE_CAMPAIGN_CONFIG =
    QName.createQName(CONTENT_NAMESPACE_URI,
        "marketingCampaignConfig");
public static final QName PROP_NEXT_CAMPAIGN_NUMBER =
    QName.createQName(CONTENT_NAMESPACE_URI, "nextNumber");
```

Get the next available sequence number to use for the campaign:

```
private long getNextCampaignSequenceNumber(StoreRef storeRef) {
    NodeRef campaignConfigNode = getCampaignConfigNode(storeRef);
    if (campaignConfigNode == null) {
        throw new IllegalArgumentException(
            "Campaign Config not found");
    }
}
```

```
// Get the next available sequence number for campaign
QName propQName = PROP_NEXT_CAMPAIGN_NUMBER;
long nextNumber = (Long) getNodeService().getProperty(
    campaignConfigNode, propQName);

// Update the config node with the next
// available sequence number
getNodeService().setProperty(
    campaignConfigNode, propQName, nextNumber + 1);

return nextNumber;
}
```

Here is the sample code for how to get the configuration node reference:

```
private NodeRef getCampaignConfigNode(StoreRef storeRef) {
    StringBuilder luceneQuery = new StringBuilder(100);
    luceneQuery.append("+PATH:\"/*\" +TYPE:\"");
    luceneQuery.append(TYPE_CAMPAIGN_CONFIG.toString());
    luceneQuery.append("\"");

    // Search for the campaign config node ref
    List<NodeRef> nodes = search(storeRef, luceneQuery);
    if (nodes.isEmpty()) {
        throw new RuntimeException(
            "Could not find campaign config node");
    }

    if (nodes.size() > 1) {
        logger.error("Found more than one campaign config node under store
            " + storeRef);
    }

    return nodes.get(0);
}
```

It uses a Lucene query to search through the repository on the top node level and then matches with any node that has the `bm:marketingCampaignConfig` type set. It then picks the first one of these.

And finally, here is the patch class that creates the node if it does not exist when we start Alfresco:

```
public class CreateCampaignConfigNodePatch extends AbstractPatch {
    private static final String MSG_SUCCESS =
        "patch.createCampaignConfigNodePatch.result";
}
```

```
private ImporterBootstrap m_importerBootstrap;
private PermissionService m_permissionService;

public void setImporterBootstrap(ImporterBootstrap
    importerBootstrap) {
    m_importerBootstrap = importerBootstrap;
}

public void setPermissionService(PermissionService
    permissionService) {
    m_permissionService = permissionService;
}
```

The patch class extends the AbstractPatch class to hook into the patch functionality of Alfresco and then we set up some setters, as we will need the ImporterBootstrap service and the PermissionService. Then we implement the applyInternal method, which is the method that Alfresco calls to run the patch:

```
@Override
protected String applyInternal() throws Exception {
    StoreRef storeRef = m_importerBootstrap.getStoreRef();
    if (storeRef == null) {
        throw new PatchException(
            "Bootstrap store has not been set");
    }

    NodeRef configNodeRef = createCampaignConfigNode(
        storeRef);
    if (configNodeRef == null) {
        throw new PatchException(
            "Campaign Config node could not be created");
    }

    // Set EDITOR Role permission for everyone
    // so they can update the properties of this node
    m_permissionService.setPermission(configNodeRef,
        PermissionService.ALL_AUTHORITIES,
        PermissionService.EDITOR, true);

    return I18NUtil.getMessage(MSG_SUCCESS);
}
```

The method starts off by getting a store reference to the store in which we will save the campaign configuration node. It then creates the configuration node by calling the `createCampaignConfigNode` method that we will implement next. The last thing the `applyInternal` method does is to set the permissions for the node to `EDITOR` for everyone, so they can update the configuration that the node holds. Next, we implement the method that creates the configuration node:

```
private NodeRef createCampaignConfigNode(
    StoreRef storeRef) {
    Map<QName, Serializable> nodeProperties =
        new HashMap<QName, Serializable>();
    nodeProperties.put (PROP_NEXT_CAMPAIGN_NUMBER, 1L);

    ChildAssociationRef childRef = nodeService.createNode(
        nodeService.getRootNode (storeRef),
        ContentModel.ASSOC_CHILDREN,
        QName.createQName (CONTENT_NAMESPACE_URI,
            QName.createValidLocalName ("campaignconfignode")),
        TYPE_CAMPAIGN_CONFIG,
        nodeProperties);

    if (childRef == null || childRef.getChildRef() == null) {
        throw new RuntimeException(
            "Could not create Campaign Config Node");
    }

    return childRef.getChildRef();
}
```

Finally, register the patch:

```
<bean id="patch.createCampaignConfigNodePatch"
    class="com.bestmoney.marketing.bootstrap.
        CreateCampaignConfigNodePatch" parent="basePatch">
    <property name="id">
        <value>patch.createCampaignConfigNodePatch</value>
    </property>
    <property name="description">
        <value>patch.createCampaignConfigNodePatch.description</value>
    </property>
    <property name="fixesFromSchema">
        <value>0</value></property>
    <property name="fixesToSchema">
        <value>${version.schema}</value></property>
    <property name="targetSchema">
```



```
        <value>10000</value></property>
    <property name="importerBootstrap">
        <ref bean="spacesBootstrap"/></property>
    <property name="permissionService">
        <ref bean="permissionService"/></property>
    <property name="nodeService">
        <ref bean="nodeService"/></property>
</bean>
```

When registering the patch, it is good to set the `fixesFromSchema` to 0 and `fixesToSchema` to `version.schema`, which is Alfresco's current version number, so we can be sure that the patch will be executed. The `targetSchema` property should be set to the schema version that this patch attempts to take the existing schema to. This is for informational purposes only, acting as an indicator of intention rather than having any specific effect.

Defining a new custom content model

In the previous chapter, we designed the folder specifications needed for the Best Money project. During this design process, we also extracted the properties that were needed for the different types. So, we have the required input to be able to create the custom content model for Best Money.

The model definition

When defining a new model, we always start by specifying the model name and what namespace should be used for the model and what other namespaces we are going to need (to do this in the `content-model.xml` file located in the `chapter_7_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\model` directory):

```
<?xml version="1.0" encoding="UTF-8"?>
<model name="bmc:contentModel"
  xmlns="http://www.alfresco.org/model/dictionary/1.0">

  <description>Best Money Content Model</description>
  <author>Martin Bergljung</author>
  <version>1.0</version>

  <imports>
    <import uri=
      "http://www.alfresco.org/model/dictionary/1.0"
      prefix="d" />
```

```

    <import uri=
      "http://www.alfresco.org/model/content/1.0"
      prefix="cm" />
  </imports>

  <namespaces>
    <namespace uri=
      "http://www.bestmoney.com/model/content/1.0"
      prefix="bmc" />
  </namespaces>

```

The namespace with prefix `d` is needed to get to the data types we are going to use in our type and aspect definitions and the `cm` namespace contains the base content type `cm:content` that we will base our new domain content types on.

Next thing we define are the constraints that will be used by property definitions. The folder specification has defined a list of languages and a list of departments that can be set up as constraints:

```

<constraints>
  <constraint name="bmc:language_options" type="LIST">
    <parameter name="allowedValues">
      <list>
        <value></value> <!--Empty value is valid -->
        <value>En</value><!-- English -->
        <value>Fr</value><!-- French -->
        <value>Ge</value><!-- German -->
        <value>Sp</value><!-- Spanish -->
        <value>Sw</value><!-- Swedish -->
        <value>Ru</value><!-- Russian -->
        <value>Jp</value><!-- Japanese -->
        <value>Po</value><!-- Portuguese -->
        <value>Ar</value><!-- Arabic -->
        <value>Ch</value><!-- Chinese -->
        <value>Ta</value><!-- Tagalog -->
        <value>In</value><!-- Indonesian -->
        <value>Tu</value><!-- Turkish -->
      </list>
    </parameter>
  </constraint>

```

Make sure to add an empty value in the list as content can be added via an interface that does not allow setting any metadata. It is also important to sort the constraint list values in the order we want to see them in the UI combo box widget. Add a country list constraint:

```
<constraint name="bmc:country_options" type="LIST">
  <parameter name="allowedValues">
    <list>
      <value></value>
      <value>Albania</value>
      <value>Algeria</value>
... other countries we might need go here...
    </list>
  </parameter>
</constraint>
```

It is a good idea to end the constraint name with `_options` so it differs from the property name. Also add a department constraint list according to departments identified in the folder specification:

```
<constraint name="bmc:department_options" type="LIST">
  <parameter name="allowedValues">
    <list>
      <value></value>
      <value>A</value><!-- Audit -->
      <value>HR</value><!-- Human Resources -->
      <value>FM</value><!-- Financial Markets -->
      <value>FS</value><!-- Financial Services -->
      <value>FU</value><!-- Funds Management -->
      <value>IT</value><!-- Information Technology -->
      <value>M</value><!-- Marketing -->
      <value>L</value><!-- Legal -->
    </list>
  </parameter>
</constraint>
</constraints>
```

Next thing we define is the type hierarchy and we will use the *Composite Type* design pattern and keep the properties in an aspect:

```
<types>
  <type name="bmc:document">
    <title>Best Money Document</title>
    <parent>cm:content</parent>
    <mandatory-aspects>
```

```

        <aspect>bmc:documentData</aspect>
    </mandatory-aspects>
</type>
<type name="bmc:circular">
    <title>Best Money Circular Document</title>
    <parent>bmc:document</parent>
</type>
<type name="bmc:meeting">
    <title>Best Money Meeting Document</title>
    <parent>bmc:document</parent>
    <mandatory-aspects>
        <aspect>bmc:meetingData</aspect>
    </mandatory-aspects>
</type>
<type name="bmc:financeDoc">
    <title>Best Money Finance Document</title>
    <parent>bmc:document</parent>
</type>
<type name="bmc:marketingDoc">
    <title>Best Money Finance Document</title>
    <parent>bmc:document</parent>
</type>
<type name="bmc:legalDoc">
    <title>Best Money Finance Document</title>
    <parent>bmc:document</parent>
</type>
<type name="bmc:prDoc">
    <title>Best Money Public Relations Doc</title>
    <parent>bmc:document</parent>
</type>
<type name="bmc:itDoc">
    <title>Best Money IT Doc</title>
    <parent>bmc:document</parent>
</type>
</types>

```

Finally, we define the aspects. Let's start with the base document data:

```

<aspects>
    <aspect name="bmc:documentData">
        <title>Best Money Document Data</title>
        <properties>
            <property name="bmc:departments">
                <title>The Best Money department(s) that created the
                    document</title>
            </property>
        </properties>
    </aspect>
</aspects>

```

```
        <type>d:text</type>
        <multiple>true</multiple>
        <constraints>
            <constraint ref="bmc:department_options"/>
        </constraints>
    </property>
    <property name="bmc:language">
        <title>Language that the document is written in</title>
        <type>d:text</type>
        <constraints>
            <constraint ref="bmc:language_options"/>
        </constraints>
    </property>
    <property name="bmc:countries">
        <title>The country/countries relevant to the document</title>
        <type>d:text</type>
        <multiple>true</multiple>
        <constraints>
            <constraint ref="bmc:country_options"/>
        </constraints>
    </property>
</properties>
</aspect>
```

Here we can see several examples of how to define a constraint for a property. Two of the properties also have the `multiple` element specified to `true`, which means that the property can have more than one value specified and in the UI we will see a widget that can be used to enter multiple values for the property. Define the meeting metadata aspect:

```
<aspect name="bmc:meetingData">
    <title>Best Money Meeting Document Data</title>
    <properties>
        <property name="bmc:meetingCode">
            <title>Unique Meeting Code</title>
            <type>d:text</type>
            <mandatory>true</mandatory>
        </property>
    </properties>
</aspect>
```

The meeting code property has the `mandatory` element specified to `true`. So, whenever this aspect is applied to content it has to be set or the content upload will not work.

The last aspect has to do with making a folder reviewable:

```
<aspect name="bmc:reviewable">
  <title>Reviewable Folder</title>
  <properties>
    <property name=" bmc:reviewPeriod">
      <title>The number of years until documents in this folder
        should be reviewed</title>
      <type>d:int</type>
      <mandatory>true</mandatory>
      <default>5</default>
    </property>
    <property name=" bmc:includeSubFolders">
      <title>Should sub folders also be affected by this review
        period</title>
      <type>d:boolean</type>
      <mandatory>true</mandatory>
      <default>true</default>
    </property>
  </properties>
</aspect>
</aspects>
</model>
```

This aspect specification also has some properties that are specified as mandatory. However, they will have their values set to default values if not specified, as the default element has been defined for both properties.

Registering the model with the repository

For Alfresco to recognize our new model, we need to register it with the repository. Open up the `bootstrap-context.xml` file located in the `chapter_7_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\context` directory and make sure the following Spring bean is present:

```
<bean id="com.bestmoney.dictionaryBootstrap"
  parent="dictionaryModelBootstrap"
  depends-on="dictionaryBootstrap">
  <property name="models">
    <list>
      <value>alfresco/module/com_bestmoney_module_cms/model/content-
        model.xml</value>
    </list>
  </property>
</bean>
```

If we start Alfresco now, we will not actually be able to use the new model for much as its types and aspects are not visible in any wizards.

Configuring property sheets for UI display

Any custom type or aspect that we define needs extra configuration for the properties and associations to be displayed in the UI.

Alfresco Explorer

The Alfresco Explorer user interface client is customized by updating the `web-client-config-custom.xml` file located in the `chapter_9_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\context` directory. Any strings that should be localized are added to the `webclient.properties` file in the same directory.

Displaying properties in content details pages

For the extra Best Money custom aspect properties to be displayed in the Details pages for documents, we need to add the following property sheet configuration:

```
<config evaluator="aspect-name" condition="bmc:documentData">
  <property-sheet>
    <separator name="sep-1" display-label-id="bmDocumentDataHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmc:departments"
      display-label-id="departments"/>
    <show-property name="bmc:language"
      display-label-id="language"/>
    <show-property name="bmc:countries"
      display-label-id="countries"/>
  </property-sheet>
</config>
<config evaluator="aspect-name" condition="bmc:meetingData">
  <property-sheet>
    <separator name="sep-1" display-label-id="bmMeetingDataHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmc:meetingCode"
      display-label-id="meetingCode"/>
  </property-sheet>
</config>
<config evaluator="aspect-name" condition="bmc:reviewable">
  <property-sheet>
    <separator name="sep-1" display-label-id="reviewableHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmc:reviewPeriod"
      display-label-id="reviewPeriod"/>
  </property-sheet>
</config>
```

```

        <show-property name="bmc:includeSubFolders"
            display-label-id="includeSubFolders"/>
    </property-sheet>
</config>

```

Localize the UI Labels as follows:

```

# bmc:document_data
#
bmDocumentDataHeader=Best Money Document Data
departments=Departments
language=Language
countries=Countries

# bmc:meeting_data
#
bmMeetingDataHeader=Best Money Meeting Data
meetingCode=Meeting Code

#bmc:reviewable
#
reviewableHeader=Reviewable Content
reviewPeriod=Review Period
includeSubFolders=Include Sub-folders?

```

These properties are referred to from the property sheets with the `display-label-id` attribute.

Displaying types in add and create content wizards

To get the new types to show up when we add content, the following configuration needs to be added:

```

<config evaluator="string-compare" condition="Content Wizards">
    <content-types>
        <type name="bmc:document"/>
        <type name="bmc:meeting"/>
        <type name="bmc:circular"/>
        <type name="bmc:financeDoc"/>
        <type name="bmc:marketingDoc"/>
        <type name="bmc:legalDoc"/>
        <type name="bmc:itDoc"/>
        <type name="bmc:prDoc"/>
    </content-types>
</config>

```


Displaying types and aspects in rules wizards

To get the new types and aspects to show up in Rules Wizards, we need to do some more configurations:

```
<config evaluator="string-compare" condition="Action Wizards">
  <aspects>
    <aspect name="bmc:documentData"/>
    <aspect name="bmc:meetingData"/>
    <aspect name="bmc:reviewable"/>
  </aspects>

  <subtypes>
    <type name="bmc:document"/>
    <type name="bmc:meeting"/>
    <type name="bmc:circular"/>
    <type name="bmc:financeDoc"/>
    <type name="bmc:marketingDoc"/>
    <type name="bmc:legalDoc"/>
    <type name="bmc:itDoc"/>
    <type name="bmc:prDoc"/>
  </subtypes>

  <specialise-types>
    <type name="bmc:document"/>
    <type name="bmc:meeting"/>
    <type name="bmc:circular"/>
    <type name="bmc:financeDoc"/>
    <type name="bmc:marketingDoc"/>
    <type name="bmc:legalDoc"/>
    <type name="bmc:itDoc"/>
    <type name="bmc:prDoc"/>
  </specialise-types>
</config>
```

Displaying properties in advanced search

If properties from custom aspects or types should be searchable via the Advanced Search screen, then we need to add the following configuration:

```
<config evaluator="string-compare" condition="Advanced Search">
  <advanced-search>
    <content-types>
      <type name="bmc:document"/>
      <type name="bmc:meeting"/>
    </content-types>
  </advanced-search>
</config>
```

```

        <type name="bmc:circular"/>
        <type name="bmc:financeDoc"/>
        <type name="bmc:marketingDoc"/>
        <type name="bmc:legalDoc"/>
        <type name="bmc:itDoc"/>
        <type name="bmc:prDoc"/>
    </content-types>
    <custom-properties>
        <meta-data aspect="bmc:documentData"
            property="bmc:departments"/>
        <meta-data aspect="bmc:documentData"
            property="bmc:language"/>
        <meta-data aspect="bmc:documentData"
            property="bmc:countries"/>
        <meta-data aspect="bmc:meetingData"
            property="bmc:meetingCode"/>
    </custom-properties>
</advanced-search>
</config>

```

Registering the property sheets and the resource file

For the new property sheets and the localization property file to be picked up by Alfresco, we need to register them too in `bootstrap-context.xml`:

```

<bean id="com.bestmoney.webclient.configBootstrap"
    class="org.alfresco.web.config.WebClientConfigBootstrap"
    init-method="init">
    <property name="configs">
        <list>
            <value>classpath:alfresco/module/com_bestmoney_module_cms/ui/
                web-client-config-custom.xml</value>
        </list>
    </property>
</bean>

<bean id="com.bestmoney.webclient.properties.webResourceBundles"
    class="org.alfresco.web.app.ResourceBundleBootstrap">
    <property name="resourceBundles">
        <list>
            <value>alfresco.module.com_bestmoney_module_cms.ui.webclient
                </value>
        </list>
    </property>
</bean>

```

Alfresco share

The Alfresco Explorer web client presents forms for data viewing, editing, and creation throughout its user interface (DM and WCM). In Alfresco Explorer, these forms are implemented in multiple ways, from JSF property sheets to declaring an XSD rendered by the XForms engine.

The Alfresco Share web client is being built on the Surf web framework that is designed to enable the development of all kinds of websites. Forms play an important part of the websites and as such Alfresco needs to provide a standard way of implementing them.

The forms architecture in version 3.2 and onwards replaces the monolithic and duplicated approach of the JSF property sheet component and XForms in version 2. The same services will be used for both DM and WCM forms, meaning there will be only one configuration syntax and one set of user interface controls.

Alfresco Share now uses the forms engine for the simple and full edit metadata pages, creating content dialog, and data lists. Some of the underlying forms technology is also used for a majority of the other forms we will find in Share.

Displaying properties in metadata pages

In Alfresco Share, we use the forms engine to display metadata for content. It generates and renders forms dynamically and it relies on configuration to manage all aspects of a form, such as what fields should appear in what order and what user interface control should be used to edit the value of a field.

The default forms configuration is defined in the `form-config.xml` file, which can be found in the `tomcat/webapps/share/WEB-INF/classes/alfresco` directory. This file contains all the default user interface controls and constraint handlers for the Alfresco content model.

Each application then has its own form configuration file that defines the forms for the built-in `cm:content` and `cm:folder` types. For Alfresco Share, this file is named `share-form-config.xml` and can be found in the `tomcat/webapps/share/WEB-INF/classes/alfresco` when deployed.

It is not recommended to change the out of the box files as these modifications will typically be removed when you update the `share.war` file. Each application has its own custom extensions file and for Alfresco Share this file is named `share-config-custom.xml`.

To display custom properties from, for example, the `bmc:meeting` type, we don't have to do anything if we are happy with the default layout. If we look at metadata for a document that has been uploaded to one of the Meetings folders, it will look as follows (a rule applies the `bmc:meeting` type automatically) in the *view* screen:

Metadata		Document Actions
Next >>	m:rendition: doclib webpreview Name: Meeting X1.docx Unique Meeting Code: 101 The country/countries relevant to the document: (None) Language that the document is written in: En Last Accessed Date: (None) Title: Meeting X1.docx Author: mbergjung Modified Date: Sun 05 Dec 2010 19:57:20 The Best Money department(s) that created the document: (None) Created Date: Sun 05 Dec 2010 19:56:22 Description: (None) Creator: admin Modifier: admin Encoding: utf-8 Mimetype: application/vnd.openxmlformats-officedocument.wordprocessingml.document Size: 20226	Download View In Browser Edit Metadata Upload New Version Edit Offline Copy to... Move to... Delete Document Start Workflow Manage Permissions Manage Aspects Change Type
	Share Download File URL http://localhost:8080/share/p Document URL	

Here we can see that the three Best Money document aspect properties `bmc:departments`, `bmc:language`, and `bmc:countries` have been automatically included in the view form and laid out randomly. The `bmc:meetingCode` property that is specific to meeting document type has also been added to the preview. These properties have also got labels with the texts from the property title in the content model.

If we click on **Edit Metadata** for the document, we will see an *edit* screen, which also contains the Best Money properties. They are laid out randomly here also. So, it is most likely that we would want to control how the custom Best Money properties are laid out.

We can do that via configuration in the `share-config-custom.xml` file located in the `chapter_7_Code\bestmoney\alf_extensions\trunk\_share\config\alfresco\web-extensions` directory.

For the view and edit screens, we need to add a `node-type` configuration for each of the content types that we wish to display and fill in the `field-visibility` element with the properties that should be visible and in what order they should be displayed.

It would make sense to define one generic `node-type` configuration for the base `bmc:document` type as follows:

```
<config evaluator="node-type" condition="bmc:document">
  <forms>
    <form>
      <field-visibility>
        <show id="bmc:countries"/>
        <show id="bmc:departments"/>
        <show id="bmc:language"/>
      </field-visibility>
    </form>
  </forms>
</config>
```

This will however, not make any difference for a document with the `bmc:meeting` document type as each `node-type` configuration is only used when the `condition` value matches exactly the type to be displayed. It does not work even if it is a subtype. And in our case, we also need the `bmc:meetingCode` property displayed.

So, we need to add one `node-type` configuration per custom type that we define. Add the following configuration for the Best Money meeting document type:

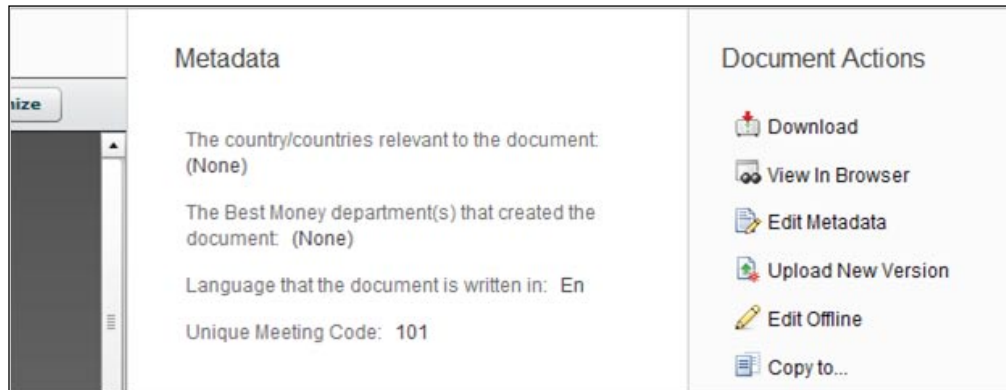
```
<config evaluator="node-type" condition="bmc:meeting">
  <forms>
    <form>
      <field-visibility>
        <show id="bmc:countries"/>
        <show id="bmc:departments"/>
        <show id="bmc:language"/>
        <show id="bmc:meetingCode"/>
      </field-visibility>
    </form>
  </forms>
</config>
```

```

    </field-visibility>
  </form>
</forms>
</config>

```

Now, this works but we only see these four properties:



So what happened to the properties in the `cm:content` type, why are they not displayed? The reason is that we need to configure all properties that should be visible. These are type properties, subtype properties, and included aspect properties.

Update the node-type configuration as follows:

```

<config evaluator="node-type" condition="bmc:meeting">
  <forms>
    <form>
      <field-visibility>
        <!-- cm:content data -->
        <show id="cm:name"/>
        <show id="cm:title" force="true"/>
        <show id="cm:description" force="true"/>
        <show id="mimetype"/>
        <show id="cm:author" force="true"/>
        <show id="size" for-mode="view"/>
        <show id="cm:creator" for-mode="view"/>
        <show id="cm:created" for-mode="view"/>
        <show id="cm:modifier" for-mode="view"/>
        <show id="cm:modified" for-mode="view"/>

        <!-- bmc:meeting data -->
        <show id="bmc:countries"/>
      </field-visibility>
    </form>
  </forms>
</config>

```

```
<show id="bmc:departments"/>
<show id="bmc:language"/>
<show id="bmc:meetingCode"/>
</field-visibility>
</form>
</forms>
</config>
```

Now the Alfresco content model properties will also be visible and displayed before the meeting document properties. However, there is no line or divider between those properties and the meeting document type properties.



To exclude a field from the edit screen we can use `for-mode="view"` attribute and to force a field to be visible even if it is empty we can use the `force="true"` attribute. For example, an aspect might not be applied to a piece of content but we still want to display the empty aspect properties. The `mimetype` and `size` properties are known as *transient* properties, as they don't actually exist as properties in the model, they are formed from the `cm:content` property.

It would be nice if we could have a header just before the meeting properties. This can be done by defining the so-called sets or groups of fields. We do this in a special appearance element where we can set up things like grouping of fields, what controls should be used to display the field, if the field should be read-only, and so on.

For some of the `cm:content` fields, we copy the appearance configuration from the default configuration in `share-form-config.xml` located in the `tomcat\webapps\share\WEB-INF\classes\alfresco` directory:

```
<config evaluator="node-type" condition="bmc:meeting">
  <forms>
    <form>
      <field-visibility>
        ...
      </field-visibility>
      <appearance>
        <field id="cm:title">
          <control template=
            "/org/alfresco/components/form/controls/textfield.ftl" />
        </field>
        <field id="cm:description">
          <control>
```

```
<control-param name="activateLinks">
  true
</control-param>
</control>
</field>
<field id="mimetype">
  <control template=
    "/org/alfresco/components/form/controls/mimetype.ftl" />
</field>
<field id="size">
  <control template=
    "/org/alfresco/components/form/controls/size.ftl" />
</field>
<field id="cm:taggable">
  <control>
    <control-param name="compactMode">
      True
    </control-param>
    <control-param name="params">
      aspect=cm:taggable
    </control-param>
    <control-param name="createNewItemUri">
      /api/tag/workspace/SpacesStore
    </control-param>
    <control-param name="createNewItemIcon">
      Tag
    </control-param>
  </control>
</field>
<field id="cm:categories">
  <control>
    <control-param name="compactMode">
      true
    </control-param>
  </control>
</field>
</appearance>
</form>
</forms>
</config>
```


The previous appearance configuration shows how we can set the controls to be used when displaying different fields and how we can pass in parameters to controls. The default controls for field types are defined in the `forms-config.xml` file located in the `tomcat/webapps/share/WEB-INF/classes/alfresco` directory.

Normally, it is fine with the default configuration and we do not have to specify the control element. This is the case when we specify the appearance for the meeting metadata, which we want in its own group/set:

```
<config evaluator="node-type" condition="bmc:meeting">
  <forms>
    <form>
      <field-visibility>
        ...
      </field-visibility>
      <appearance>
        ...
        <set id="meetingMetadata" appearance="title"
          label-id="meeting.metadata.header" />
        <field id="bmc:countries"
          label-id="meeting.metadata.countries"
          set="meetingMetadata"/>
        <field id="bmc:departments"
          label-id="meeting.metadata.departments"
          set="meetingMetadata"/>
        <field id="bmc:language"
          label-id="meeting.metadata.language"
          set="meetingMetadata"/>
        <field id="bmc:meetingCode"
          label-id="meeting.metadata.meetingCode"
          set="meetingMetadata"/>
      </appearance>
    </form>
  </forms>
</config>
```

We also need to add the `label-ids` for the different fields and the groups/set header, the labels we defined for the Alfresco Explorer client will not be picked up. Add them to the `extension-app.properties` file located in the `bestmoney\alf_extensions\trunk\_share\config\alfresco\messages` directory:

```
meeting.metadata.header=Meeting Document Info
meeting.metadata.countries=Countries
meeting.metadata.departments=Departments
meeting.metadata.language=Language
meeting.metadata.meetingCode=Meeting Code
```

Now, we should see something like this when previewing a meeting document (after stopping Alfresco, running the deploy-share-jar ant target, and starting Alfresco):

Metadata

Name: Meeting X1.docx
Title: Meeting X1.docx
Description: (None)
Mimetype: Microsoft Word 2007
Author: mbergljung
Size: 20 KB
Creator: admin
Created Date: Sun 05 Dec 2010 19:56:22
Modifier: admin
Modified Date: Sun 05 Dec 2010 19:57:20

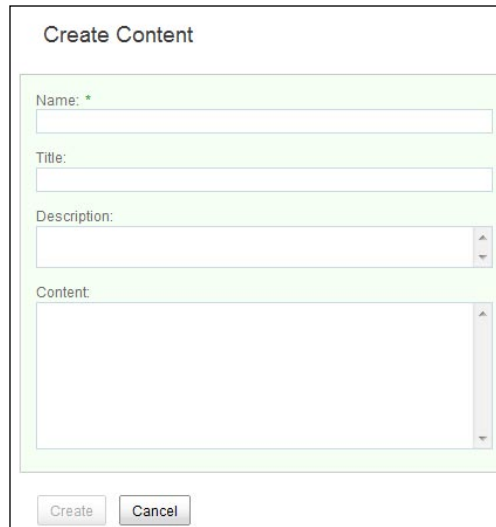
Meeting Document Info
Countries: (None)
Departments: (None)
Language: En
Meeting Code: 101

We can see the meeting metadata clearly separated from the standard document metadata and that the label texts have been picked up correctly. To change the group/set appearance change the appearance attribute of the set element. For example, to surround the meeting data by a bordered panel use `appearance="bordered-panel"`:

Modifier: admin
Modified Date: Sun 05 Dec 2010 19:57:20

Meeting Document Info
Countries: (None)
Departments: (None)
Language: En
Meeting Code: 101

The same appearance configurations also work in the edit screen. If we now look at the **Create Content** screen, we will see that it does not display the extra meeting document properties:



If we would like the user to be able to specify some of these properties when they, for example, select **Create Content...** followed by **Plain Text...**, then we have to add a configuration for the `model-type`. Because the out of the box **Create Content** screen will assume that we are creating `cm:content`, we need to specify the condition as that type:

```
<config evaluator="model-type" condition="cm:content">
  <forms>
    <form>
```

Then we configure the `field-visibility` section where the `cm:content` fields have been copied from the default configuration in `share-form-config.xml`:

```
<field-visibility>
  <show id="cm:name" />
  <show id="cm:title" force="true" />
  <show id="cm:description" force="true" />
  <show id="cm:content" force="true" />
  <show id="mimetype" />
  <show id="app:editInline" force="true" />
```

```

    <show id="bmc:countries" force="true" />
    <show id="bmc:departments" force="true" />
    <show id="bmc:language" force="true" />
    <show id="bmc:meetingCode" force="true"/>
  </field-visibility>

```

Note that the meeting metadata fields have to be specified with `force="true"` as they are not part of `cm:content` and would not be displayed in the screen unless we force it. Next, we copy the appearance configuration for the `cm:content` fields from the default configuration in `share-form-config.xml`:

```

<appearance>
  <field id="cm:title">
    <control template=
      "/org/alfresco/components/form/controls/textfield.ftl" />
  </field>
  <field id="cm:content">
    <control>
      <control-param name="editorAppearance">explorer
    </control-param>
    </control>
  </field>
  <field id="mimetype">
    <control template=
      "/org/alfresco/components/form/controls/hidden.ftl">
      <control-param name="contextProperty">mimeType
    </control-param>
    </control>
  </field>
  <field id="app:editInline">
    <control template=
      "/org/alfresco/components/form/controls/hidden.ftl">
      <control-param name="contextProperty">editInline
    </control-param>
    </control>
  </field>

```

And finally, we add the appearance configuration for the meeting metadata the same way we did for the view and edit screen configuration:

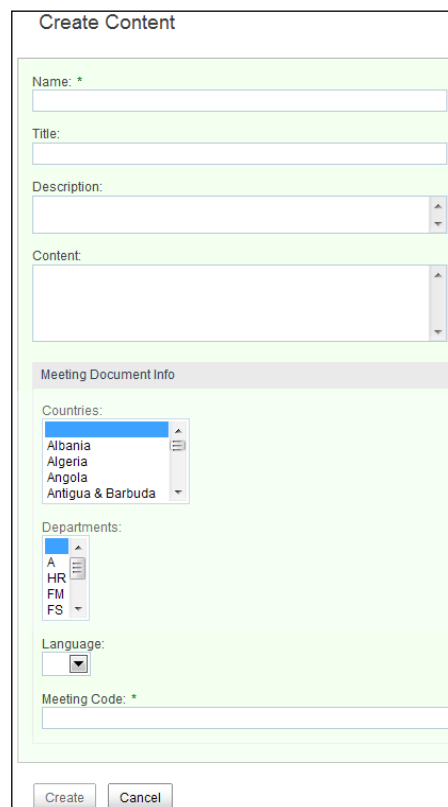
```

<set id="meetingMetadata"
  appearance="bordered-panel"
  label-id="meeting.metadata.header" />
<field id="bmc:countries"
  label-id="meeting.metadata.countries"
  set="meetingMetadata"/>

```

```
<field id="bmc:departments"
      label-id="meeting.metadata.departments"
      set="meetingMetadata"/>
<field id="bmc:language"
      label-id="meeting.metadata.language"
      set="meetingMetadata"/>
<field id="bmc:meetingCode"
      label-id="meeting.metadata.meetingCode"
      set="meetingMetadata"/>
</appearance>
</form>
</forms>
</config>
```

The create screen will now look like the following screenshot:



There are a lot more Alfresco Share forms configurations that we can do, but this is all we have room for in this chapter. There will be more forms configuration in *Chapter 10, Business Process Implementation Solutions: Part 1* when we start implementing workflow.

There is also a lot of information available at the Alfresco Wiki:

http://wiki.alfresco.com/wiki/Forms_Examples

<http://wiki.alfresco.com/wiki/Forms>

Displaying aspects and types

To have the new types and aspects available in Alfresco Share when adding an aspect or changing type, the following configuration needs to be added:

```
<config evaluator="string-compare" condition="DocumentLibrary"
  replace="true">
  <aspects>
    <!-- Aspects that a user can see -->
    <visible>
      <aspect name="cm:generalclassifiable" />
      <aspect name="cm:complianceable" />
      <aspect name="cm:dublincore" />
      <aspect name="cm:effectivity" />
      <aspect name="cm:summarizable" />
      <aspect name="cm:versionable" />
      <aspect name="cm:templatable" />
      <aspect name="cm:emailed" />
      <aspect name="emailserver:aliasable" />
      <aspect name="cm:taggable" />
      <aspect name="app:inlineeditable" />
      <aspect name="bmc:documentData"/>
      <aspect name="bmc:meetingData"/>
      <aspect name="bmc:reviewable"/>
    </visible>

    <!-- Aspects that a user can add. Same as "visible" if left empty
    -->
    <addable>
    </addable>

    <!-- Aspects that a user can remove. Same as "visible" if left
    empty -->
    <removeable>
    </removeable>
  </aspects>

  <types>
    <type name="cm:content">
      <subtype name="bmc:document" />
    </type>
  </types>
```

```
</type>
<type name="bmc:document">
  <subtype name="bmc:meeting"/>
  <subtype name="bmc:circular"/>
  <subtype name="bmc:financeDoc"/>
  <subtype name="bmc:marketingDoc"/>
  <subtype name="bmc:legalDoc"/>
  <subtype name="bmc:itDoc"/>
  <subtype name="bmc:prDoc"/>
</type>
</types>
</config>
```

With this new Alfresco Share configuration, we will be able to select from the new Best Money types when using the **+More...** followed by **Change Type** menu item. The new aspects will be available when selecting the **+More...** followed by **Manage Aspects** menu item.

For the aspects and types to have nicer labels in the user interface, we need to add label texts in the `extension-app.properties` located in the `bestmoney\alf_extensions\trunk\_share\config\alfresco\messages` directory:

```
type.bmc_document=Best Money Document
type.bmc_meeting=Best Money Meeting
type.bmc_circular=Best Money Circular
type.bmc_financeDoc=Best Money Finance Doc
type.bmc_marketingDoc=Best Money Marketing Doc
type.bmc_legalDoc=Best Money Legal Doc
type.bmc_itDoc=Best Money IT Doc
type.bmc_prDoc=Best Money PR Doc
aspect.bmc_documentData=Best Money Document Data
aspect.bmc_meetingData=Best Money Meeting Data
aspect.bmc_reviewable=Best Money Reviewable
```

Displaying properties in advanced search

To be able to search on Best Money Meeting metadata via the **Advanced Search** Screen an extra form configuration is needed in the `model-type` configuration:

```
<config evaluator="model-type" condition="cm:content">
  <forms>
    <form>. . .
  </form>
```

Add it just after the default form configuration:

```
<form id="search">
  <field-visibility>
```

Again, the default field visibility configuration for `cm:content` fields is copied from the `share-form-config.xml` as follows:

```
<show id="cm:name"/>
<show id="cm:title" force="true"/>
<show id="cm:description" force="true"/>
<show id="mimetype"/>
<show id="cm:modified"/>
<show id="cm:modifier"/>
<show id="bmc:countries" force="true"/>
```

Then the meeting metadata fields:

```
<show id="bmc:departments" force="true"/>
<show id="bmc:language" force="true"/>
<show id="bmc:meetingCode" force="true"/>
</field-visibility>
```

And the appearance configuration for the `cm:content` fields. This is also copied from the `share-form-config.xml`:

```
<appearance>
  <field id="mimetype">
    <control template=
      "/org/alfresco/components/form/controls/mimetype.ftl"/>
  </field>
  <field id="cm:modifier">
    <control>
      <control-param name="forceEditable">true</control-param>
    </control>
  </field>
  <field id="cm:modified">
    <control template=
      "/org/alfresco/components/form/controls/daterange.ftl"/>
  </field>
```


And finally the meeting metadata appearance configuration:

```
<set id="meetingMetadata"
    appearance="bordered-panel"
    label-id="meeting.metadata.header"/>
<field id="bmc:countries"
    label-id="meeting.metadata.countries"
    set="meetingMetadata"/>
<field id="bmc:departments"
    label-id="meeting.metadata.departments"
    set="meetingMetadata"/>
<field id="bmc:language"
    label-id="meeting.metadata.language"
    set="meetingMetadata"/>
<field id="bmc:meetingCode"
    label-id="meeting.metadata.meetingCode"
    set="meetingMetadata"/>
</appearance>
</form>
</forms>
</config>
```

Summary

In this chapter, we have learned that a content model is used to classify content stored in a CMS system. A meta model defines the syntax that we can use when defining our new domain-specific content models.

Alfresco's meta model offers a comprehensive syntax that enables us to build content models in an object-oriented way, enabling re-use. There are quite a few content models available out of the box that it is important to check out before we start building our own models.

From the content models available out of the box, the core Alfresco content model defined in the `contentModel.xml` file is probably the most important one to study, as it contains the base types for document content and folder content.

We also went through some useful content model design patterns such as Composite Type and Domain Document Root Type and finally we defined our own model and saw how to configure Alfresco to recognize the custom model in the different UI clients.

In the next chapter, we will look at some solutions for how to do data migration when you have an existing network drive with documents and want to merge them into Alfresco.

8

Document Migration Solutions

The Best Money CMS project is now in full swing and we have the folder structure with business rules designed and implemented and the domain content model created. It is now time to start importing any existing documents into the Alfresco repository. Most companies that implement an ECM system, and Best Money is no exception, will have a substantial amount of files that they want to import, classify, and make searchable in the new CMS system.

The planning and preparation for the document migration actually has to start a lot earlier, as there are a lot of things that need to be prepared:

- Who is going to manage sorting out files that should be migrated?
- What is the strategy and process for the migration?
- What sort of classification should be done during the import?
- What filesystem metadata needs to be preserved during the import?
- Do we need to write any temporary scripts or rules just for the import?

In this chapter, you will learn:

- Different strategies for implementing document migration
- Planning the document migration
- Implementing document migration using CIFS or purpose-built tools

Document migration strategies

The first thing we need to do is to figure out how the document migration is actually going to be done. There are several ways of making this happen. We will discuss a couple of different ways, such as via the CIFS interface and via tools. There are also some general strategies that apply to any migration method.

General migration strategies

There are some common things that need to be done no matter which import method is used, such as setting up a document migration staging area.

Document staging area

The end users need to be able to copy or move documents—that they want to migrate—to a kind of staging area that mirrors the new folder structure that we have set up in Alfresco. The best way to set up the staging area is to copy it from Alfresco via CIFS. When this is done the end users can start copying files to the staging area. However, it is a good idea to train the users in the new folder structure before they start copying documents to it. We should talk to them about folder structure changes, what rules and naming conventions have been set up, the idea behind it, and why it should be followed.

If we do not train the end users in the new folder structure, they will not honor it and the old structure will get mixed up with the new structure via document migration, and this is not something that we want. We did plan and implement the new structure for today's requirements and future requirements and we do not want it broken before we even start using the system.

The end users will typically work with the staging area over some time. It is good if they get a couple of weeks for this. It will take them time to think about what documents they want to migrate and if any re-organization is needed. Some documents might also need to be renamed.

Preserving Modified Date on imported documents

We know that Best Money wants all their modified dates on the files to be preserved during an import, as they have a review process that is dependent on it. This means that we have to use an import method that can preserve the Modified Date on the network drive files when they are merged into the Alfresco repository. The CIFS interface cannot be used for this as it sets Modified Date to Current Date.

There are a couple of methods that can be used to import content into the repository and preserve the Modified Date:

- Create an ACP file via an external tool and then import it
- Custom code the import with the Foundation API and turn off the Audit Aspect before the import
- Use an import tool that also has the possibility to turn off the Audit Aspect

At the time of writing (when I am using Alfresco 3.3.3 Enterprise and Alfresco Community 3.4a) there is no easy way to import files and preserve the Modified Date. When a file is added via Alfresco Explorer, Alfresco Share, FTP, CIFS, Foundation API, REST API, and so on, the Created Date and Modified Date is set to "now", so we lose all the Modified Date data that was set on the files on the network drive.

The Created Date, Creator, Modified Date, Modifier, and Access Date are all so called Audit properties that are automatically managed by Alfresco if a node has the `cm:auditable` aspect applied. If we try and set these properties during an import via one of the APIs, it will not succeed.

Most people want to import files via CIFS or via an external import tool. Alfresco is working towards supporting preserving dates when using both these methods for import. Currently, there is a solution to add files via the Foundation API and preserve the dates, which can be used by custom tools. The Alfresco product itself also needs this functionality in, for example, the Transfer Service Receiver, so the dates can be preserved when it receives files.

The new solution that enables the use of the Foundation API to set Auditable properties manually has been implemented in version 3.3.2 Enterprise and 3.4a Community. To be able to set audit properties do the following:

1. Inject the policy behavior filter in the class that should do the property update:

```
<property name="behaviourFilter" ref="policyBehaviourFilter"/>
```
2. Then in the class, turn off the audit aspect before the update, it has to be inside a new transaction, as in the following example:

```
RetryingTransactionCallback<Object> txnWork = new
    RetryingTransactionCallback<Object>() {
    public Object execute() throws Exception {
        behaviourFilter.disableBehaviour
            (ContentModel.ASPECT_AUDITABLE);
```

3. Then in the same transaction update the Created or Modified Date:

```
nodeService.setProperty(nodeRef,  
    ContentModel.PROP_MODIFIED, someDate);  
    . . .  
}  
};
```



With JDK 6, the Modified Date is the only file data that we can access, so no other file metadata is available via the CIFS interface. If we use JDK 7, there is a new NIO 2 interface that gives access to more metadata. So, if we are implementing an import tool that creates an ACP file, we could use JDK 7 and preserve Created Date, Modified Date, and potentially other metadata as well.

Post migration processing scripts

When the document migration has been completed, we might want to do further processing of the documents such as setting extra metadata. This is specifically needed when documents are imported into Alfresco via the CIFS interface, which does not allow any custom metadata to be set during the import. There might also be situations, such as in the case of Best Money, where a lot of the imported documents have older filenames (that is, following an older naming convention) with important metadata that should be extracted and applied to the new document nodes.

For post migration processing, JavaScript is a convenient tool to use. We can easily define Lucene queries for the nodes we want to process, as the rules have applied domain document types such as Meeting to the imported documents, and we can use regular expressions to match and extract the metadata we want to apply to the nodes.

Search restrictions when running post migration scripts

What we have to think about though, when running these post migration scripts, is that the repository now contains a lot of content, so each query we run might very well return much more than 1,000 rows. And 1,000 rows is the default max limit that a search will return.

To change this to allow for 5,000 rows to be returned, we have to make some changes to the permission check configuration (Alfresco checks the permissions for each node that is being accessed, so the user running the query is not getting back content that he or she should not have access to). Open the `alfresco-global.properties` file located in the `alfresco/tomcat/shared/classes` directory and add the following properties:

```
# The maximum time spent pruning results (was 10000)
system.acl.maxPermissionCheckTimeMillis=100000
# The maximum number of results to perform permission checks against (was
1000)
system.acl.maxPermissionChecks=5000
```

Unwanted Modified Date updates when running scripts

So we have turned off the audit feature during document migration, or made some custom code changes to Alfresco, to get the document's Modified Date to be preserved during import. Then we have turned on auditing again so the system behaves in the way the users expect.

The last thing we want now is for all those preserved modified dates to be set to current date when we update metadata. And this is what will happen if we are not running the post migration scripts with the audit feature turned off. So this is important to think about unless you want to start all over again with the document migration.

Versioning problems when running post migration scripts

Another thing that can cause problems is when we have versioning turned on for documents that we are updating with the post migration scripts. If we see the following error:

```
org.alfresco.service.cmr.version.VersionServiceException: 07120018 The
current implementation of the version service does not support the
creation of branches.
```

by default new versions will be created even when we just update properties/metadata. This can cause errors such as the preceding error and we might not even be able to check-in and check-out the document. To prevent this error from popping up, and turn off versioning during property updates once and for all, we can set the following property at the same time as we set the other domain metadata in the scripts:

```
legacyContentFile.properties["cm:autoVersionOnUpdateProps"] = false;
```

Setting this property to false, effectively turns off versioning during any property/metadata update for the document.

Another thing that can be a problem is, if folders have been set up as versionable by mistake. The most likely reason for this is that we probably forgot to set up the Versioning Rule to only apply to `cm:content` (and not to "All Items"). Folders in the `workspace://SpacesStore` store do not support versioning.



The WCM system comes with an AVM store that supports advanced folder versioning and change sets. Note that the WCM system can also store its data in the Workspace store.

So we need to update the versioning rule to apply to the content and remove the versionable aspect from all folders, which have it applied, before we can update any content in these folders. Here is a script that removes the `cm:versionable` aspect from any folder having it applied:

```
var store = "workspace://SpacesStore";
var query = "PATH:\"/app:company_home/*\" AND TYPE:\"cm:folder\" AND ASPECT:\"cm:versionable\"";
var versionableFolders = search.luceneSearch(store, query);

for each (versionableFolder in versionableFolders) {
    versionableFolder.removeAspect("cm:versionable");
    logger.log("Removed versionable aspect from folder: " +
        versionableFolder.name);
}

logger.log("Removed versionable aspect from " +
    versionableFolders.length + " folders");
```

Post migration script to extract legacy meeting metadata

Best Money has a lot of documents that they are migrating to the Alfresco repository. Many of the documents have filenames following a certain naming convention. This is the case for the meeting documents that are imported. The naming convention for the old imported documents are not exactly the same as the new meeting naming convention, so we have to write the regular expression a little bit differently.

An example of a filename with the new naming convention looks like this: `10En-FM.02_3_annex1.doc` and the same filename with the old naming convention looks like this: `10Eng-FM.02_3_annex1.doc`. The difference is that the old naming convention does not specify a two-character code for language but instead a list that looks like this: `Arabic, Chinese, Eng | eng, F | Fr, G | Ger, Indonesian, Jpn,`

Port,Rus|Russian,Sp,Sw,Tagalog,Turkish. What we are interested in extracting is the language and the department code and the following script will do that with a regular expression:

```
// Regulars Expression Definition
var re = new RegExp("^\\d{2}(Arabic|Chinese|Eng|eng|F|Fr|G|Ger|Indone
sian|Ital|Jpn|Port|Rus|Russian|Sp|Sw|Tagalog|Turkish) - (A|HR|FM|FS|FU|
IT|M|L).*");

var store = "workspace://SpacesStore";
var query = "+PATH:\"/app:company_home/cm:Meetings/*\" +TYPE:\"cm:
content\"";
var legacyContentFiles = search.luceneSearch(store, query);

for each (legacyContentFile in legacyContentFiles) {
    if (re.test(legacyContentFile.name) == true) {
        var language = getLanguageCode(RegExp.$1);
        var department = RegExp.$2;
        logger.log("Extracted and updated metadata (language=" + language
            + ")(department=" + department + ") for file: " +
            legacyContentFile.name);
        if (legacyContentFile.hasAspect("bmc:document_data")) {
            // Set some metadata extracted from file name
            legacyContentFile.properties["bmc:language"] = language;
            legacyContentFile.properties["bmc:department"] = department;

            // Make sure versioning is not enabled for property updates
            legacyContentFile.properties["cm:autoVersionOnUpdateProps"] =
                false;

            legacyContentFile.save();
        } else {
            logger.log("Aspect bmc:document_data is not set for document
                " + legacyContentFile.name);
        }
    } else {
        logger.log("Did NOT extract metadata from file: " +
            legacyContentFile.name);
    }
}

/**
 * Convert from legacy language code to new 2 char language code
 */
```



```
* @param parsedLanguage legacy language code
*/
function getLanguageCode(parsedLanguage) {
  if (parsedLanguage == "Arabic") {
    return "Ar";
  } else if (parsedLanguage == "Chinese") {
    return "Ch";
  } else if (parsedLanguage == "Eng" || parsedLanguage == "eng") {
    return "En";
  } else if (parsedLanguage == "F" || parsedLanguage == "Fr") {
    return "Fr";
  } else if (parsedLanguage == "G" || parsedLanguage == "Ger") {
    return "Ge";
  } else if (parsedLanguage == "Indonesian") {
    return "In";
  } else if (parsedLanguage == "Ital") {
    return "";
  } else if (parsedLanguage == "Jpn") {
    return "Jp";
  } else if (parsedLanguage == "Port") {
    return "Po";
  } else if (parsedLanguage == "Rus" || parsedLanguage == "Russian") {
    return "Ru";
  } else if (parsedLanguage == "Sp") {
    return "Sp";
  } else if (parsedLanguage == "Sw") {
    return "Sw";
  } else if (parsedLanguage == "Tagalog") {
    return "Ta";
  } else if (parsedLanguage == "Turkish") {
    return "Tu";
  } else {
    logger.log("Invalid parsed language code: " + parsedLanguage);
    return "";
  }
}
```

This script can be run from any folder and it will search for all documents under the /Company Home/Meetings folder or any of its subfolders. All the documents that are returned by the search are looped through and matched with the regular expression. The regular expression defines two groups: one for the language code and one for the department. So after a document has been matched with the regular expression it is possible to back-reference the values that were matched in the groups by using `RegExp.$1` and `RegExp.$2`.

When the language code and the department code properties are set, we also set the `cm:autoVersionOnUpdateProps` property, so we do not get any problem with versioning during the update.

Importing documents via CIFS

When the new staging area has been populated by the end users, it is time to do the actual migration. One way to do this is to use the CIFS interface as it is a simple and well-known way of copying files from one drive to another. When copying the files it's a good idea to copy them in batches, selecting one top folder per batch, for example.

Before starting the migration it is also often necessary to turn off rules that check things such as naming conventions. In Best Money's case, there are going to be documents that do not follow the new naming conventions that we have defined. And when the rules throw an error the document migration will halt.



If we do not turn off rules Alfresco might throw errors after different kinds of checks, then we will most likely get "Could not find this item" error in Windows Explorer. This is because the node was not added to the repository after the rule error, and when Windows Explorer tries to make updates to it via CIFS it gets back this message that the file was not found by Alfresco.

Also, note that any error message that is being displayed in the Alfresco Explorer UI because of a rule is not going to be displayed in Windows Explorer in the same way. Most likely we will get the "Could not find this item" message.

When we are ready to start the document migration, it is best to do it first on a Test Box and sort out any problems. Then when we are confident that the document migration works nicely, we can move onto the Production Box.

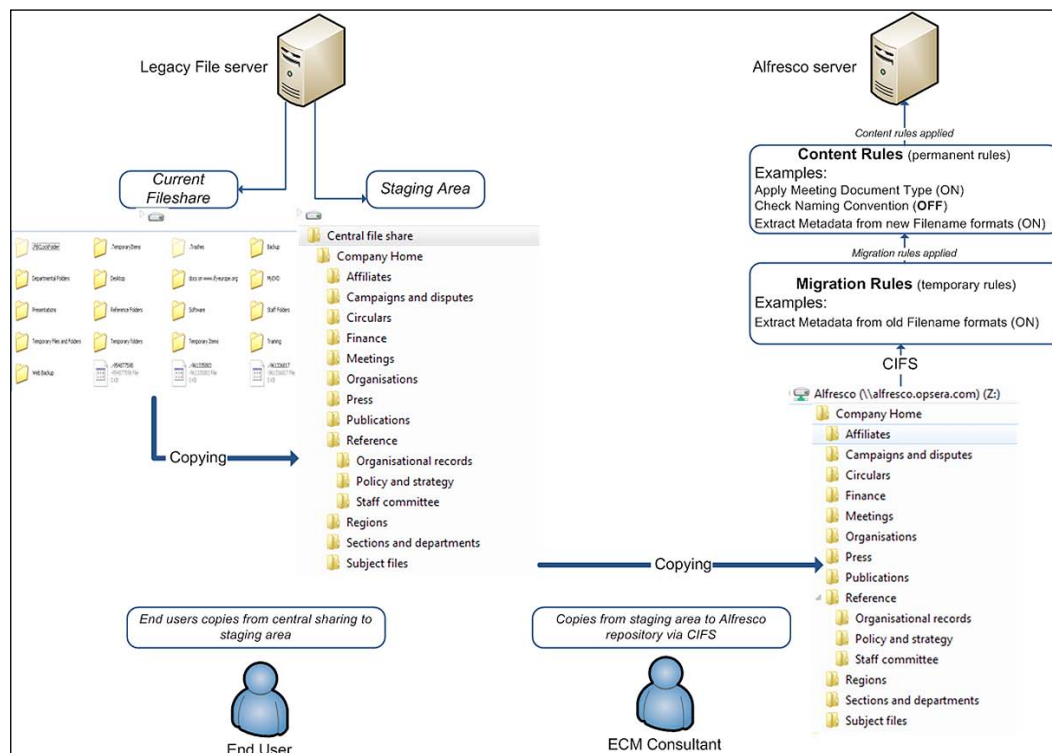


An alternative way is to do the migration on Test and then export the complete Test folder structure including migrated documents to an ACP file. Remember that you might have to split the exported data up into several ACP files if there is a lot of content and the ACP file gets bigger than 4 GB. This is the max ZIP file size that Java handles (you can try using JDK7 as it is supposed to handle larger ZIP files). Then wipe out the folder structure on the Production Box and import the ACP file from the Test Box. That will be faster and easier than doing the import via CIFS. We can export several ACP files if there are a lot of documents.

Make sure to always copy all documents that should be migrated to the local disk where Alfresco is running for best performance. One should also take an ACP backup of the new file structure on the Test Box before doing the migration. This way it will be easy to go back to the initial start state if something goes wrong with the migration.

When using CIFS for document migration, it is not going to be the fastest way of doing an import. The CIFS interface is quite "chatty" and as an example, importing 13,500 documents (9 GB) took around four hours on a Windows 2008 R2 64-bit box, Xeon E5520 2.34 GHz, 4 CPUs, and 4 GB RAM. The documents were then stored on the same disk as Alfresco was running. So if you are looking at significantly more data to import, it might be better to look at another import method such as using a purpose-built tool.

The following picture shows an overview of CIFS-based document migration:



The picture shows the legacy file server that has all documents that should be migrated over to Alfresco. On the legacy server, we have set up the staging area that mirrors the folder structure in Alfresco. End users are then over time, at their own pace, copying files into the staging area.

When the population of the staging area is completed, it is copied via CIFS into Alfresco and any temporary migration rules are executed and permanent rules are also executed setting custom types, for example.

Pros and cons with CIFS import

The following are the advantages and disadvantages when using the CIFS interface for importing documents.

Advantages:

- Easy and well-known interface to work with
- Does not require any use of external tool or coding

Disadvantages:

- Slow – CIFS is a rather "chatty" protocol, so it takes much longer to do the import than for example, an in-process method
- Cannot apply custom metadata during import

Importing documents via external tool

We have talked about using CIFS for the import but it has some disadvantages, such as being slow, which we could overcome by using a tool for the import. One good open source tool that we can use is called Alfresco bulk filesystem import and can be downloaded from <http://code.google.com/p/alfresco-bulk-filesystem-import/>.

This tool provides a bulk import process that loads content into the Alfresco repository from the local filesystem. It will update an imported document if it already exists in the repository. This tool is not designed to do a full synchronization of the filesystem with Alfresco, so it will not delete files.

This tool assumes that the imported files are on the disk that is locally accessible to the Alfresco server. This will allow the import code to directly stream from the disk into the repository. Typically this means disk-to-disk streaming, which is far more efficient than any kind of mechanism that requires network I/O.

This tool is different from some other tools – in that it executes all import logic in-process via Foundation API calls. This makes it very fast and eliminates any RPC calls over the network. The tool also breaks up large imports into multiple batches, where each batch runs in its own transaction, eliminating problems with long running transactions.

We can compare the speed of this tool to CIFS by looking at one Alfresco implementation that regularly loaded thousands of image files, each one being several MBs in size. The CIFS import took approximately an hour to load 1,500 image files while this tool could load the same image files into the repository in less than five minutes. So this tool will make a huge difference when there is a lot of content to be imported.

This tool will also allow us to set metadata on any document or folder that is imported into Alfresco. This is supported by plugins so we can easily custom code what metadata should be set. Some plugins exist out of the box, such as:

- **Basic metadata loader:** It checks the type of data and sets either `cm:content` or `cm:folder` depending on if it is a file or a directory. Also populates the `cm:name` and `cm:title` with the filename as on the disk. This plugin is *mandatory* as it sets the types and names of the nodes. This is done automatically when CIFS is used.
- **Properties file metadata loader:** It reads a properties file that is associated with the imported file and sets type, aspects, and properties according to the property file.

Finally, this tool supports preserving the modified dates of imported files. It uses the technique of disabling the Audit Aspect before the import. So if you are building your own import tool it might be worth having a look at the source code for this tool, if you plan to support preserving dates.

Pros and cons with tool import

The following are the advantages and disadvantages of using an external tool for importing documents.

Advantages:

- Very fast in-process tool that doesn't do RPC calls over the network
- Splits up import into multiple transactions
- This tool can be used to apply metadata such as types and aspects during the import
- Preserves modified dates of imported files

Disadvantages:

- More complicated than the easy CIFS interface and might require some Java coding to get the best out of the tool.
- Requires you to install an AMP and restart Alfresco

Importing documents via ACP file

Another way of importing documents is via the so called **Alfresco Content Package (ACP)** files. An ACP file can be generated from an existing Alfresco installation via Alfresco Explorer UI or from several available command-line tools.

When importing ACP files from the Alfresco Explorer interface, the folder hierarchy cannot exist as this will give you duplicate name errors. So the folder hierarchy would have to be wiped out before importing the ACP.

If we did not want to wipe out the folder structure in Production, we could use Alfresco's *import* command-line tool to do the import. This tool can be told to update nodes if they exist. Unfortunately, the node UUID in the ACP file has to match the node UUID in the folder structure; so this is highly unlikely to work unless the tool can query Alfresco for UUIDs during runtime.

The following are the advantages and disadvantages with using an ACP file for importing documents.

Advantages:

- It is fast and the import is in-context, so it will be much faster than CIFS as it uses the Foundation API
- The Alfresco server does not have to be stopped
- The import is done in one transaction, so either everything gets imported or nothing
- It preserves Created and Modified Dates (using ACP import might be the only way to preserve dates if you are running an older version of Alfresco)
- The tool that generates the ACP file can apply metadata such as types to nodes that it adds to the ACP file

Disadvantages:

- Import is done in a single transaction so might cause long-running transaction problems if a huge amount of data is being imported
- The ACP file itself has to be copied into Alfresco before import starts
- More complicated than the easy CIFS interface and might require some Java coding to get the best out of the ACP Generator tool.

Common steps during document migration

No matter what method we use for the actual document import, there are some general steps that we can follow for the document migration process. They are in chronological order:

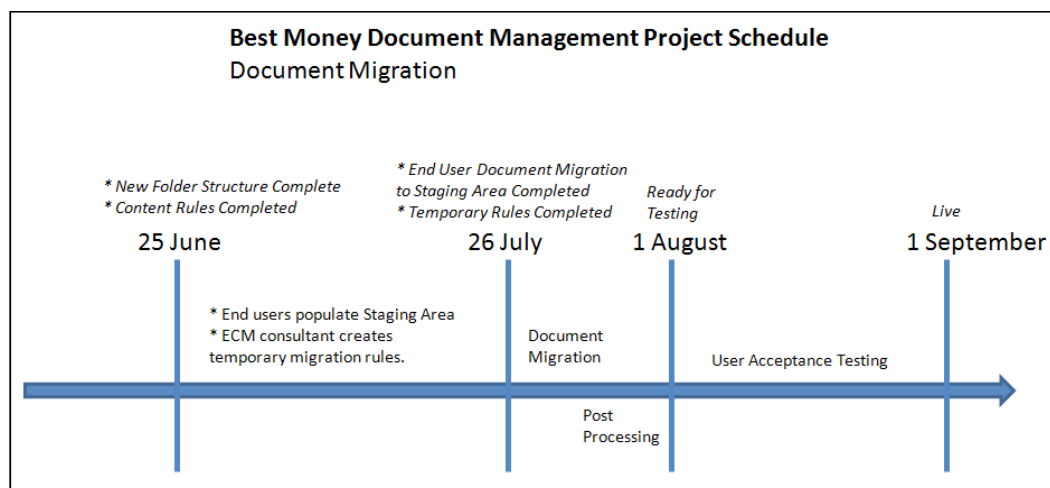
1. Set up staging area on the file share (that is, network drive). It should mirror the new folder structure in Alfresco.
2. Train end users in the new folder structure.
3. End users copy files to the staging area over a period of time.
4. Set up any temporary migration rules in Alfresco (Test Box).
5. Turn off rules that might throw errors and stop migration process (Test Box).
6. Copy the staging area to the Test Box running Alfresco.
7. Do the document migration, copy staging area via CIFS or use purpose-built tool (Test Box).
8. Remove any temporary document migration rules (Test Box).
9. Turn on any rules that were turned off during migration (Test Box).
10. Run post migration processing scripts (Test Box).
11. If Test box migration went OK, do the same thing on the Production Box.
Alternatively, if the amount of data is not that huge, export all necessary top folders from the Test Box and then import the ACPs into the Production Box.

Planning document migration

Now we have got a strategy for how to do the document migration and we have several import methods to choose from, but we have not yet thought about planning the document migration. The end users will need time to select and organize the files they want to migrate and we might need some time to write temporary import scripts. So we need to plan this well ahead of production day.

The end users will have to go through all their documents and decide which ones they want to keep and which ones they will no longer need. Sometimes the decision to keep a document is not up to the end user but instead might be controlled by regulations, so this requires extra research.

The following screenshot shows the Best Money schedule for document migration:



It is not only electronic files that might need to be imported, sometimes there are paper-based files that need to be scanned and imported. This needs to be planned into the schedule too.

Implementing document migration

So we have a document migration strategy and we have a plan. Now let's see a couple of examples of how we can implement document migration in practice.

Using Alfresco bulk filesystem import tool

A tool such as the Alfresco bulk filesystem import tool is probably what most people will use and it is also the preferred import tool in the Best Money project. So let's start looking at how this tool is used. It is delivered in an AMP and is installed by dropping the AMP into the `ALFRESCO_HOME/amps` directory and restarting Alfresco.

However, we prefer to install it manually with the **Module Management Tool (MMT)** as we have other AMPs, such as the Best Money AMP, that have been installed with the MMT tool.

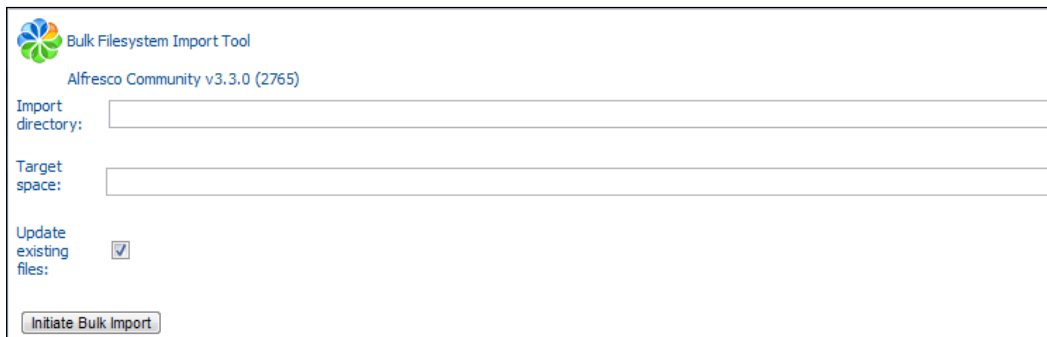
Copy the `alfresco-bulk-filesystem-import-0.8.amp` (or newest version) file into the `ALFRESCO_HOME/bin` directory. Stop Alfresco and then install the AMP as follows:

```
C:\Alfresco3.3\bin>java -jar alfresco-mmt.jar install alfresco-bulk-filesystem-import-0.8.amp C:\Alfresco3.3\tomcat\webapps\alfresco.war -verbose
```

Running Alfresco bulk import tool

Remove the `ALFRESCO_HOME/tomcat/webapps/alfresco` directory, so the files contained in the new AMP are recognized when the updated WAR file is exploded on restart of Alfresco.

The tool provides a UI form in Alfresco Explorer that makes it very simple to do the import. It can be accessed via the `http://localhost:8080/alfresco/service/bulk/import/filesystem` URL, which will display the following form (you will be prompted to log in first, so make sure to log in with a user that has access to the spaces where you want to upload the content):



The screenshot shows a web form titled "Bulk Filesystem Import Tool" for "Alfresco Community v3.3.0 (2765)". It contains three input fields: "Import directory:" (mandatory), "Target space:" (mandatory), and "Update existing files:" with a checked checkbox. An "Initiate Bulk Import" button is at the bottom.

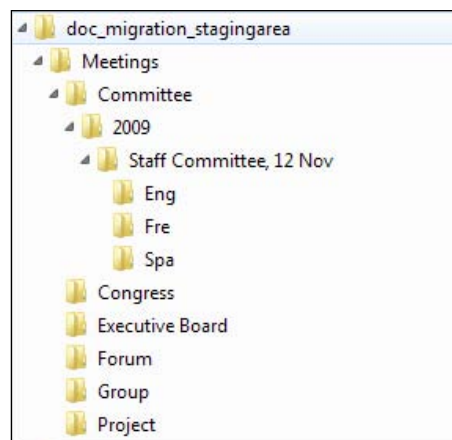
Here, the **Import directory** field is mandatory and specifies the absolute path to the filesystem directory from where to load the documents and folders from. It should be specified in an OS-specific format such as for example `C:\docmigration\meetings` or `/docmigration/meetings`. Note that this directory must be locally accessible to the server where the Alfresco instance is running. It must either be a local filesystem or a locally mounted remote filesystem.

The **Target space** field is also mandatory and specifies the target space/folder to load the documents and folders into. It is specified as a path starting with `/Company Home`. The separator character is Unix-style (that is, `/`), regardless of the platform Alfresco is running on. This field includes an AJAX auto-suggest feature, so you may type any part of the target space name, and an AJAX search will be performed to find and display matching items.

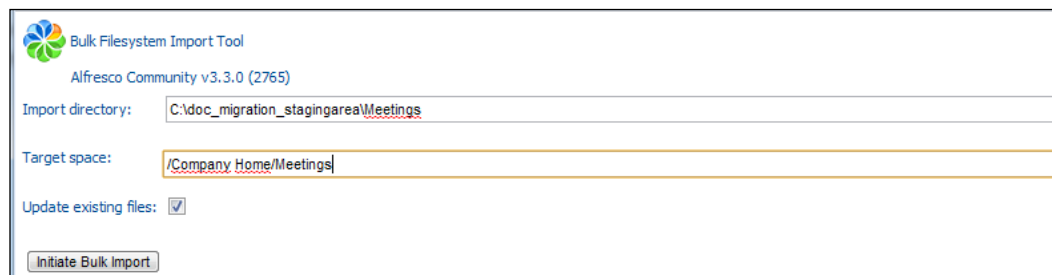
The **Update existing files** checkbox field specifies whether to update files that already exist in the repository (checked) or skip them (unchecked).

The import is started by clicking on the **Initiate Bulk Import** button. Once an import has been initiated, a status Web Script will display that reports on the status of the background import process. This Web Script automatically refreshes every 10 seconds until the import process completes.

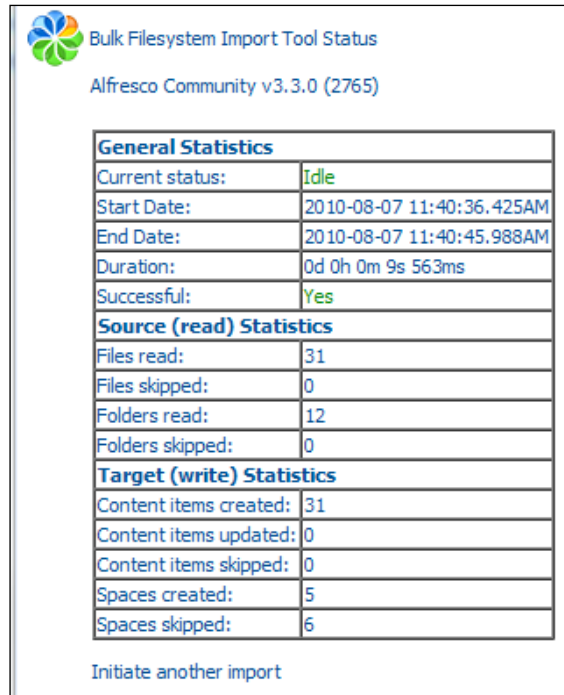
For the Best Money project, we have set up a staging area for the document migration where users can add documents to be imported into Alfresco. Let's import the **Meetings** folder, which looks as follows, in the staging area:



One Committee meeting has been added and that is what we will test to import with the tool. Fill out the Bulk Import form as follows:



Click **Initiate Bulk Import** button to start the import. The form should show the progress of the import and when finished we should see something like this:



The screenshot shows the 'Bulk Filesystem Import Tool Status' window for 'Alfresco Community v3.3.0 (2765)'. It contains three tables of statistics: General, Source (read), and Target (write). The 'Current status' is 'Idle' and 'Successful' is 'Yes'. The import took 9.563 seconds and processed 31 files and 12 folders, creating 5 spaces and skipping 6.

General Statistics	
Current status:	Idle
Start Date:	2010-08-07 11:40:36.425AM
End Date:	2010-08-07 11:40:45.988AM
Duration:	0d 0h 0m 9s 563ms
Successful:	Yes

Source (read) Statistics	
Files read:	31
Files skipped:	0
Folders read:	12
Folders skipped:	0

Target (write) Statistics	
Content items created:	31
Content items updated:	0
Content items skipped:	0
Spaces created:	5
Spaces skipped:	6

[Initiate another import](#)

In this case, the import took 9.5 seconds and 31 documents (totaling 28 MB) were imported and five folders created. If we look at the document nodes, we will see that they all have the `bmc:document` type applied and the `bmc:documentData` aspect applied. This is because of the "Apply Best Money Document Type" rule that we created in a previous chapter and added to the **Meetings** folder. All documents also have the `cm:versionable` aspect applied via the "Apply Versioning" rule, which again we created in a previous chapter and added to the **Meetings** folder.

Running Alfresco bulk import tool and applying extra metadata

What would be nice though is if the `bmc:language` property would be populated with the language that the document was written in as we have three subfolders under the committee meeting "Staff Committee, 12 Nov" for documents in English, French, and Spanish.

We can solve this by using metadata property files for each document. So for example, if we have a document in the `doc_migration_stagingarea\Meetings\Committee\2009\Staff Committee, 12 Nov\Eng` folder with the name `09Eng-ABC Report.pdf`, we can create a property file for it called `09Eng-ABC Report.pdf.metadata.properties` with the following content:

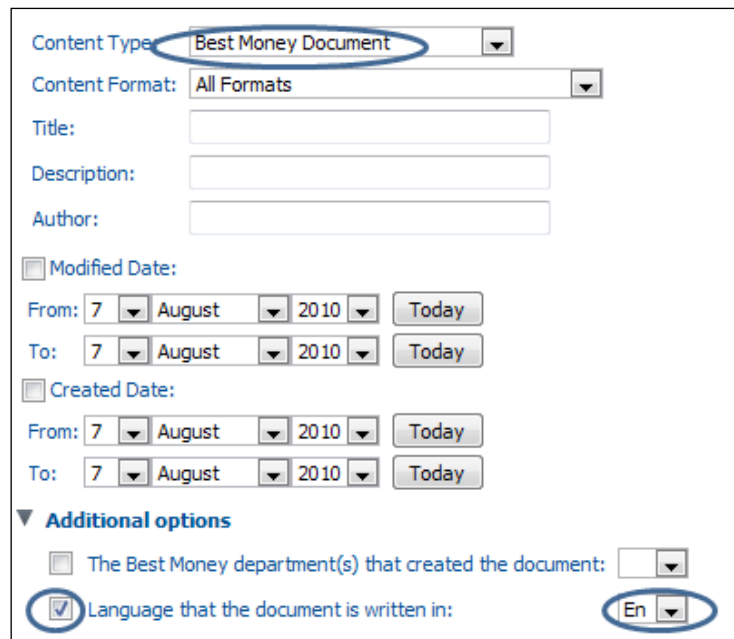
```
bmc\ : language=En
```

Note that the language has to be specified according to the `bmc:language_options` constraints that have been defined in the content model. The filename pattern for these "shadow" property files is `<filename>.<extension>.metadata.properties`. Other property files will be imported like any other file.

Now just re-run the import, with the **Update existing files** checkbox checked, and this particular file should be updated with the language property set to `En`. If we look at the details page for this document, we will see the following screenshot:



We can now use this metadata when searching via **Advanced Search** dialog in Alfresco Explorer:



[ Creating these "shadow" property files is of course not going to be very practical when there are thousands of files to import. Then it is better to have some temporary migration rules handling it or do some post migration processing of documents with JavaScript.]

Using an ACP Generator tool

Sometimes it is useful to be able to do the document migration using an ACP file. For once, it preserves modified dates, which is useful when we use older Alfresco versions that do not support preserving modified dates by turning off the Audit aspect. In other cases it might be useful to do a document migration without having to restart the Alfresco server or install extra modules in the Alfresco server, and an ACP import does not require any of this.

There are a couple of different "ACP generators" that can be found on the Internet, written in different languages and working with different Alfresco versions. The source code for this chapter comes with an ACP Generator implemented in Java.

This is to be able to get the ACP Generator to work with the latest Alfresco version, use the Java NIO API for file copy, and be able to extend it with custom functionality. We will use this ACP Generator in the following example:

The executable jar file for the ACP Generator can be found in the 3340_08_Code\bestmoney\alf_extensions\trunk_acp_generator\build\dist directory and is called acp_gen-1.0.jar. If you want to build it from scratch, use the package-acp-gen-jar ant target located in the 3340_08_Code\bestmoney\alf_extensions\trunk_acp_generator directory.

Let's try out the ACP Generator by creating a content package from the same documents and folders that we imported with the Alfresco bulk filesystem import tool. To do this we feed the ACP Generator with the source path on the local disk where the documents and folders exist and what name the ACP file should have:

```
C:\tools\acp-gen>java -jar acp_gen-1.0.jar C:\doc_migration_stagingarea\
Meetings MEETINGS_FOLDER_HIERARCHY.ACP
ACP File will be created from content in: C:\doc_migration_stagingarea\
Meetings
Temp directory for ACP: C:\tools\acp-gen\ACPTmp
Temp directory for ACP content: C:\tools\acp-gen\ACPTmp\import
ACP Metadata file: C:\tools\acp-gen\ACPTmp\import.xml
---- Generating Metadata XML for directory: C:\doc_migration_stagingarea\
Meetings
About to navigate directory tree: C:\doc_migration_stagingarea\Meetings
---- Generating Metadata XML for directory: C:\doc_migration_stagingarea\
Meetings\Committee
...
Added file (C:\doc_migration_stagingarea\Meetings\Committee\2009\Staff
Committee, 12 Nov\Eng\09Eng-ATS Report.pdf) as /import/content0.pdf
...
```

The ACP Generator will log what it's doing and here we can see that it prints logs about navigating directories, generating metadata, and adding files to the ACP package. We can now import the MEETINGS_FOLDER_HIERARCHY.ACP file from the Alfresco Explorer UI. However, when we stand on the /Company Home folder and import the ACP file, it will not work and a "**Duplicate child name not allowed: meetings**" error message will be displayed.

This is because the import via the Alfresco Explorer UI does not support updating nodes. We can import the ACP file to, for example, /Company Home/Test and it would work fine as there are no duplicate folders in that case. Alfresco supports updating nodes during import by using a command-line tool for the import. The import tool can be passed a parameter called `uuidBinding` that specifies what to do when encountering duplicate nodes.

The importer tool should be run from the `Alfresco\tomcat\webapps\alfresco\WEB-INF` directory and the ACP file should be copied into this directory. The importer tool also starts its own embedded Alfresco, so the Alfresco server needs to be shut down before doing the import. Note that the MySQL server needs to be running though. Here is how to use it:

```
Alfresco\tomcat\webapps\alfresco\WEB-INF>java -cp "classes\alfresco\
module;..\..\..\shared\classes\classes;..\..\..\lib\*;..\..\..\common\
endorsed\*;lib\*" org.alfresco.tools.Import -user admin -pwd admin -store
workspace://SpacesStore -path /app:company_home -uuidBinding UPDATE_
EXISTING -verbose MEETINGS_FOLDER_HIERARCHY.ACP

Alfresco Repository Importer

. . .
```

This is however, not working either and we will see the **"DuplicateChildNodeNameException: Duplicate child name not allowed: meetings"** error in the console window. This is because we do not have the correct UUIDs in the metadata XML for folders that exist in the repository. For the import tool to update a folder when it exists, the metadata XML in the ACP file must specify the same UUID as the folder has in the repository.

The ACP Generator tool does not add any UUIDs, if we do not tell it to. The ACP Generator can take an extra parameter that specifies the name of a property file with UUID mappings. This UUID mapping file can be generated with the following JavaScript:

```
var filename = "uuidMapping.properties";
var file = companyhome.childByNamePath(filename);

if (file == null) {
    file = space.createFile(filename);
}

if (file != null){
    var store = "workspace://SpacesStore";
    var query = "+PATH:\"/app:company_home/*\" +TYPE:\"cm:folder\"";
    var folders = search.luceneSearch(store, query);
```

```

var content = "";
for each (folder in folders) {
    var uuid = folder.properties["sys:node-uuid"];
    var pathAndName = folder.displayPath + "/" + folder.name;
    content += pathAndName + "=" + uuid + "\r\n";
}
file.content = content;
}

```

This script will generate a file with mappings as in the following example:

```
/Company Home/Meetings=a98682d2-217a-45dd-86cb-73fe01e8dde8
```

The only thing we need to do now is feed this mapping file into the ACP Generator and it will make sure each existing folder has the correct <sys:node-uuid> entity value, so when we run the *import* tool again it will work even if folders exist. Here is how to run the ACP Generator with the UUID mapping file specified:

```
C:\tools\acpgen>java -jar acp_gen-1.0.jar C:\doc_migration_stagingarea\
Meetings MEETINGS_FOLDER_HIERARCHY.ACP uuidMapping.properties
```

Now run the Alfresco import tool again and it should work fine.

Summary

This chapter has talked about a very important subject in Document Management projects, the document migration phase. Most companies have documents on a network drive that they want to migrate over to Alfresco before it goes live. We have talked about setting up a strategy for document migration and how to plan it. We have also looked at different ways to import documents into Alfresco.

When starting a document migration project it is important to plan ahead and set up a staging area where users can start copying over documents that they want to be migrated over to Alfresco.

The Alfresco bulk filesystem import tool is probably the most efficient and fastest tool to use at the moment and will probably be many people's favorite tool for document migration. And it supports the very important feature of being able to preserve modified dates during imports.

CIFS is also used by many for document imports because of its simplicity and non-intrusiveness (that is, you do not have to install anything). It is however, slow if you have a lot of data to import and cannot handle metadata.

ACP files can also be used to import documents and this chapter comes with a tool that can create ACP files from the local filesystem. ACP files can be imported with the modified dates for files preserved.

In the next chapter, we will take a look at the embedded workflow engine that Alfresco comes with. We will look at how to design business processes and how to implement them with the JBoss jBPM workflow engine.

9

Business Process Design Solutions

So far we have seen that the Alfresco platform is quite powerful when it comes to being able to implement business rules or doing different kinds of document processing with scripts. However, sometimes the processing that is needed is too complex to handle with just rules or scripts. This is where business process automation and workflows come into the picture.

This chapter will take you through how to design advanced workflows with Alfresco. Or more specifically, how to use Swimlane diagrams to design workflows that can be easily converted into jPDL, the process's definition language used by JBoss jBPM workflow engine, which is embedded in the Alfresco platform.

The Best Money's project has a requirement to implement a marketing production business process, so we are going to design this process in this chapter.

In this chapter, you will learn:

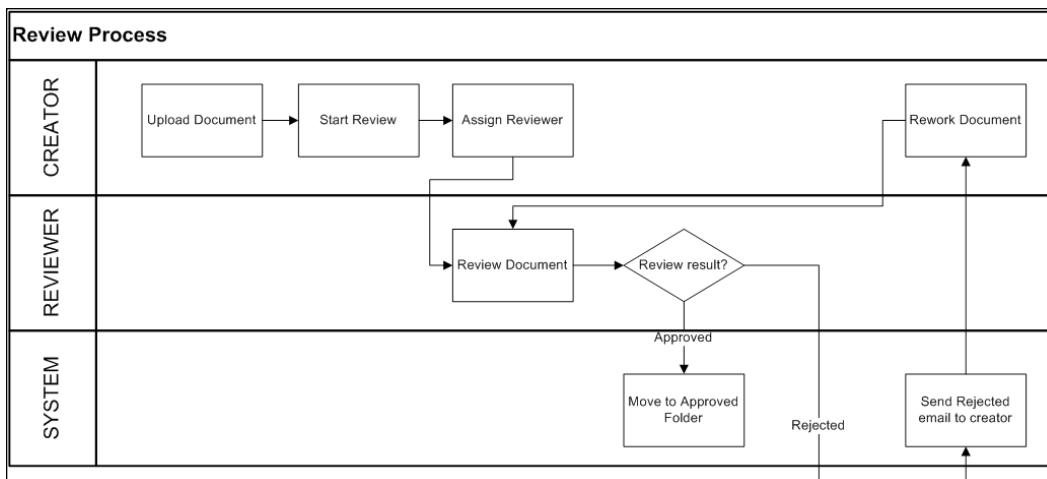
- How to define business processes with Swimlane diagrams, so it is easy to see who is supposed to complete what task
- What is jPDL
- How to use naming conventions for tasks, so they are easy to tell apart during discussions
- How to design with phases to make the process more comprehensible
- How to design with subprocesses to be able to reuse process constructs

Designing business processes with Swimlane diagrams

Before starting any implementation of workflows, it is always good to have the design completed. Just as you would not start building a house without having the blueprints completed.

Introduction to Swimlane diagrams

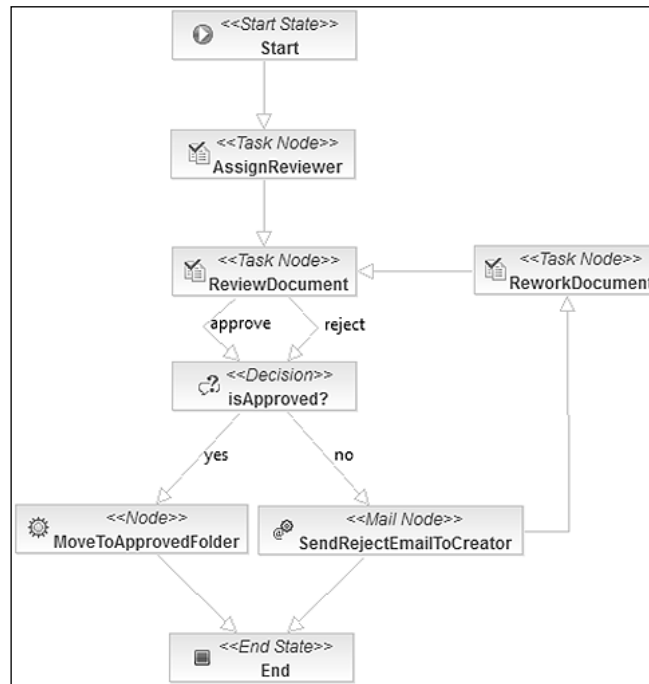
One very good way to define or describe business processes is to use a so-called Swimlane diagram. A Swimlane diagram is unique in that it keeps the tasks in the lanes grouping them together. Each lane is represented by a user or a group that will execute the tasks in that lane. A Swimlane diagram looks something like this:



As a System Architect, you will – in most cases – work together with a Business Analyst to map out the business processes that will be automated with the workflow engine. It is a good idea to spend some time with the Business Analyst to agree on how to best design the workflows with Swimlane diagrams. One good tool to use when creating the Swimlane diagrams is Microsoft Visio, which has been used to create the Swimlane diagrams that you see in this book.

Decide what symbols to use for tasks, subprocesses, decisions, parallel flows, and so on (if you do not have access to MS Visio or do not want to use Swimlane diagrams then another option is to use the standard BPMN syntax and a GUI tool supporting it).

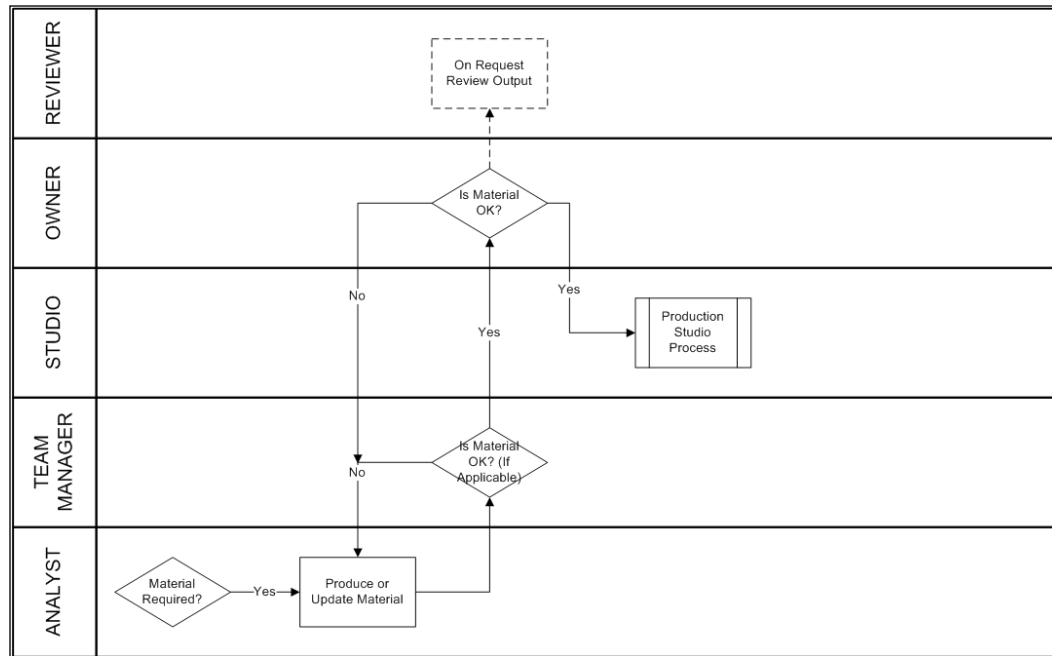
Make sure that the Swimlane diagram is easily mapped to the process definition language (PDL) that you will use when implementing the workflow. In our case, we will use the **jBPM Process Definition Language (jPDL)**. jPDL is the workflow definition language used by the JBoss jBPM workflow engine that is embedded in the Alfresco platform. The previous Swimlane diagram would look as follows in jPDL Graphical representation (using an Eclipse plug-in):



Here, we can see that the Swimlane diagram is a good compliment to the jPDL diagram as it also shows you what role or user is responsible for executing a particular task.

In the beginning, it might be a good idea for the System Architect to work closely with the Business Analyst when defining the Swimlane diagrams that will represent the business processes. By doing so they are both on the same page on how to do this and the Swimlane diagrams will be created in a way so as to be easily represented in jPDL.

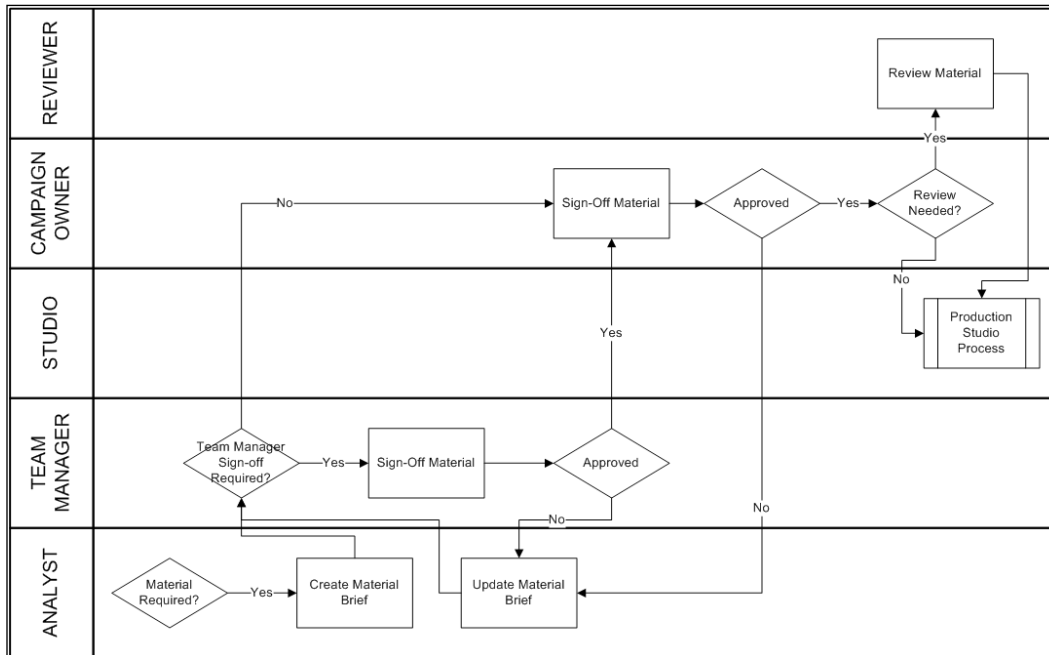
If we do not follow this, and let the Business Analyst work alone, then we could encounter problems as in the following example where a Business Analysts starts off by drawing a piece of a Marketing Material Production process like this:



At first, this might look okay, but there are several things that need to be clarified before they can be mapped into the process definition language (that is jPDL):

- The "**Produce or Update Material Brief**" task for the ANALYST probably needs to be split into two tasks as you might want to display different Title, Description, other data, and so on for when the brief is first created and when it is updated. The Update Material Task would also come at a different point in the flow.
- The "**Is Material OK? (If Applicable)**" decision for the TEAM MANAGER would have to be preceded with a task where the TEAM MANAGER decides if the material brief looks okay, and then this can be followed with a decision node. Also, what does "If Applicable" mean in this case?
- The "**Is Material OK?**" decision for the OWNER would have to be preceded with a task where the OWNER decides if the material brief looks okay, and then this can be followed with a decision node.
- The "**On Request Review Output**" task seems to be put in the diagram to only be executed some times, which will not work.

Every task and decision in the Swimlane diagram has to be clear and non-ambiguous. A better version of the Swimlane diagram would look like this:



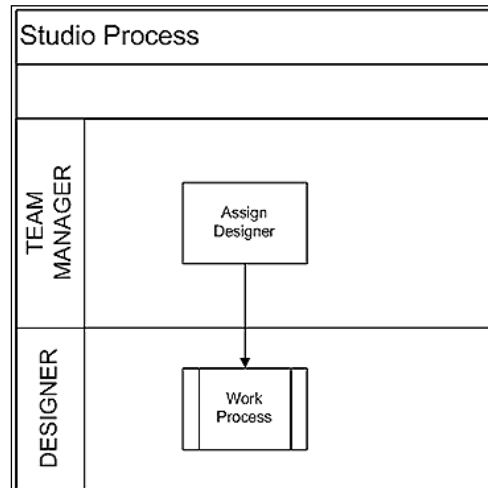
Here, we can see that every task and decision is clear and specified in such a way that it is easy to know how to map this business process into the process definition language. Make sure to work together with the Business Analyst a lot in the beginning; it will save you loads of time in the implementation phase. You will avoid lots of questions like "What do you mean here?", "Do you want it to take this path or that path, you cannot have it both ways?", "Is this a task or a decision, should this be a subprocess or what?", and so on.

If the Business Analyst understands how you have to write the process definition, it is going to help a lot and increase productivity enormously.

Subprocesses

Another thing that you might want to make the Business Analyst aware of and that will again save you a lot of time is the use of subprocesses. As in any programming, apply the DRY (Do not Repeat Yourself) principle. If you see that a group of nodes in a Swimlane diagram occur in several places, for example, a sign-off procedure, it is time to think about lifting them out into a separate subprocess.

Subprocesses are depicted in the following way:



This Swimlane diagram shows a subprocess called **Work Process** that is called from a parent process called **Studio Process**.

Using subprocesses has many advantages:

- You save implementation time as you do not have to define and implement the same group of nodes in several places in the main process.
- You will have less chance of errors as you have less process definition and implementation to maintain.
- You will only have to test this group of nodes once, in the subprocess, and not in every place in the main process where they might be needed.
- Updates to this group of nodes have to be done only in one place, which makes maintenance a lot easier.
- Over viewing and discussing around the complete/main process becomes much easier.

So, subprocesses can be used not only to re-use process definitions and implementation code but also to divide processes into more comprehensible pieces. So when you feel you begin to have a problem following the process definition from start to end, it might be a good time to start extracting some parts of it into subprocesses.

Further on, when you use subprocesses it is easier to track down problems. Subprocesses are deployed separately from the main process. So if you suddenly get an error after deploying a new version of a subprocess, you can just focus on that particular subprocess and if you cannot immediately figure out what the problem is, you can just re-deploy the previous working version.

If you just have one big mega process it gets much harder to track down where the problem is in the process compared to if you are focusing on a smaller part of a process definition.

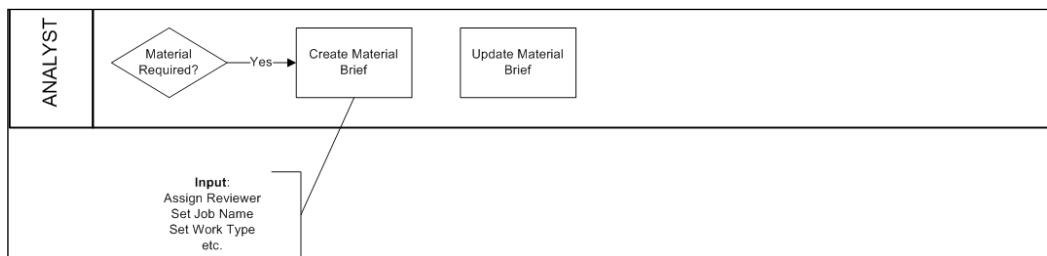
You have heard only about positive things with subprocesses, isn't there any negative stuff about using them? Well, there is actually. When you use subprocesses, you do not see their task history from task dialogs in the main process or vice versa. So, if I am in a Sign-off subprocess I do not see any of the previously completed tasks from the main process that called the Sign-off process. So when I open up a task dialog and look in the task history section I only see completed tasks from the Sign-off process. It's the same thing in the main process, I do not see any completed tasks from the Sign-off subprocess.

To get a complete task history for the main process including all involved subprocesses, you will have to do some custom coding. We will look at one way of doing this by generating an Excel spreadsheet at the end of the main process with all the involved process's completed tasks.

Task metadata

When defining a business process, it is also a good idea to think about what information (that is, metadata) should be carried through the process and displayed in the different task dialogs. I find it beneficial to actually have the Business Analyst specify what properties (that is, metadata) should be displayed or input for a task when they define the Swimlane diagram. After all, the Business Analyst should know the domain inside out for the system that should be built.

So we can for example, use annotations when we specify the properties for the different tasks. If we think about the Swimlane diagram in the previous figure, we could specify properties for the **Create Material Brief** task as follows:

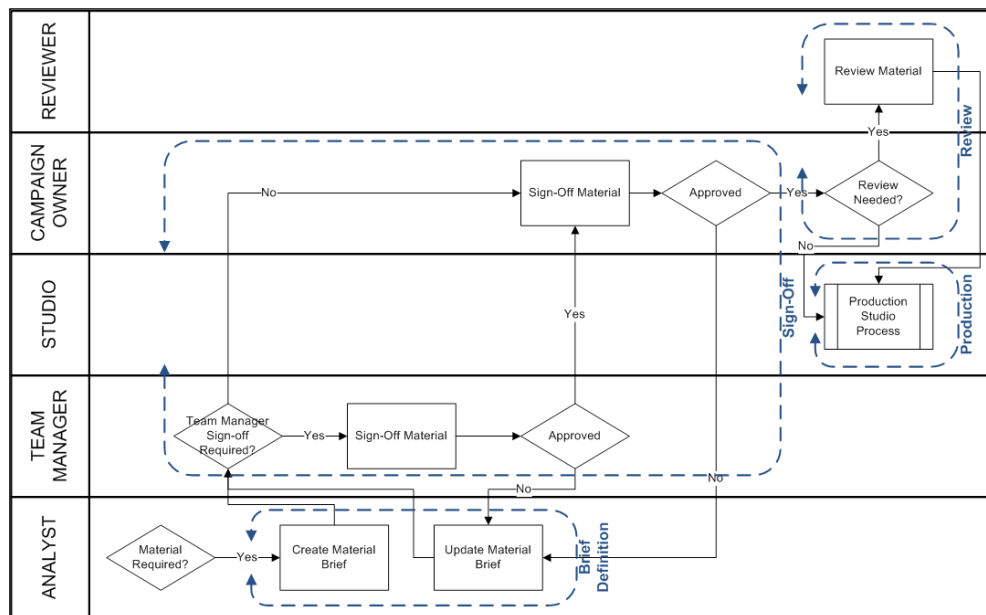


Here, the Business Analyst has told the process implementer that the end user should be able to assign a reviewer, set a job name, and set a work type when completing this task. If the work type has some constraints then the Business Analyst would specify that also such as, for example, "Set Work Type [concept, design]".

Process phases

If you have a rather big process with loads of tasks, it is sometimes good to divide it into different phases. These phases can be graphically depicted in the Swimlane diagram and they can also be defined in the process definition language as Super States.

The following Swimlane diagram illustrates:



In this Swimlane diagram, we have specified four different phases, first the **Brief Definition** phase, then it transitions into the **Sign-off** phase, then to the **Review** phase, and finally it transitions into the **Production** phase.

Phases are useful as you quite often get requests to keep some kind of process status as the process progresses. Setting this status is best done when you enter each phase, (that is, when you enter the Super State during implementation in jPDL) as you then do not have to worry about setting it for each node in the phase. And when nodes are added to a phase or moved outside a phase you can be confident that the process status is still accurate for a particular node.

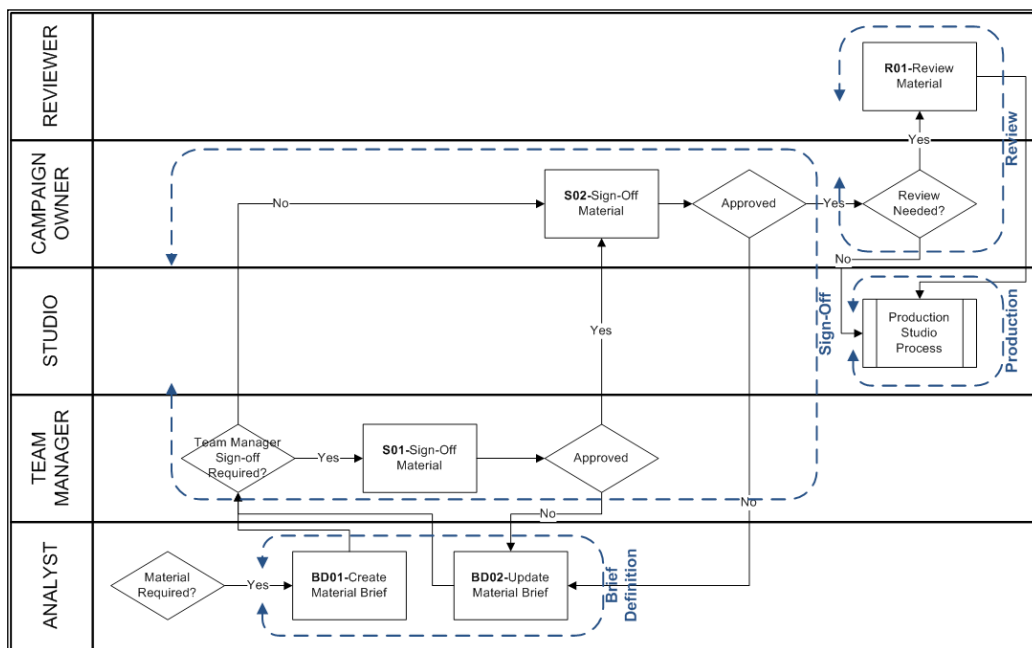
Quite often permissions and status information is updated when documents pass through the different phases of the process. For example, when a document enters the production phase, we probably want to make it read-only and put in a "live" folder. Alfresco supports setting permissions on documents and folders via both embedded and remote APIs.

The process status is also often used when you produce reports. Another thing that you can also use phases for is to use them in task naming conventions, which is the next thing we are going to talk about.

Task naming convention

When the business process you have developed is going live, (that is, deployed into production) it is sometimes useful to have some kind of naming convention for all the tasks. This makes it easier for End Users, Business Analysts, and System Architects to know exactly what task is referred to in a discussion around something.

The tasks could for example, be named and numbered according to what phase they are in and where in the phase they are located:



Here, we have used a task naming convention that starts with the Phase (that is, BD = Brief Definition, S = Sign-Off, R = Review, and P = Production) followed by where in the flow the task is located (that is, 1st, 2nd, 3rd ... task in the phase).

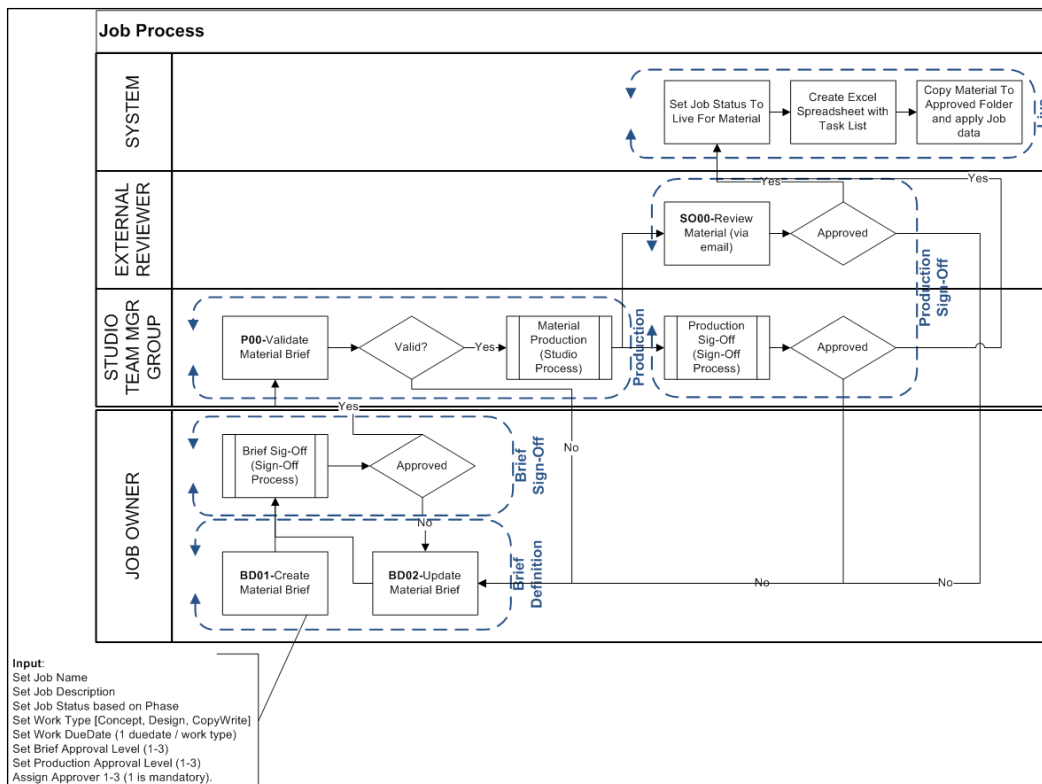
Now, when I talk about task **BD01**, everyone knows that I am talking about the **Create Material Brief** task and there are no misunderstandings.

Designing the material production process

This section takes you through how to design the complete Best Money marketing material process with its subprocesses using Swimlane diagrams.

Job process Swimlane diagram

For Best Money, the Job process is the main process for marketing material production. The following Swimlane diagram has been created for it by the Business Analyst:



The Job Process is started by the Job Owner who creates a brief (BD01) with information about what marketing material should be created. This brief then goes into a sign-off procedure where there can be up to three levels of sign-off depending

on what Brief Approval Level has been set by the Job Owner. If the brief is approved then it goes to the Studio Team Managers, if it is not approved then it goes back to the Job Owner for an update (BD02).

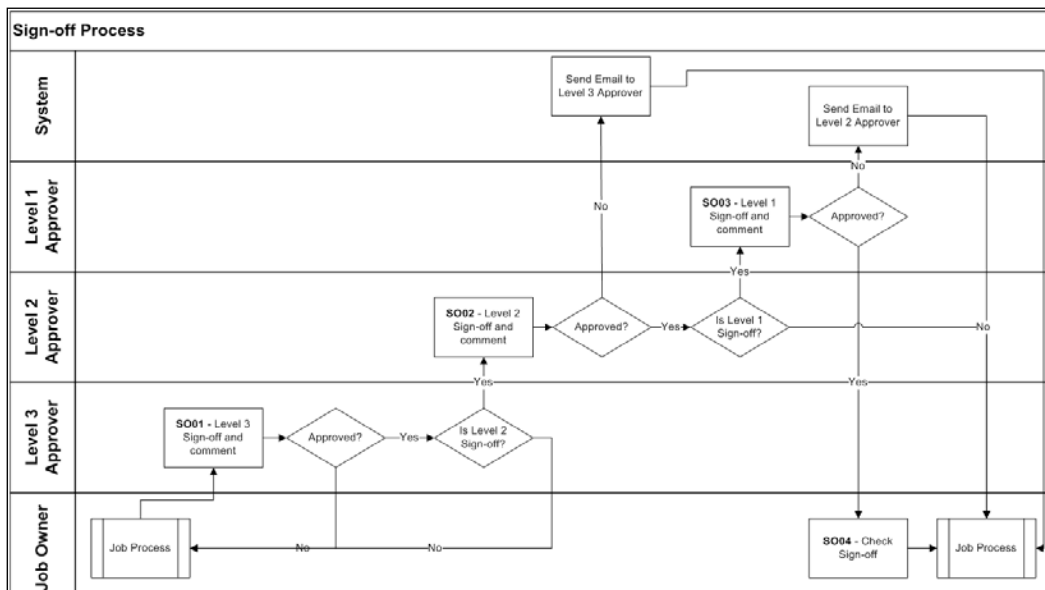
After approval, one of the Studio Team Managers takes ownership of the task to validate the brief (P00). If validation is successful then the brief is passed on to the Studio Process where the material is created according to the marketing brief. Once the material has been created by the Studio Team, it goes into the sign-off procedure again.

The Production sign-off procedure is the same here as it is when the brief is approved, so we reuse the same subprocess. At the same time as the production sign-off, (that is, in parallel) the produced material is sent off to an External Reviewer via e-mail for approval (SO00).

When the marketing material has been approved by the External Reviewer and the production sign-off process has approved the material, its status is set to Live and it is copied to an Approved Material folder and the Job data is applied to all content. Also, a list of all completed tasks is created in the form of an Excel Spreadsheet and stored in the material folder.

Sign-off process Swimlane diagram

The sign-off process has been designed in a generic way so it can be used for both Brief and Production sign-off. The Sign-Off subprocess looks as follows:



There are three levels of approval in this diagram:

- **Level 1:** Director level approval
- **Level 2:** Marketing Manager level approval
- **Level 3:** Group Manager level approval

So, it starts off by the Level 3 Approver reviewing the marketing brief or produced material (SO01). The Level 3 Approver is mandatory. If the Level 3 Approver approves the brief or material then it is passed on to the Level 2 Approver – if level 2 approvals have been selected by the Job Owner when the brief was created.

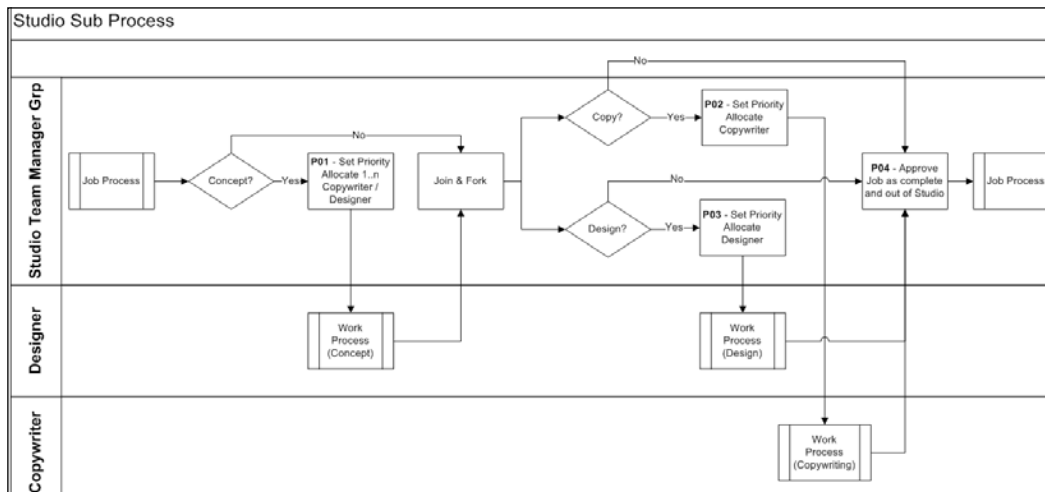
The Level 2 Approver reviews the brief or material, (SO02) and if approved it is passed on to the Level 1 Approver – if Job Owner has selected level 1 approval. The Level 1 approver reviews the brief or material (SO03) and rejects or approves.

If the Level 2 Approver is rejecting the brief or material then an e-mail is sent to the Level 3 Approver. Finally, if the Level 1 Approver rejects the brief or material an e-mail is sent to the Level 2 Approver. Anytime the brief is rejected by an approver the sign-off process aborts and the sign-off result is set to Rejected and passed back into the Job Process.

A successfully approved marketing brief or produced material is passed on to the Job Owner so he or she can check any comments (SO04) made by the approvers before the brief or material is passed on back into the Job Process as Approved.

Studio process Swimlane diagram

The studio process is where the material described in the brief is produced. The Studio subprocess looks as follows:



When the marketing brief has been approved it is passed on into the Studio Process where one of the Studio Team managers will start by allocating one or more copywriters or designers to do the Concept work (P01). The Concept work is also handled by a subprocess as all the work done in the Studio Process is the same, so no need to repeat these nodes everywhere.

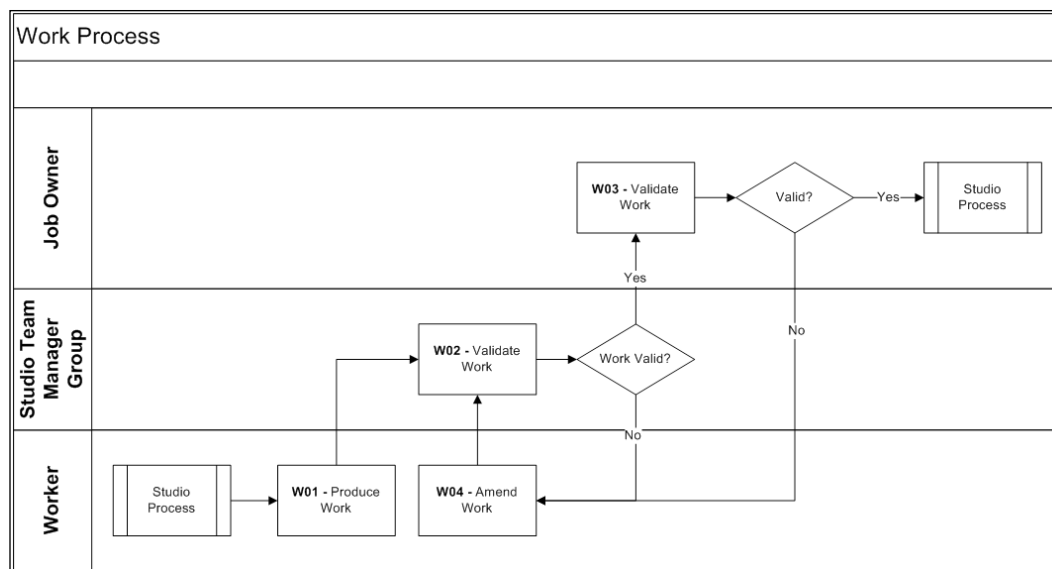
After Concept work is completed by all workers assigned to it, we fork out into two parallel paths to do the design (P03) and copywriting (P02) work. When all work is completed it is approved/validated (P04) by one of the Studio Team managers before it goes back into the Job Process.

As you can see, this process contains both joins and forks. There is one node depicting this "Join and Fork" and there is also an implicit join. Sometimes it can be good to use a node to depict a Join or a Fork and other times it might do with just drawing the transition lines.

In this process, we can see that the P01 task node allows the user to allocate one or more concept workers to work in parallel. So here we will need a special construct for this and it will probably include more nodes than just this one task node in jPDL.

Work process Swimlane diagram

The work process is designed in a generic way so it can be reused in several places. Its Swimlane diagram looks as follows:



The Work Process starts with the worker producing the marketing material (W01). This Swimlane has an Abstract person called Worker, so this process would have to have the concrete Designer or Copywriter passed into it. When the work is produced it is validated first by one of the Studio Team Managers (W02) and then the Job Owner (W03).

If the work is not approved then the Worker amends (W04) it and then it is validated again and so on, until approved and passed back up to the Studio process.

Summary

In this chapter, we have learned how to use Swimlane diagrams to design business processes that can be easily converted into the JBoss jBPM process definition language (that is jPDL). We designed one main Job process that uses a Sign-Off subprocess and a Studio subprocess. The Studio subprocess also uses a Work subprocess in several places.

A Swimlane diagram has the advantage that it is laid out in a way that makes it easy to see what role or user is supposed to complete a task in the business process. Swimlane diagrams are a very good compliment to jPDL diagrams and they are easy to discuss for people from different backgrounds, such as end users, business analysts, software architects, and so on.

We have seen how we can use reusable subprocesses when designing business processes and how this makes it easier to overview the process, saves time, and gives a lower maintenance burden in the long run.

Using a naming convention for task names also makes it easier to maintain the business process and end users, the business analyst, and the software architect will have an easy way to distinguish between all the tasks in the business processes. The task naming convention also tells everyone about what phase the task belongs to.

Finally, we had a look at how a process can be divided into different phases to make it easier to manage. During implementation it is also good to use phases as that sometimes make life easier when setting variables related to a specific phase.

In the next chapter, we will implement the complete Marketing Production process based on these Swimlane diagram designs.

10

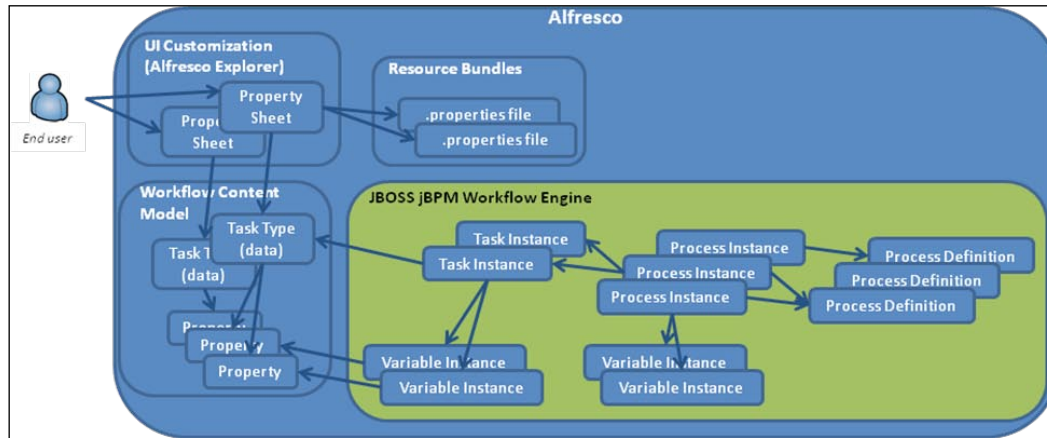
Business Process Implementation Solutions: Part 1

In the last chapter, we saw how to design business processes with the help of Swimlane diagrams and we designed Swimlane diagrams for the Best Money's Marketing job process. In this chapter and the next, we will go through how to build advanced workflows and implement the Marketing Job process with Alfresco's built-in workflow engine.

This chapter, which is part 1 of implementing the complete Job process, starts off by going through how to implement the generic Work subprocess. It shows us the basic techniques needed to implement workflows in the Alfresco environment.

Alfresco implements workflow support by embedding the JBoss jBPM workflow engine and integrating it with Alfresco features such as the client UI, content model, and resource management.

The following figure shows how the Alfresco workflow feature is built up:



The jBPM workflow engine basically handles all the runtime execution of a registered *process definition*. When a process is started from a definition, a *process instance* is created. The workflow engine will create a *task instance* whenever there is something for an Alfresco user to do.

A process instance can also create variables that will follow the process throughout its life. Task instances can also create *variable instances* and in that case the variables are accessible only from the task instance.

[ Tasks are completed by a physical person while for example a node that executes a script is managed by the system and does not involve a task instance.]

All variable instances are based on Alfresco properties from standard content model definitions. The user sees task information via standard Alfresco property sheet definitions. The task dialogs are localized via standard Alfresco resource file management.

In this chapter, you will learn:

- Implementing business processes with jPDL
- Defining workflow data as a standard Alfresco content model
- Using process and task variables
- Using standard Alfresco property sheets to define task dialogs for data input
- Externalizing the labels in the task dialogs

- Deploying a workflow definition
- Testing a workflow definition

Implementing the marketing production workflow

When the Business Analyst has finished the Swimlane diagrams for the Job, Studio, Sign-off, and Work processes, it is time for the architect to turn those into workflow definitions so they can be implemented and deployed into the Alfresco jBPM workflow engine.

To do this we are going to use the jBPM plug-in for Eclipse so we can get the skeleton workflow definitions (that is, the start of the jPDL files) from just drawing the workflows graphically. The jBPM Process Designer can be downloaded from <http://sourceforge.net/projects/jbpm/files>.

We are going to start implementing the generic Work subprocess, which is used by the Studio process in several places, as it is a rather simple one and we can explain some of the most important concepts when defining and implementing this subprocess.

Start up Eclipse and create a new Java project from the existing source (that is, from 3340_10_Code\bestmoney\alf_extensions\trunk_alfresco). Then add a new process definition for the work process by right-clicking on the workflows directory and then select **New | Other....** Give this process the name work.

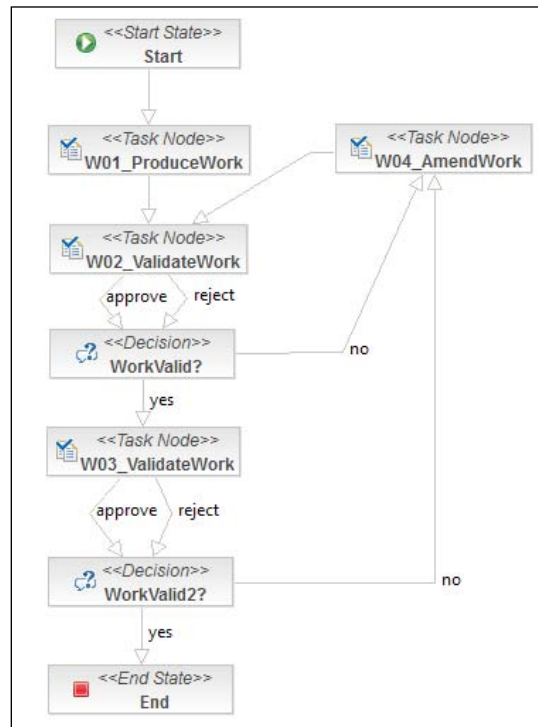
Implementing the Work subprocess

This chapter will take us through everything that is needed to implement a workflow with the JBoss jBPM workflow engine and its integration with Alfresco. We will do the implementation of the Work subprocess in the following order:

1. Create the process definition (jPDL).
2. Define content model for task data.
3. Create property files for UI labels.
4. Setting dynamic descriptions and due date.
5. Defining Job data.
6. Creating task property sheets.
7. Bootstrapping UI property files and the task property sheets.
8. And finally testing the workflow implementation.

Work process—workflow definition (jPDL)

Now define a process definition graphically that looks like this for the Work subprocess (Note: Use underscore (_) instead of dash (-) when naming the tasks, it will make life easier later on when specifying the resource file property names):



For the Work subprocess, we have decided to prefix all task node names with W<sequence number>_, so it is easy to recognize the task as belonging to the Work process. We have added an extra transition after each ValidateWork task to show that there are two paths (that is, reject or approve) that the process can take when completing these tasks. This will also result in two buttons in the **Task** dialog in Alfresco Explorer (that is, **Approve** and **Reject**) for a ValidateWork task (compared to just one **Complete** button).

Note also that we have added 2 to the WorkValid2? decision node name as it has to differ from the other decision node (that is, the WorkValid? node). Other than that, it was quite straightforward to create this jPDL process definition graphically.

If you now switch from the **Diagram** tab to the **Source** tab in Eclipse, you will see a skeleton process definition looking something like this:

```
<?xml version="1.0" encoding="UTF-8"?>

<process-definition xmlns="urn:jbpml.org:jpd1-3.2"
  name="bmw:workProcess">

  <start-state name="Start">
    <transition to="W01_ProduceWork"></transition>
  </start-state>
  <task-node name="W01_ProduceWork">
    <transition to="W02_ValidateWork"></transition>
  </task-node>
  <task-node name="W02_ValidateWork">
    <transition to="WorkValid?" name="reject"></transition>
    <transition to="WorkValid?" name="approve"></transition>
  </task-node>
  <task-node name="W03_ValidateWork">
    <transition to="WorkValid2?" name="approve"></transition>
    <transition to="WorkValid2?" name="reject"></transition>
  </task-node>
  <task-node name="W04_AmendWork">
    <transition to="W02_ValidateWork"></transition>
  </task-node>
  <decision name="WorkValid?">
    <transition to="W03_ValidateWork" name="yes"></transition>
    <transition to="W04_AmendWork" name="no"></transition>
  </decision>
  <decision name="WorkValid2?">
    <transition to="End" name="yes"></transition>
    <transition to="W04_AmendWork" name="no"></transition>
  </decision>
  <end-state name="End"></end-state>
</process-definition>
```

The process definition name has manually been set to `bmw:workProcess` (we will define the `bmw` namespace later when we create the **Workflow Content Model**). The jPDL XML Schema version has also been updated to 3.2 (from 3.1) as the jBPM jPDL designer is targeted towards version 3.2; it also enables us to add a `description` element to the node definitions.

Here you can see that using the Eclipse plug-in quickly gets you going with the process definition and you don't have to know too much about jPDL syntax. However, this process definition basically does not do anything useful, it needs some tasks with Swimlane definitions, logic, and variables to work properly.

First, we are going to need the so-called **swimlanes** for the users and groups that are going to execute the different tasks in the Work Process. So let's define the `jobOwner`, `studioTeamManagers`, and `worker` swimlanes, as follows, at the top of the file:

```
<?xml version="1.0" encoding="UTF-8"?>
<process-definition xmlns="urn:jbpm.org:jpdl-3.2"
    name="bmw:workProcess">

    <swimlane name="jobOwner">
        <assignment class=
            "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
            <actor>#{jobOwner}</actor>
        </assignment>
    </swimlane>
    <swimlane name="studioTeamManagers">
        <assignment class=
            "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
            <pooledactors>#{people.getGroup(studioTeamMgrsGroupName)}
            </pooledactors>
        </assignment>
    </swimlane>
    <swimlane name="worker">
        <assignment class=
            "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
            <actor>#{bpm_assignee}</actor>
        </assignment>
    </swimlane>
    ...

</process-definition>
```

Here, we can see that the actors of the swimlanes are all specified as variables and are going to be passed into the subprocess from the parent process. The `#{jobOwner}` and `#{bpm_assignee}` variable expressions are `Person` objects, while the `#{people.getGroup(studioTeamMgrsGroupName)}` variable expression uses the passed-in Studio Team Manager's group name to fetch a `Group` object to act as a pooled actor. We want to pass in group names and other literals from the parent process as we do not want them defined in more than one place.



When pooled actors are used, all members of the group will get the task assigned. And the task will show up in the **My Pooled Tasks** dashlet in Alfresco Explorer. Then one member takes ownership of the task and completes it (only one person can complete it). Also note that the **My Pooled Tasks** dashlet is not displayed by default in the dashboard and has to be added via configuration.

The `bpm_assignee` variable will be set to the currently selected worker (that is, designer or copywriter) in the parent process before being passed into the Work subprocess.

The next thing that we want to do is save the process ID for the subprocess into a process variable and we do that in the `start-state` node just before we leave it:

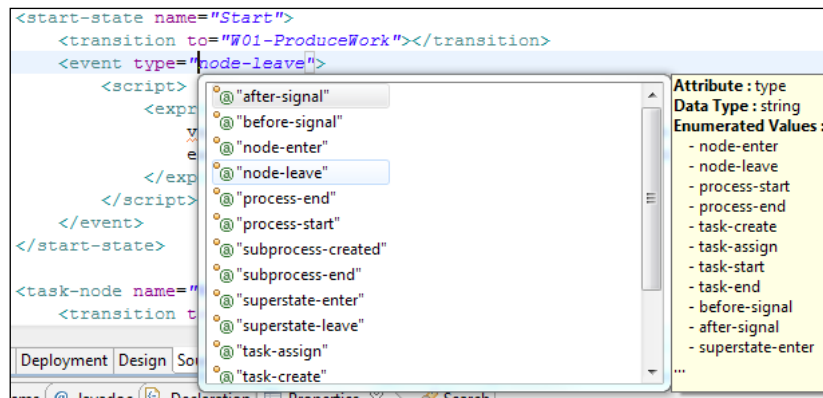
```
<start-state name="Start">
  <transition to="W01_ProduceWork"></transition>
  <event type="node-leave">
    <script>
      var procId = executionContext.processInstance.getId();
      executionContext.setVariable("procId", procId);
    </script>
  </event>
</start-state>
```

So why do we save the subprocess ID? We want to keep a process ID in the parent process for each subprocess that has been executed as part of the parent process. This will become useful later on when you want to create reports, query the task database, and so on. So this variable will actually be passed back up into the parent process.

Here we save the process ID as a process variable called `procId`. There are two ways of storing variables—either as process variables or as task variables. If you have a value and you want it to live throughout the lifecycle of the process instance, you should define it as a process variable. On the other hand, if you only need the variable when handling a task then you should store the variable as a task variable, of which we will see examples later on.

We have now seen quite a bit of jPDL syntax and when we set the process variable we did that in an event handler and we then needed to know what events were available to use. So how do we find out what syntax is legal, and what options are available, and so on? For example, there are many events that can be used in a node to do some processing, but which ones we can use depends on the node type. So to have an idea of what event(s) can be used in a certain node, it is a good idea to use an editor that recognizes the jPDL syntax (that is, an editor that has imported the jPDL Schema, which can be downloaded from <http://docs.jboss.org/jbpm/xsd/jpdl-3.2.xsd>).

The Eclipse JBoss jPDL plug-in recognizes the jPDL schema and it gives you the possibility to see what events are available to use by clicking *Ctrl + Space* as follows:



Unfortunately, the event type list that we can see in the preceding screenshot is the complete list of event types available in jPDL and not all of them can be used for a start-state node, in fact only the node-leave event is allowed. The jPDL schema does not restrict the event type per node type so we have to look elsewhere for information about what event types can be used for a node. The best is to look up the JBoss jBPM documentation at <http://docs.jboss.org/jbpm/v3/userguide/jpdl.html>. The *jPDL xml Schema* section contains information about each element in jPDL.

If we look at the documentation for the start-state node we can see that it says that the event element description is supported event types: {node-leave}.

So if we tried to set the subprocess instance ID in the node-enter event of the start-state node it would result in it being set to null. The process syntax would however be correct and the process definition would be deployable and executable. This is something to watch out for, otherwise we can spend significant time tracking down why an event definition is not working the way we want it to.

You are probably wondering now what this script syntax is that we are using to set the process variable. A script tag is an action that executes a BeanShell script or an Alfresco JavaScript. When using jBPM with Alfresco you can choose between two different script languages depending on what you want to do.

Either you can use a BeanShell script (<http://www.beanshell.org/home.html>) as we have done here, or you can use an Alfresco JavaScript (http://wiki.alfresco.com/wiki/JavaScript_API) if you need access to Alfresco-specific things, such as a variable pointing to the company home folder. We will discuss more around scripting as we go along.

We have got the swimlanes and what we need now are some tasks that use them.



So why do we need to define tasks when we already defined a task-node element? Good question, the `task-node` element is just a container for all behavior having to do with a task. It does not specify who should do the task or where to transit after the task is completed or any event processing, and so on.

So in each task node we are going to define task elements as follows:

```
<task-node name="W01_ProduceWork">
  <task name="bmw:W01_ProduceWorkTask"
    swimlane="worker"></task>
  <transition to="W02_ValidateWork"></transition>
</task-node>
<task-node name="W02_ValidateWork">
  <task name="bmw:W02_ValidateWorkTask"
    swimlane="studioTeamManagers"></task>
  <transition to="WorkValid?" name="reject"></transition>
  <transition to="WorkValid?" name="approve"></transition>
</task-node>
<task-node name="W03_ValidateWork">
  <task name="bmw:W03_ValidateWorkTask" swimlane="jobOwner">
    </task>
  <transition to="WorkValid2?" name="approve"></transition>
  <transition to="WorkValid2?" name="reject"></transition>
</task-node>
<task-node name="W04_AmendWork">
  <task name="bmw:W04_AmendWorkTask" swimlane="worker">
    </task>
  <transition to="W02_ValidateWork"></transition>
</task-node>
```

For each task-node we have added a task and named it according to the following naming convention:

```
{namespace from workflow content model}:{task node name}Task
```

And for each task we have set what swimlane should be used. Basically, we specify who should execute the task, either a person or a group of people (that is, pooled assignment).

In the workflow content model, (which we will define later) we will define a type for each task with the same name and namespace as the task name. You do not have to follow this naming convention, but it makes life easier when you look at a definition in the workflow content model and you want to know what it is used for.

Last thing we need to do to get this workflow going is to define one variable in the W02_ValidateWork task node and one in the W03_ValidateWork task node that holds the result of the validation (that is, reject or approve). And then check these variables in the WorkValid? and WorkValid2? decision nodes:

```
<task-node name="W02_ValidateWork">
  <event type="node-enter">
    <script>
      <variable name="workApprovedByTeamMgr" access="write"/>
      <expression>workApprovedByTeamMgr = false;
    </expression>
    </script>
  </event>
  <task name="bmw:W02_ValidateWorkTask"
    swimlane="studioTeamManagers"></task>
  <transition to="WorkValid?" name="reject"></transition>
  <transition to="WorkValid?" name="approve">
    <script>
      <variable name="workApprovedByTeamMgr"
        access="read,write"/>
      <expression>workApprovedByTeamMgr = true;
    </expression>
    </script>
  </transition>
</task-node>
<decision name="WorkValid?">
  <transition to="W04_AmendWork" name="no"></transition>
  <transition to="W03_ValidateWork" name="yes">
    <condition>#{workApprovedByTeamMgr == true}</condition>
  </transition>
</decision>
<task-node name="W03_ValidateWork">
  <event type="node-enter">
    <script>
      <variable name="workApprovedByJobOwner" access="write"/>
      <expression>workApprovedByJobOwner = false;
    </expression>
    </script>
  </event>
```

```

<task name="bmw:W03_ValidateWorkTask" swimlane="jobOwner"></task>
<transition to="WorkValid2?" name="reject"></transition>
<transition to="WorkValid2?" name="approve">
  <script>
    <variable name="workApprovedByJobOwner"
      access="read,write"/>
    <expression>workApprovedByJobOwner = true;
  </expression>
</script>
</transition>
</task-node>
<decision name="WorkValid2?">
  <transition to="W04_AmendWork" name="no"></transition>
  <transition to="End" name="yes">
    <condition>#{workApprovedByJobOwner == true}</condition>
  </transition>
</decision>

```

Here we have used the `workApprovedByTeamMgr` and `workApprovedByJobOwner` Boolean variables to keep track of what the approver's result is.



One thing to note here is that we have changed the location of the no transition element in both decision nodes so that it sits before the yes transition element. This is because there is a problem with some Alfresco versions to get it to work if the no transition is not first, the condition then always evaluates to true no matter what is selected (that is, approved or rejected).

You might have noticed that a different script syntax has been used here to set the process variable. A `variable` element has been defined to specify the variable name that should be written (that is, because of the `access="write"`) to the execution context and the script code is inside an `expression` element.

By default (that is, by using the `executionContext.setVariable` and not using the `variable` and `expression` elements), all process variables are available as script variables (that is, inside the `script` element) but no new script variables (that is, variables defined as `var procId = ...`) will be written to the execution context as process variables without using the `executionContext.setVariable`.

To customize the default behavior of loading and storing variables into the script, the `variable` element can be used as a sub-element of the `script` element. In that case, the script code also has to be put in a sub-element called `expression`. And then the process variables specified with the `variable` element will be the only custom variables available to use in the script.

Note that when using the `variable` element, only one variable can be written back to the process context (that is, you can have only one variable defined with `access="write"`) but multiple variables can be loaded into the script (that is, with `access="read"`).

Further on, the following script variables will always be available to use in a script:

- `executionContext`: Current execution context, which can be used to get to process instance ID, write new custom process variables.
- `token`: Current execution path's token. A token is the runtime concept that maintains a pointer to a node in the graph.
- `node`: Current node instance.
- `Task`: Current task container.
- `taskInstance`: The task instance that can be used to read and write task variables.

Try to keep your execution context as tidy as possible and only keep/store process variables that are used to control the process or used as a help to fetch data for reporting, for example. Do not use the execution context to store model data in process variables (model data is stuff from your local domain, in our case Job data).



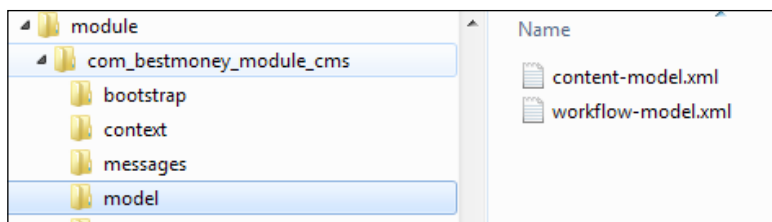
If we were to start storing Job data as process variables we would eventually end up duplicating the variable definitions as we would also have to define the Job data in the Alfresco workflow content model.

Each task is associated with a type in the workflow model. This type defines all the data or variables that the task will manage. We might define `jobStatus` as a process variable but we would also have to map it to the corresponding content model variable such as `bmw:jobStatus`.

This is mostly what we need for the Work subprocess definition. Let's move on to define a workflow content model and document content model to support this workflow definition.

Work process—workflow content model

In the `3340_10_Code\bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\module\com_bestmoney_module_cms\model` directory we have two model definition files called `content-model.xml` and `workflow-model.xml`:



They contain the domain document content model and the domain workflow content model. These two content models could be defined in the same file but it is easier to keep track of what model definitions have to do with document classification and what model definitions have to do with workflows, if we keep two files.

To define the supporting types for the Work subprocess open up the `workflow-model.xml` file.

When defining a new model we always start by importing the Alfresco namespaces that we are going to need when defining our new model. The `d` namespace is used when you set the data type (for example, `d:text`) for a property. The `cm` namespace is used when you refer to objects in the Alfresco content model, for example, `cm:person`. And the `bpm` namespace is used when we define our workflow model (for example, when defining a new task it extends, `bpm:workflowTask`):

```
<model name="bmw:workflowModel"
  xmlns="http://www.alfresco.org/model/dictionary/1.0">
  <imports>
    <import uri="http://www.alfresco.org/model/dictionary/1.0"
      prefix="d"/>
    <import uri="http://www.alfresco.org/model/content/1.0"
      prefix="cm"/>
    <import uri="http://www.alfresco.org/model/bpm/1.0"
      prefix="bpm"/>
  </imports>
</model>
```

After importing the Alfresco namespaces we also import the existing Best Money document content namespace (`bmc`). Then we define a new Best Money workflow namespace (`bmw`) to be used for all workflow-related model definitions. They have the following meaning:

- `bmw`: Workflow Content Model Namespace that will contain all types, aspects, properties, and associations that have to do with the workflows that we are implementing. Types and aspects from this model are never applied to any documents/files in the repository (located in the `workflow-model.xml` file).

- **bmc:** Document Content Model Namespace that is used to define types, aspects, and properties that are used by both the workflow tasks and are also applied to documents for classification purpose. In our case, we have defined the **bmc:job** aspect in this namespace (located in the **content-model.xml** file).

Here is how this should look:

```
<import uri="http://www.bestmoney.com/model/content/1.0"
        prefix="bmc"/>
</imports>
<namespaces>
    <namespace uri="http://www.bestmoney.com/model/workflow/1.0"
        prefix="bmw"/>
</namespaces>
```

After the namespace imports definitions, we define all the types we are going to need. For the workflows we need one type per task. Because these task types will have pretty much the same properties, it is a good idea to define some base types first with the general data that is going to be needed for all task types. In that way you do not have to duplicate properties in every task, and it is easy to add stuff to all tasks:

```
<types>
    <type name="bmw:baseJobTask">
        <parent>bpm:workflowTask</parent>
        <overrides>
            <property name="bpm:packageActionGroup">
                <default>add_package_item_actions</default>
            </property>
        </overrides>
        <mandatory-aspects>
            <aspect>bmc:job</aspect>
            <aspect>bmw:job</aspect>
            <aspect>bmw:assigneeApprover1Person</aspect>
            <aspect>bmw:assigneeApprover2Person</aspect>
            <aspect>bmw:assigneeApprover3Person</aspect>
        </mandatory-aspects>
    </type>
    <type name="bmw:baseAssignJobTask">
        <parent>bmw:baseJobTask</parent>
        <mandatory-aspects>
            <aspect>bpm:assignee</aspect>
        </mandatory-aspects>
    </type>
```

```

<type name="bmw:baseWorkTask">
  <parent>bmw:baseAssignJobTask</parent>
  <properties>
    <property name="bmw:workType">
      <title>Current work type such as for example
        Concept</title>
      <type>d:text</type>
    </property>
  </properties>
</type>

```

We can now add a type for each task and extend the generic base types that we just defined:

```

<type name="bmw:W01_ProduceWorkTask">
  <parent>bmw:baseWorkTask</parent>
</type>
<type name="bmw:W02_ValidateWorkTask">
  <parent>bmw:baseWorkTask</parent>
</type>
<type name="bmw:W03_ValidateWorkTask">
  <parent>bmw:baseWorkTask</parent>
</type>
<type name="bmw:W04_AmendWorkTask">
  <parent>bmw:baseWorkTask</parent>
  <overrides>
    <property name="bpm:packageItemActionGroup">
      <default>edit_package_item_actions</default>
    </property>
  </overrides>
</type>
</types>

```

Finally, add a new aspect called `bmw:job` that will contain data needed only during the workflow execution (that is, the data that will not be used for document classification; document classification data is contained in the `bmc:job` aspect):

```

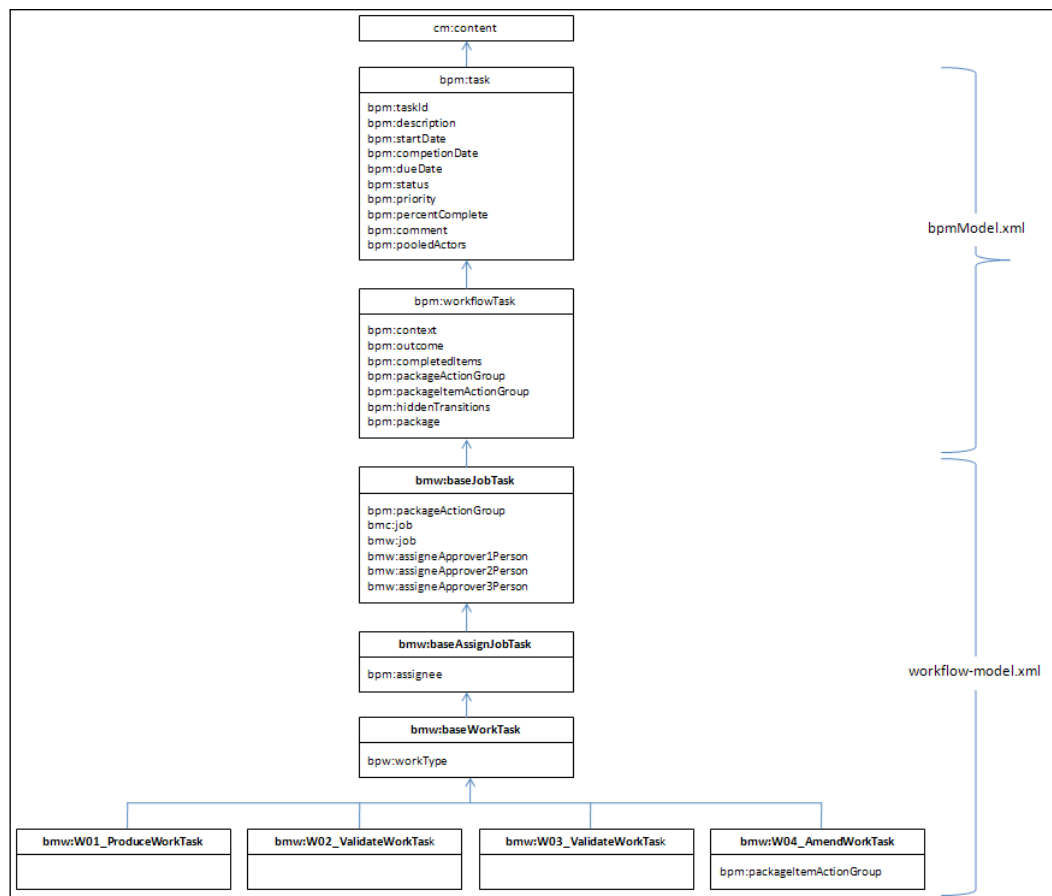
<aspects>
  <aspect name="bmw:job">
    . . .
  </aspect>
</aspects>
</model>

```

When defining the workflow content model it is quite important to know what is available in the Alfresco models. Specifically, the workflow-related model (`bpm`) as that is what we are currently interested in. If you have not done so already, you should download the Alfresco SDK (http://wiki.alfresco.com/wiki/Alfresco_SDK). To see what types, aspects, and so on are available in the `bpm` model open up the `bpmModel.xml` file located in the `sdk\lib\server\config\alfresco\model` directory.

You might have been thinking that a `priority` property could come in handy for the tasks and why did we not define it. A `priority` property and other properties such as `description`, `startDate`, `completionDate`, `dueDate`, `status`, and so on are already available in the `bpm:task` definition. So it is really important to study the Alfresco models first. In this way, we do not create duplicate definitions of stuff that is already defined and available.

Our workflow model has an inheritance hierarchy that looks like this:



So basically, we have created one type for each task in the workflow definition. And these types extend the base Work subprocess task type `bmw:baseWorkTask` that contains current work type `bmw:workType` that will be set when the Work subprocess is initiated.

Then this type extends the base assign job task `bmw:baseAssignJobTask` that contains the `bpm:assignee` property, which corresponds to the worker swimlane. The worker swimlane is an association to a person and can be used to assign a user to a task. In our case, it will point to the person that will complete the `W01_ProduceWork` task and optionally, the `W04_AmendWork` task. The `bpm:assignee` value will also be passed into the Work subprocess from the parent process.

The `W02_ValidateWorkTask` is associated with the `studioTeamManagers` swimlane, which is a pooled actor's swimlane. So it does not really have to inherit from the `bmw:baseAssignJobTask` as it does not need the `bpm:assignee` property and is instead backed by the `bpm:pooledActors` property of the `bpm:task`. The same goes for the `W03_ValidateWorkTask` that also would not have to extend the base assign Job task as it is associated with the `jobOwner` swimlane that will be defined in the parent process as an initiator swimlane and is handled in a special way. However, it makes the diagram much clearer to have all four tasks extend the same type.



The user who starts a workflow will be available in a special swimlane called `initiator`.

The last type we extend is the base job task type `bmw:baseJobTask` and this type will be used by all task types that we define in all our workflows. It contains the `bmc:job` aspect with job information that is used to classify documents and it contains the `bmw:job` aspect that contains workflow-related job information. The `bmc:job` aspect is defined in the **content-model.xml** file as it is used for document classification. The base job task also contains the three approver aspects used in the sign-off subprocess.

We can also see that a property from the Alfresco BPM model has been overridden. We have overridden the `bpm:packageActionGroup` property in the `bmw:baseJobTask` to make it possible to add documents via the task dialogs in Alfresco Explorer. By default we can only read the document that was used to start the workflow.

Documents that are managed by a workflow instance are available via the `bpm:package` property of the `bpm:workflowTask`. Further on, in the `bmw:W04_AmendWorkTask` type we have overridden the `bpm:packageItemActionGroup` property so each document attached to the workflow instance is editable/updatable via the Amend Work task dialog.

The last thing we need to define are the three aspects for the approvers used in the sign-off subprocess. They are defined as aspects with associations; start off by defining the director-level approval person association as follows:

```
<aspects>
  <aspect name="bmw:assigneeApprover1Person">
    <associations>
      <association name="bmw:assigneeApprover1Person">
        <source>
          <mandatory>false</mandatory>
          <many>false</many>
        </source>
        <target>
          <class>cm:person</class>
          <mandatory>false</mandatory>
          <many>false</many>
        </target>
      </association>
    </associations>
  </aspect>
```

Because the mandatory property is set to false, this person field will not be mandatory in the property sheet UI that we will define. Define the second-level approval person as follows (it is the same configuration as for the previous one):

```
<aspect name="bmw:assigneeApprover2Person">
  <associations>
    <association name="bmw:assigneeApprover2Person">
      <source>
        <mandatory>false</mandatory>
        <many>false</many>
      </source>
      <target>
        <class>cm:person</class>
        <mandatory>false</mandatory>
        <many>false</many>
      </target>
    </association>
  </associations>
</aspect>
```

The third and lowest level of approval person is defined as follows; note that it is defined as mandatory by setting the `mandatory` field to `true`, which means that the Job Owner has to assign a user to the Approver Level 3 property to be able to complete the task that is used to set the approvers:

```
<aspect name="bmw:assigneeApprover3Person">
  <associations>
    <association name="bmw:assigneeApprover3Person">
      <source>
        <mandatory>true</mandatory>
        <many>false</many>
      </source>
      <target>
        <class>cm:person</class>
        <mandatory>true</mandatory>
        <many>false</many>
      </target>
    </association>
  </associations>
</aspect>
</aspects>
```

Setting the `mandatory` property to `true` will also mean that an asterisk (*) is displayed in the Alfresco Explorer UI to denote that this property is mandatory.

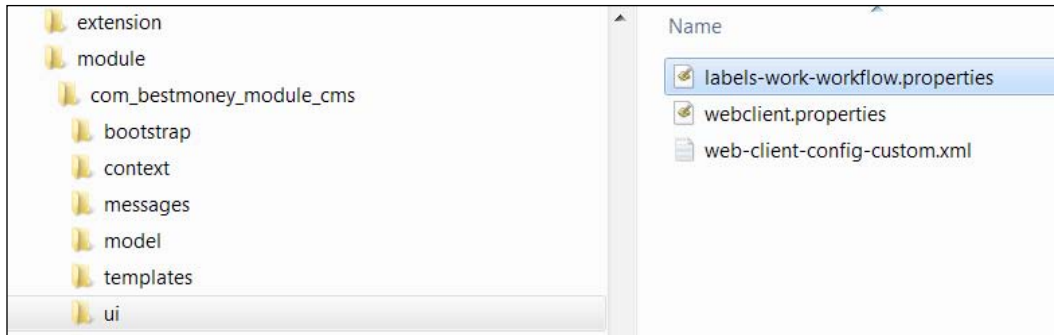
The `many` property can be used to control a "many-to-many" relationship, but in our case, we just need one approver per approval level and Job so this property is set to `false` for all definitions.

Work process—property files for UI labels

When the Alfresco task dialogs are displayed in Alfresco Explorer, the labels for the task property fields need to be specified somewhere. This is where `i18n` property files come into the picture. We are going to set up a property file for the Work process and it will contain labels for things like the workflow name and description, the reject and approve transition names, and task names and descriptions. The values for these properties are used in Alfresco Explorer in the following places:

- Starting advanced workflow dialog
- Task details dialog
- Task lists in to-do dashlets

We will create one property file for each process definition so it is easy to maintain (compared to one big property file containing all properties for all workflows). For the Work process create a property file called `labels-work-workflow.properties` in the UI directory as shown:



We will use a naming convention for these property files as follows:

`labels-{process name}-workflow.properties`

You do not need to follow this naming convention but it will make life easier if you have some kind of system when you name files for all the workflows. We will define a Spring Bean later that loads this property file. For now, add the following workflow definition-related properties to it:

```
#
# Best Money Marketing Workflows
# Properties for the Generic Work subprocess.
#

# Workflow Definition strings
bmw_workProcess.workflow.title=Generic Work
bmw_workProcess.workflow.description=Do the actual Studio Work
bmw_workProcess.node.W02_ValidateWork.transition.approve.
    title=Approve
bmw_workProcess.node.W02_ValidateWork.transition.approve.
    description=Approve Completed Work
bmw_workProcess.node.W02_ValidateWork.transition.reject.title=Reject
bmw_workProcess.node.W02_ValidateWork.transition.reject.
    description=Reject Completed Work
bmw_workProcess.node.W03_ValidateWork.transition.approve.
    title=Approve
bmw_workProcess.node.W03_ValidateWork.transition.approve.
    description=Approve Completed Work
```

```

bmw_workProcess.node.W03_ValidateWork.transition.reject.
  title=Reject
bmw_workProcess.node.W03_ValidateWork.transition.reject.
  description=Reject Completed Work

```

We have now got labels for the workflow name itself and the different transitions that are part of the workflow. The workflow model-related properties also need to be defined as follows:

```

# Workflow Model strings
bmw_workflowModel.type.bmw_W01_ProduceWorkTask.title=W01-Produce Work
bmw_workflowModel.type.bmw_W01_ProduceWorkTask.description=
  Create material in the Studio
bmw_workflowModel.type.bmw_W02_ValidateWorkTask.title=
  W02-Validate Completed Work
bmw_workflowModel.type.bmw_W02_ValidateWorkTask.description=
  Validate or reject completed work
bmw_workflowModel.type.bmw_W03_ValidateWorkTask.title=
  W03-Validate Completed Work
bmw_workflowModel.type.bmw_W03_ValidateWorkTask.description=
  Validate or reject completed work
bmw_workflowModel.type.bmw_W04_AmendWorkTask.title=W04-
  Amend Completed Work
bmw_workflowModel.type.bmw_W04_AmendWorkTask.description=
  Amend completed work after it has been rejected

```

These property names are going to look rather confusing at first. But if you look at them you can see that the property names actually refer to both the workflow definition (that is, `bmw_workProcess...`) and the workflow model (`bmw_workflowModel...`).

The Alfresco workflow property naming convention can be specified as something like this:

```

{namespace from workflow model}_{process name from jPDL}.(workflow|node|type)[.{node name}|{type name}][.transition.{transition name}].(title|description)={value}

```

So, the following example sets the title for the Work process in the `bmw` namespace:

```

bmw_workProcess.workflow.title=Generic Work

```

And the following example sets the title for the `reject` transition leading out from the `W03_ValidateWork` node in the `bmw` namespace (this will set the button name to **Reject** in the **Task Details** dialog):

```

bmw_workProcess.node.W03_ValidateWork.transition.reject.title=Reject

```

The name and description for a task is set by using the type defined for the task in the workflow model. The following example shows how to set the description for the W01_ProduceWork task:

```
bmw_workflowModel.type.bmw_W01_ProduceWorkTask.description=  
    Create material in the Studio
```

In the workflow model you would normally find scope definitions belonging to a certain namespace by using a colon (:), as in the following example:

```
<type name="bmw:W01_ProduceWorkTask">
```

However, in the property file you cannot use colons as part of a property name but have to instead use an underscore in place of them.

Work process—using dynamic descriptions and setting task due date

The Work process is a generic process and when we set the description of, for example, the W01_ProduceWork task in the property file it is just set to W01-Produce Work — this does not tell the end user what kind of work should be produced. It would be nice to set the description dynamically via the parent process. Current work type is passed in from the parent process and available in the `bmw_workType` variable. So it would be good to be able to update the description with this value before it is displayed.

To do this we have to intercept the process before the task is displayed and change the task description. We can do this by listening to a `task-create` event and when it happens we set the description. At the same time as we do this, we can also take the opportunity to set the due date for the task so it is displayed correctly in the **My Tasks ToDo** dashlet:

```
<task-node name="W01_ProduceWork">  
  <task name="bmw:W01_ProduceWorkTask" swimlane="worker">  
    <event type="task-create">  
      <script>  
        <variable name="bmw_workType" access="read"/>  
        <variable name="bpm_dueDate" access="read"/>  
        <expression>  
          taskInstance.description =  
            "W01-Produce Work (" + bmw_workType + ")";  
          if (bpm_dueDate != null)  
            taskInstance.dueDate = bpm_dueDate;  
        </expression>  
      </script>  
    </event>  
  </task>  
</task-node>
```

```

        </script>
      </event>
    </task>
    <transition to="W02_ValidateWork"></transition>
  </task-node>

```

In this BeanShell script, we first specify that we want to access the value of the current work type and the due date variables. Then we use the `taskInstance` variable (which is always accessible) to set the task description and the task due date (note that we use `_` instead of `:` when referring to variables such as `bmw_workType`). Make sure to check the due date before we set it for the task instance, it would cause a process exception if it was set to null.

We can set the description for the other three tasks in the same way. See source code for this chapter for an example.



Setting the due date does not mean that the user will receive a reminder e-mail or a new task notification in the UI. It is just information for the user and for any manager looking at a list of tasks.

Work process—defining the job data

So far, we have not talked about what properties are part of the `bmc:job` aspect or the `bmw:job` aspect. First of all, why is the job information split up into two different aspects? This has to do with how it is used. Part of the Job information such as Campaign ID and related Product is going to be used to classify the material (that is, documents and files) that are produced via the Job workflow, and will be stored in the `bmc:job` aspect. Other Job information is needed just during the execution of the workflow, such as Job status and Work types, and will be stored in the `bmw:job` aspect.

Let's start with the properties for the `bmc:job` aspect—update the job aspect in the `content-model.xml` file with the following properties:

Property name	Data type	Mandatory?	Default	Multiple?	Constraint
<code>bmc:campaignId</code>	<code>d:text</code>	true	-	-	-
<code>bmc:product</code>	<code>d:text</code>	true	Credit Card	-	<code>bmc:product_options</code>
<code>bmc:jobType</code>	<code>d:text</code>	true	-	-	<code>bmc:jobType_options</code>

The Job Owner will associate each Job with a marketing campaign and one of Best Money's products. A job type will be specified telling the studio what type of material should be produced, for example, e-mail, logo, advert, web page, and so on.

Now add properties to the `bmw:job` aspect, update the job aspect in the `workflow-model.xml` file with the following properties:

Property name	Data type	Mandatory?	Multiple?	Constraint
<code>bmw:jobStatus</code>	<code>d:text</code>	false	-	<code>bmw:jobStatus_options</code>
<code>bmw:briefSignOffCategory</code>	<code>d:text</code>	false	-	<code>bmw:signOffCategory_options</code>
<code>bmw:productionSignOffCategory</code>	<code>d:text</code>	false	-	<code>bmw:signOffCategory_options</code>
<code>bmw:workTypes</code>	<code>d:text</code>	true	true	<code>bmw:workType_options</code>
<code>bmw:conceptWorkDueDate</code>	<code>d:datetime</code>			
<code>bmw:designWorkDueDate</code>	<code>d:datetime</code>			
<code>bmw:copywriteWorkDueDate</code>	<code>d:datetime</code>			

The status of a job will be kept in the `jobStatus` property, so management reports can be created with status for all ongoing campaign jobs. Both the Job Brief and the produced Job Material is signed off and there are two properties that the Job owner can use to specify what level of sign-off is required for the brief and for the material.

The Work type is used to tell the studio what type of work is needed – either concept, design, or copywriter (or a combination). Due dates are required for each work type and have to be specified separately as there is no way of defining a composite property such as Work Info (that would contain work type and work due date).

Work process—task property sheets

For a Workflow task to be displayed with the correct properties in the Alfresco Explorer UI, we have to define a property sheet for it. For more information about property sheets see *Chapter 9, Content Model Definition Solutions*.

The first thing that we want to do is fix it so all documents and files that have been produced via a Marketing production workflow will show the `bmw:job` properties when the **Details View** page is displayed in Alfresco Explorer (Note: This aspect will be set on all produced material at the end of the Job process).

All files that we update and add configuration to in this section are located in the 3340_10_Code\bestmoney\alf_extensions\trunk_alfresco\config\alfresco\module\com_bestmoney_module_cms\ui directory.

Open up the web-client-config-custom.xml file and add the following property sheet:

```
<config evaluator="aspect-name" condition="bmc:job">
  <property-sheet>
    <separator name="sep-1" display-label-id="jobDataHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmc:campaignId"
      display-label-id="campaignId" read-only="true"/>
    <show-property name="bmc:product" display-label-id="product"
      read-only="true"/>
    <show-property name="bmc:jobType" display-label-id="jobType"
      read-only="true"/>
  </property-sheet>
</config>
```

Make sure to make each property read-only as the users are not supposed to edit these properties.

This property sheet uses some label properties that we need to define too. Open up the webclient.properties file and add the following properties:

```
# bmc:job
#
jobDataHeader=Marketing Production Material
campaignId=Camapign Id
product=Associated Product
jobType=Job Type
```

Now, for the property sheets related to the Work process tasks, we will use a separate web client configuration file called web-client-config-work-workflow.xml. Here, we also use a naming convention so property sheets for each workflow are kept in their own file:

Web-client-config-{process name}-workflow.xml

Define the property sheet for the W01 task first:

```
<config evaluator="node-type" condition="bmw:W01_ProduceWorkTask">
  <property-sheet>
    <separator name="sep-1" display-label-id="generalJobDesc"
      component-generator="HeaderSeparatorGenerator"/>
```



```
<show-property name="bmw:jobStatus" display-label-id="jobStatus"
  read-only="true"/>
<separator name="sep-2"
  display-label-id="workAndPriorityHeader"
  component-generator="HeaderSeparatorGenerator"/>
<show-property name="bmw:workType"
  display-label-id="workType" read-only="true"/>
<show-property name="bpm:priority"
  display-label-id="workPriority"/>
<show-property name="bpm:dueDate"
  display-label-id="workDueDate"/>
</property-sheet>
</config>
```

Because it is types that we are defining property sheets for, we need to use the `node-type` evaluator. The property sheet is associated with the type by setting the condition. Then we can display any property in that type or subtype.

The `bmw:jobStatus` and the `bmw:workType` properties have been defined as *read-only* as nobody should be able to change them after they have been set by the Job Owner in the parent Job process.

Define the next property sheet for the `W02` task as follows:

```
<config evaluator="node-type"
  condition="bmw:W02_ValidateWorkTask">
  <property-sheet>
    <separator name="sep-1" display-label-id="generalJobDesc"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus"
      display-label-id="jobStatus" read-only="true"/>
    <separator name="sep-2" display-label-id="workAndPriorityHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:workType"
      display-label-id="workType" read-only="true"/>
    <show-property name="bpm:priority"
      display-label-id="workPriority"/>
    <show-property name="bpm:dueDate"
      display-label-id="workDueDate"/>
    <show-association name="bpm:assignee"
      display-label-id="worker" read-only="true"/>
  </property-sheet>
</config>
```

The validate tasks also display the bpm:assignee property, so they can see who was producing the work. When the work is validated, the bpm:assignee cannot be changed and is set to *read-only*. The other validate task looks similar:

```
<config evaluator="node-type" condition="bmw:W03_ValidateWorkTask">
  <property-sheet>
    <separator name="sep-1"
      display-label-id="generalJobDesc"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus"
      display-label-id="jobStatus" read-only="true"/>
    <separator name="sep-2"
      display-label-id="workAndPriorityHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:workType"
      display-label-id="workType" read-only="true"/>
    <show-property name="bpm:priority"
      display-label-id="workPriority"/>
    <show-property name="bpm:dueDate"
      display-label-id="workDueDate"/>
    <show-association name="bpm:assignee"
      display-label-id="worker" read-only="true"/>
  </property-sheet>
</config>
```

The last property sheet that we need for the Work process tasks is for the W04 task that is used when the work is not approved and needs to be amended:

```
<config evaluator="node-type" condition="bmw:W04_AmendWorkTask">
  <property-sheet>
    <separator name="sep-1"
      display-label-id="generalJobDesc"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus"
      display-label-id="jobStatus" read-only="true"/>
    <separator name="sep-2"
      display-label-id="workAndPriorityHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:workType"
      display-label-id="workType" read-only="true"/>
    <show-property name="bpm:priority"
      display-label-id="workPriority"/>
    <show-property name="bpm:dueDate"
      display-label-id="workDueDate"/>
  </property-sheet>
</config>
```

These property sheets do not contain any property display information for the `bmc:job` aspect because their properties will automatically be displayed, as all task types have the `bmc:job` aspect applied.

We can see here that the four property sheet definitions look almost the same. So it would have been good to be able to for example, inherit a base property sheet definition and then just add any extra properties. This is not possible and we cannot really use just one property sheet for the `bmc:job` aspect as there are other properties in the `bpm` namespace that we also want to display for the task.

We also want to display the task properties divided into two sections. So in most cases we are going to end up defining lots of property sheets with almost the same configurations in them.



There is a way to reuse property sheet configurations and that is to use property sheet definitions for aspects instead of types. If we have a simpler property sheet that we want to display for a couple of tasks, and we want to display the same properties for each task, then we can just define the properties in an aspect (which we usually do any way) and then define a property sheet for that aspect. However, when we have properties from several aspects in a type and some properties defined directly in the task, then it is usually necessary to define a new property sheet per type to get what we want.

Finally, we need to add the `label` properties used by the property sheets to a UI property file. We will add a new UI property file specifically for Workflow property sheet labels. Create a file called `webclient-workflow.properties` and put it in the `ui` directory. Then add the following `label` properties to it:

```
#
# Best Money Marketing - Workflow property sheet labels
#
jobWorkflowHeader=General information about this job
jobStatus=Job Status
workAndPriorityHeader=Work Item and Priority
workType=Work Type
workPriority=Work Priority
workDueDate=Work Due Date
worker=Worker
```

This file will be loaded by some Spring bootstrap configuration, which we are going to do next.

Work process—bootstrapping UI property files and property sheets configuration

The label property files and the property sheet configuration file need to be bootstrapped so Alfresco knows about them when displaying the different dialogs and windows in Alfresco Explorer. This is done in the bootstrap-context.xml

Spring configuration file located in the 3340_10_Code\bestmoney\alf_extensions\trunk_alfresco\config\alfresco\module\com_bestmoney_module_cms\context directory.

1. Add the following bean configuration for the Work process properties that has to do with labels for workflow definition and workflow types:

```
<bean id=
    "com.bestmoney.marketing.properties.workflowBootstrap"
    parent="workflowDeployer">
    <property name="labels">
        <list>
            <value>alfresco.module.com_bestmoney_module_cms.ui.
                labels-work-workflow
            </value>
        </list>
    </property>
</bean>
```

2. Then update the bean, which bootstraps resource bundles with general workflow properties for property sheet labels:

```
<bean id=
    "com.bestmoney.webclient.properties.webResourceBundles"
    class="org.alfresco.web.app.ResourceBundleBootstrap">
    <property name="resourceBundles">
        <list>
            <value>alfresco.module.com_bestmoney_module_cms.ui.webclient
            </value>
            <value>alfresco.module.com_bestmoney_module_cms.ui.
                webclient-workflow</value>
        </list>
    </property>
</bean>
```

3. And finally, bootstrap the Work process task property sheets:

```
<bean id="com.bestmoney.webclient.configBootstrap"
      class="org.alfresco.web.config.WebClientConfigBootstrap"
      init-method="init">
  <property name="configs">
    <list>
      <value>classpath:alfresco/module/com_bestmoney_module_cms/
        ui/web-client-config-custom.xml
      </value>
      <value>classpath:alfresco/module/com_bestmoney_module_cms/
        ui/web-client-config-work-workflow.xml</value>
    </list>
  </property>
</bean>
```

Work process—testing it

So now we are ready to test this new Work process. However, this process is designed to be part of a bigger parent process so we need a way of testing it without depending on information being passed in from the parent process. A couple of changes to it are necessary before we can test it standalone. Create a copy of the processdefinition.xml file located in the 3340_10_Code\bestmoney\alf_extensions\trunk_alfresco\config\alfresco\module\com_bestmoney_module_cms\workflows\work directory and name it processdefinition_test.xml.

Then update the first swimlane part of the Work processes definition in this copy as follows:

```
<swimlane name="initiator"/>

<swimlane name="jobOwner">
  <assignment class=
    "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
    <actor>#{people.getPerson('jobOwner')}</actor>
  </assignment>
</swimlane>

<swimlane name="studioTeamManagers">
  <assignment class=
    "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
    <pooledactors>#{people.getGroup(
      'GROUP_STUDIO_TEAM_MANAGERS')}
    </pooledactors>
  </assignment>
</swimlane>
```

```

<swimlane name="worker">
  <assignment class=
    "org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
    <actor>#{people.getPerson('worker')}</actor>
  </assignment>
</swimlane>

```

So, what we have done here is changed the swimlane assignments so that they are independent of any values passed in from the parent Job process. We also added a special initiator swimlane that is necessary to have when the process is to be run standalone – the initiator refers to the person that started the process instance.

Now, update the start node so that it contains a start task and initialization of necessary variables:

```

<start-state name="Start">
  <task name="bmw:startTestProcessTask" swimlane="initiator"></task>
  <event type="node-leave">
    <action class=
      "org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
      <script>
        executionContext.setVariable("bmc_campaignId", "Campaign 1");
        executionContext.setVariable("bmc_jobType", "Poster");
        executionContext.setVariable("bmw_jobStatus", "Start Up");
        executionContext.setVariable("bmw_workTypes", "[Concept]");
        executionContext.setVariable("bmw_workType", "Concept");
        executionContext.setVariable("bmw_assigneeApprover3Person",
          people.getPerson("admin").getNodeRef());
        executionContext.setVariable("bpm_assignee",
          people.getPerson("worker").getNodeRef());

        var procId = executionContext.processInstance.getId();
        executionContext.setVariable("procId", procId);
      </script>
    </action>
  </event>
  <transition to="W01_ProduceWork"></transition>
</start-state>

```

A workflow that is not a subworkflow always needs a start task that kicks things off. So that is why we have added the `bmw:startTestProcessTask` to the node definition. Then we changed to using `AlfrescoJavaScript` as we would like to use the root object `people` to get to `Person` nodes. Note that we assign the node reference as a string to the assignee variables and that is how it is stored in the jBPM database.

We need to set up all variables here that have been defined as mandatory in the content model. If the variable has a constraint (for example, `bmc_jobType`) then we must set a value that is valid for the constraint. If the variable is defined as *multiple* then we need to enclose the value in `[value1,value2...]` to indicate that it is an array (for example, `bmw_workTypes`).

The `startTestProcessTask` task needs to be backed by a type defined in the Workflow Content Model. So we have added the following definition to the `workflow-model.xml` file:

```
<type name="bmw:startTestProcessTask">
  <parent>bpm:startTask</parent>
</type>
```

Note that this start task extends another task type called `bpm:startTask` while all the other task types extend the `bpm:workflowTask`.

Now, we can build and deploy this new workflow. Stop Alfresco and run the Ant target, `deploy-alfresco-amp` and that should build the Best Money AMP and deploy it into the Alfresco installation that we have configured in `build.properties`.

Start Alfresco and log in as administrator so that you can get to the Administration Console. For this test, we are using two new users and one new group, which we need to add to the repository. Via the **Manage System Users** screen add the following users:

Name	User Name	Home Space
Worker Worksson	worker	/Company Home/User Homes/worker
Studio Team Manager 1	studiomgr1	/Company Home/User Homes/studiomgr1
Job Owner	jobOwner	/Company Home/User Homes/jobOwner

And via the **Manage System Groups** screen, add the following group and member:



It's now time to deploy the new workflow. We do this via the **Workflow Console** that can be accessed via the `http://localhost:8080/alfresco/faces/jsp/admin/workflow-console.jsp` URL. Note that we need to be logged in as admin to access this console.

In the input field type in the following command:

```
deploy alfresco/module/com_bestmoney_module_cms/workflows/work/
processdefinition_test.xml
```

And then click on the **Submit** button. You should see something like the following if the deployment went okay.

```
deployed definition id: jbpms$8 , name: jbpms$bmw:workProcess , title:
Generic Work , version: 1
```

```
definition: jbpms$8 , name: Generic Work , version: 1
```

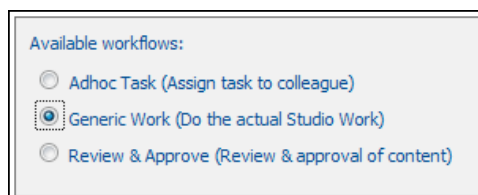
```
workflow: None
```

```
path: None
```



Sometimes, you need to click on the **Submit** button twice for it to react and deploy the workflow.

The workflow definition is now ready to be used. What we now need is just a document from which to start the workflow. So upload some document and then select **Start Advanced Workflow** from the drop-down pop-up menu:



In the first screen of the **Start Advanced Workflow** wizard select the new workflow that we just deployed. Then click on **Next** and then **Next** again, followed by clicking on the **Finish** button to start the workflow. This creates a new `W01_ProduceWork` task and assigns it to the worker user that we just created. If we log in as with the worker username and password, we will see a new task in the **My Tasks To Do** dashlet:

My Tasks To Do			
Description	Type	Id	Created
W01-Produce Work (Concept)	W01-Produce Work	8	20 August 2010 14:16

We can see that the description is correct with the **Concept** part being added. To complete this task, and move the workflow instance forward, click on the description to open the **Task** dialog screen:

Manage Task: W01-Produce Work
Create material in the Studio

From resource file labels-work-workflow.properties

Task Properties

Owner: worker

General information about this job

- Campaign Id: Campaign 1
- Associated Product: Credit Card
- Job Type: Poster
- Job Status: Start Up

Work Item and Priority

- Work Type: Concept
- Work Priority: 3
- Work Due Date: None

Resources Can be accessed via the bpm:package variable

Name	Description	Path	Created	Modified	Actions
CIFS access from Mac.docx		/Company Home/test	15 August 2010 14:17	20 August 2010 14:16	

Page 1 of 1

Add Resource

Controlled by bpm:packageActionGroup property definition in bmc:baseJobTask in workflow-model.xml

Controlled by default value for bpm:packageItemActionGroup property but overriding in bmc:W04_AmendWorkTask to allow editing

Save Changes Task Done Cancel

The preceding screenshot of the **Task** dialog has been updated with information about where the different texts and available actions come from. Clicking on the **Task Done** button will complete the task and the `W02_validateWorkTask` will be created and assigned to all Studio Managers that are part of the `STUDIO_TEAM MANAGERS` group, which is currently just one manager called `studiomgr1`. Log in as Studio Team Manager 1, the Dashboard appears to be empty and there is no task in the **My Tasks To Do** dashlet:

My Tasks To Do

No tasks found.

This is because this task is a pooled task and will be assigned to every user that is a member of the `STUDIO_TEAM MANAGERS` group. And these pooled tasks are available via another dashlet called **My Pooled Tasks**. Click on the **Configure** link on the dashboard to add this dashlet. Then complete the task by clicking on the **Approve** button in the **Task** dialog:



The **Reject** and **Approve** buttons are the result of the possible transitions that we defined in the workflow's definition. And the button labels come from the resource definitions in the `labels-work-workflow.properties` file.

If there are a lot of Studio Managers around, it is probably a good idea to first click on the **Take Ownership** button, which will move this task over to the Studio Manger's **My Tasks To Do** dashlet and remove it from all other Studio Managers' **My Pooled Tasks** dashlets.


 If a user does not first take ownership of a pooled task, and just completes it at the same time as another user does, then the outcome will be the result of the actions of the user who first completes the transaction.

When this task is completed, a new task `W03_ValidateWorkTask` is created and assigned to the `jobOwner` user. Approve this task and the workflow instance is completed. In one of the validate tasks we could also choose to *reject* the work – this would mean that the workflow would go back to the `W04_AmendWork` task, where the worker can update the material and then submit it again for validation.

If a user wants to see completed tasks he or she can do that by using the **My Completed Tasks** dashlet. We have now designed, implemented, and tested a workflow and are ready to move on to more complex scenarios with a parent process calling subprocesses.

For more information on how the workflow instance is actually executed and what controls the different execution paths read on. A **workflow definition** represents a formal specification of a business process and is based on a *directed graph*. The graph is composed of nodes and transitions between them. Every node in the graph is of a specific type.

The type of the node defines the runtime behavior. A workflow definition has exactly one start state. When a new advanced workflow is started, a workflow instance is created based on the latest deployed version of the process definition. A so-called *token* represents one path of execution in an active workflow instance. A **token** is the runtime concept that maintains a pointer to a node in the graph.

A **workflow instance** is one execution of a workflow definition. When a workflow instance is created, a token is created for the main path of execution. This token is called the root token of the process instance and it is positioned in the start state of the process definition.

A signal instructs a token to continue graph execution. When receiving an unnamed signal, the token will leave its current node over the default leaving transition. When a transition-name (for example, **Approve**) is specified in the signal, the token will leave its node over the specified transition. A signal given to the workflow instance is delegated to the root token.

After the token has entered a node, the node is executed. Nodes themselves are responsible for the continuation of the graph execution. Continuation of graph execution is done by making the token leave the node. Each node type can implement a different behavior for the continuation of the graph execution. A node that does not propagate execution will behave as a state.

Running the work process from the Alfresco Share UI

Now when we have got the generic work process up and running with Alfresco Explorer it would be good to know how much effort it is to get this workflow going in Alfresco Share. As we know, Alfresco Share is the new UI client that will supersede the Alfresco Explorer UI client.

From version 3.4a and onwards Alfresco Share supports custom advanced workflows, before that the available workflows for Alfresco Share were hardcoded. Now with the newer versions, any deployed workflow is visible in Alfresco Share and can be started from Alfresco Share.

We don't even have to configure any property sheets or forms as they are called in Alfresco Share. There will be a default layout for all tasks including the start task. So let's try this out.

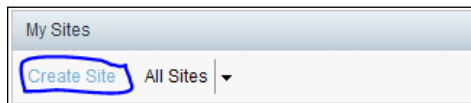
Log in to Alfresco Share (<http://localhost:8080/share>) as the admin user. This is a sharing and collaboration environment where we create sites that are then used to share documents, wiki pages, blogs, event calendars, and so on.

In Alfresco Share, we can start a workflow without the need for a document. The **My Tasks** dashlet contains the **Start Workflow** link that can be used for this:

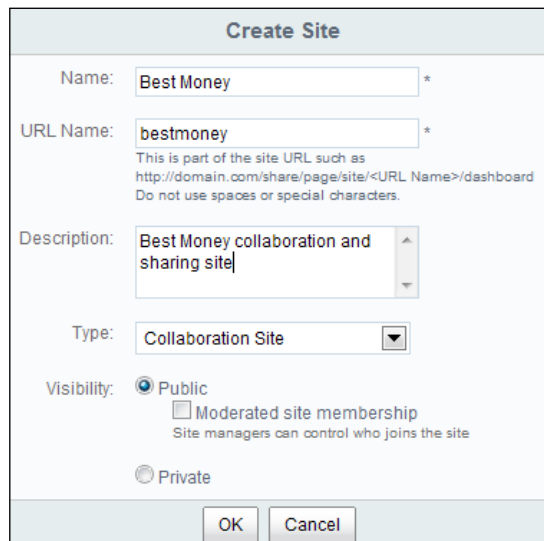


We are not going to start the workflow directly like this but it is useful to know that we can start workflows without content. Instead, let's create a Best Money site, add a document to its document library, and start a workflow from the document.

Start by clicking on the **Create Site** link in the **My Sites** dashlet as depicted in the following screenshot:



In the next dialog, enter site information as per the following screenshot:



Create Site

Name: *

URL Name: *

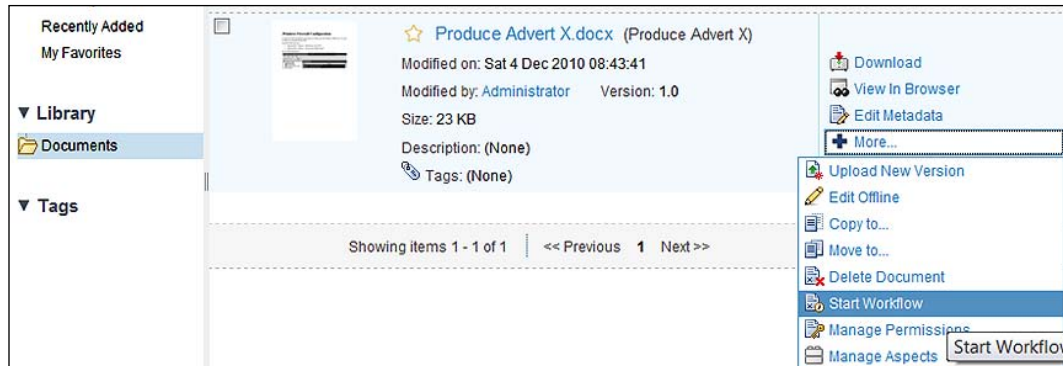
This is part of the site URL such as
<http://domain.com/share/page/site/<URL Name>/dashboard>
Do not use spaces or special characters.

Description:

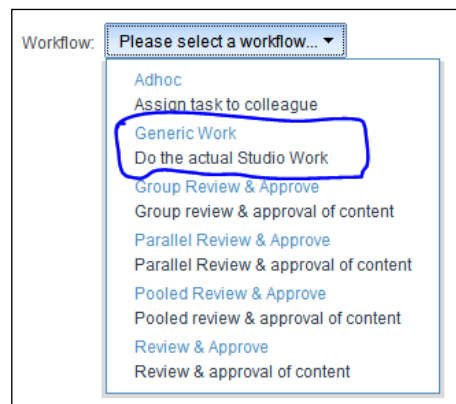
Type:

Visibility: ☒ Public
☐ Moderated site membership
Site managers can control who joins the site
☐ Private

Then click on **OK** to create the site. Now click on the **Document Library** link and add a document to the library via the **Upload** button. To start a workflow for this document click on the **More... | Start Workflow** (to the right-hand side of the screen) link, as shown in the following screenshot:



This brings up a dialog where we can select from the deployed workflows:



We can see here that the **Generic Work** workflow has been picked up and the text labels are working.

It is quite easy to hide any workflow that we do not want to see in this dialog. This is done via configuration in `share-config-custom.xml`.

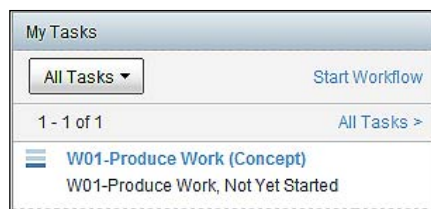
For example, the WCM workflows are hidden like this:



```
<config evaluator="string-compare"
condition="Workflow">
  <hidden-workflows>
    <workflow name="jbpm$wcmwf:*/>
    <workflow name="jbpm$wf:articleapproval"/>
    <workflow name="jbpm$inwf:invitation-
nominated"/>
    <workflow name="jbpm$imwf:invitation-
moderated"/>
  </hidden-workflows>
  <hidden-tasks>
    <task type="wcmwf:*/>
  </hidden-tasks>
</config>
```

Now click on the link for the **Generic Work** workflow. This brings up the form for the start task and it has got some default fields such as **Comment**, **Description**, **Due Date**, **Status**, **Priority**, and so on. This form can be customized as we will see in a bit. For now, click on the **Start Workflow** button at the bottom of the form.

This creates the W01 task for the `worker` user and if we log in to Alfresco Share as this user we will see a new task in the **My Tasks** dashlet:



We can see here too that the labels for the task description and the name have been picked up automatically from the previous configurations that we have done. Click on the **W01-Produce Work (Concept)** link to open the task form:

Workflow Assignee: *

Worker Svensson (worker)

Select

bmw.assigneeApprover1Person:

Select

bmw.assigneeApprover2Person:

Select

bmw.assigneeApprover3Person: *

Administrator (admin)

Select

Description:

W01-Produce Work (Concept)

Job Type: *

Poster

Unique campaign identifier: *

Campaign 1

We can see that the task form does not look very appealing and this is the default layout that we get if we do not customize the task form for a task – the properties are just listed as the system finds them.

We can see though that constraints for things like Job Type have been picked up and the combo boxes are loaded with the correct options and the selected values have been set. Also, labels for Job type, Workflow assignee, and so on have been picked up correctly. However, the form will show all properties that we definitely do not want. So we have got to find some way of customizing the layout.

We can do that by defining a form layout in the `share-config-custom.xml` configuration file located in the `3340_10_Code\bestmoney\alf_extensions\trunk\_share\config\alfresco\web-extension` directory. Open it up and add the following form configuration for the W01 task:

```
<config evaluator="task-type" condition="bmw:W01_ProduceWorkTask">
  <forms>
    <form>
```

```

<field-visibility>
  <show id="bmc:campaignId"/>
  <show id="bmc:product"/>
  <show id="bmc:jobType"/>
  <show id="bmw:jobStatus"/>
  <show id="bmw:workType"/>
  <show id="bpm:priority"/>
  <show id="bpm:dueDate"/>
  <show id="message" />
  <show id="packageItems" />
  <show id="transitions" />
</field-visibility>

```

We start off by specifying what task type the form should be for and what fields (that is, properties) should be visible. Then we move on and configure how these fields should appear in the form:

```

<appearance>
  <!-- Field sets -->
  <set id="" appearance="title" label-id="workflow.set.task.info" />
  <set id="jobWorkflowHeader" appearance="title"
    label-id="jobWorkflowHeader
    template="/org/alfresco/components/form/2-column-set.ftl" />
  <set id="workAndPriorityHeader" appearance="title"
    label-id="workAndPriorityHeader" />
  <set id="documents" appearance="title" label="Documents" />
  <set id="response" appearance="title"
    label-id="workflow.set.response" />

  <!-- Field set "" -->

```

The first thing that we configure in the **Appearance** section is fieldsets (that is, grouping of fields) and we define the same fieldsets that we used in the Alfresco Explorer property sheet, with the same label IDs. We also need to include a fieldset for the documents attached to the workflow and a fieldset for the response area with the transition buttons.

After this, we just need to define each individual field and set up what fieldset it should appear under:

```

<field id="message">
  <control template=
    "/org/alfresco/components/form/controls/info.ftl" />
</field>

```



```
<!-- Field set "jobWorkflowHeader" -->
  <field id="bmc:campaignId" label-id="campaignId"
    set="jobWorkflowHeader" read-only="true"/>
  <field id="bmc:product" label-id="product"
    set="jobWorkflowHeader" read-only="true"/>
  <field id="bmc:jobType" label-id="jobType"
    set="jobWorkflowHeader" read-only="true"/>
  <field id="bmc:jobStatus" label-id="jobStatus"
    set="jobWorkflowHeader" read-only="true"/>

<!-- Field set "workAndPriorityHeader" -->
  <field id="bmc:workType" label-id="workType"
    set="workAndPriorityHeader" read-only="true"/>
  <field id="bmc:priority" label-id="workPriority"
    set="workAndPriorityHeader" />
  <field id="bmc:dueDate" label-id="workDueDate"
    set="workAndPriorityHeader" />

<!-- Field set "documents" -->
  <field id="packageItems" set="documents" />

  <!-- Field set "response" -->
  <field id="transitions" set="response" />
</appearance>
</form>
</forms>
</config>
```

Unfortunately, the externally defined labels, which we previously defined in the `labels-work-workflow.properties` file, and the other property files, are not going to be picked up in Alfresco Share. So open up the `extension-app.properties` file located in the `3340_10_Code\bestmoney\alf_extensions\trunk\_share\config\alfresco\messages` directory and add the following properties to it:

```
jobWorkflowHeader=General information about this job
campaignId=Camapign Id
product=Associated Product
jobType=Job Type
jobStatus=Job Status
workAndPriorityHeader=Work Item and Priority
workType=Work Type
workPriority=Work Priority
workDueDate=Work Due Date
```

To deploy these custom Share configurations stop Alfresco and run the `deploy-share-jar` Ant target. Then start Alfresco again. If we now look at the task form again, it should look something like this:

Edit Task: W01-Produce Work

Reassign

* Required Fields

Info

Message: W01-Produce Work (Concept)

General information about this job

Campaign Id: *

Campaign 1

Associated Product: *

Credit Card

Job Type: *

Poster

Job Status:

Start Up

Work Item and Priority

Work Type:

Concept

Work Priority: *

3

Work Due Date:

29/12/2010

DD/MM/YYYY

Documents

Items:

Produce Advert X.docx

Description: (None)

Modified on: Sat 4 Dec 2010 08:56:43

View More Actions

Add

Response

Task Done

Save and Close

Cancel

So as we can see, Alfresco Share can now be used to drive any custom workflow that we might want to use and it is very easy to configure the task forms needed.

For more information about Alfresco Share and Advanced Workflow support have a look at this Wiki page http://wiki.alfresco.com/wiki/Custom_Share_Workflow_UI.

Summary

This chapter has given us a thorough walkthrough of how to implement a basic workflow with the embedded jBoss jBPM workflow engine and its capabilities. It is a powerful workflow system that allows us to implement very complex workflows.

We have seen how easy it is to define the process definition with the jPDL process definition language based on the Swimlane diagram that we designed in the previous chapter. We then went on to define the workflow content model that is used to define the data managed by each task. One type definition is needed for each task in the workflow definition.

The task data was displayed in the Alfresco UI by defining the so-called property sheets. One property sheet is needed for each task type that we have. The labels in each property sheet were externalized by standard property files. The property sheet file and the label file were bootstrapped into the Alfresco system via Spring configurations.

We have also learned that process definitions are versioned when deployed into Alfresco. When we start a new workflow instance it is based on the latest deployed version of the process definition.

The jBPM workflow engine is very well integrated with Alfresco and we have seen that we can directly assign users and groups from Alfresco when creating the process definition.

The Alfresco Share client from version 3.4a can now also be used to drive custom workflows and we have seen how easy it is to define task forms used by this UI client.

In the next chapter, we will continue implementing the rest of the Swimlane diagrams for the main Job process and we will see how to call subprocesses, how to implement parallel process flows, how to use JavaScript in a node, how to send e-mails, how to implement phases, and more.

11

Business Process Implementation Solutions: Part 2

In the last chapter, we saw how to implement the generic Work process based on the Swimlane diagram we designed earlier. In this chapter, we will dig deeper into the functionality that is available in the JBoss jBPM workflow engine.

In this chapter, you will learn:

- Implementing parallel flows
- Implementing subprocesses and calling them from a parent process
- Using Alfresco JavaScript in a business process
- Sending template-based e-mails from a business process
- Implementing phases with jPDL
- Adding useful workflow management Dashlets

Completing the implementation of the marketing production workflow

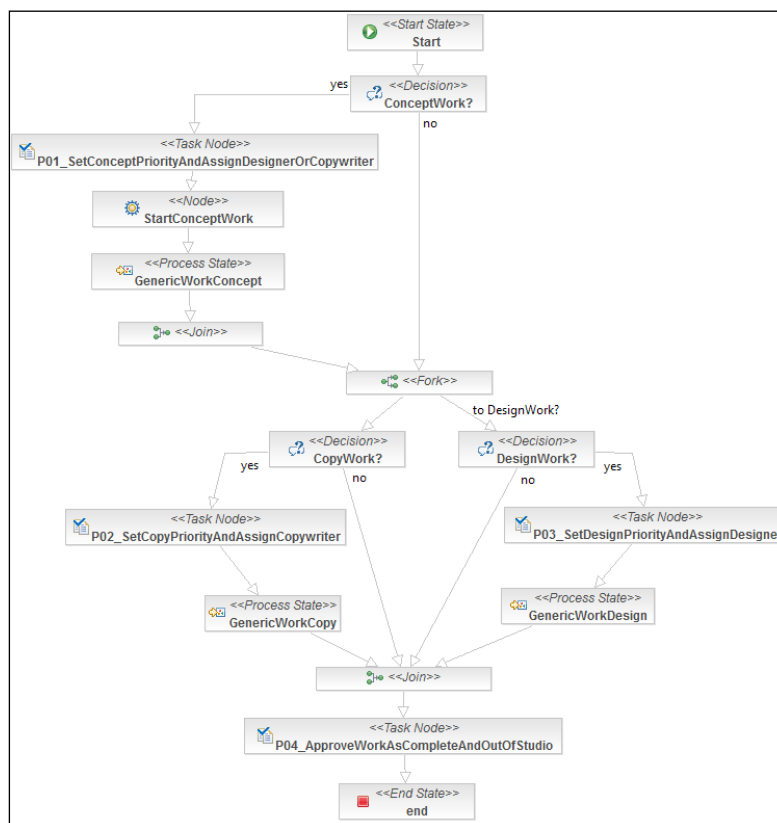
We already have the generic Work process implemented. It is used by the Studio process in several places. We will start implementing the Studio process followed by the Sign-Off process and finally tie everything together with the Job process.

Implementing the Studio subprocess

In this section, we will implement the Studio subprocess. Start up Eclipse and add a new process definition for the Studio process and call it **studio** (that is right-click on the **workflows** directory and then select **New** followed by **Other...**).

Studio process—workflow definition (jPDL)

Define a process definition graphically for the Studio subprocess. It is usually a good idea to have the Swimlane diagram printed out or on a second screen when doing this. The process diagram should look something like this when we are done:



When we implemented the Work process we used only two types of nodes — Task nodes and Decision nodes. With the Studio process we bring in a couple of other types of nodes. We will use the Fork node to split up the execution path into two concurrent parallel execution paths. The Studio Team Manager can assign more than one concept worker, so we use a Join node to bring all the concept work together before continuing with design and copywrite work.

The Copywriter and the Designer will be able to do their work at the same time in parallel. To bring those parallel paths back together into one execution path we will use the `Join` node.

The `Process State` node type is used when we want to call a subprocess such as the `Work` process. There is also a plain `Node` that we use to loop through all Studio workers that have been assigned by the Job Owner to do Concept work. We will see in a bit how this node uses a `foreach` loop to fork off one `Work` subprocess for each Studio worker that is assigned to do Concept work.

Now, set the process definition name manually to `bmw:studioProcess`. The jPDL XML Schema version also needs to be updated to 3.2. The Studio process is going to use two Swimlanes called the `jobOwner` and the `studioTeamManagers`, and the definitions for these are the same as in the `Work` process so we are not going to repeat them here, just add them to the workflow definition.

For this subprocess, we are also going to save the process ID in the `procId` variable. This is done in the `start-state` node just before we leave it. The definition is the same as in the `Work` process.

The task nodes need tasks and they in turn need to be assigned to a Swimlane. Add the task elements as follows:

```
<task-node
  name="P01_SetConceptPriorityAndAssignDesignerOrCopywriter">
  <task name=
    "bmw:P01_SetConceptPriorityAndAssignDesignerOrCopywriterTask"
    swimlane="studioTeamManagers">
  . . .
  <task-node name="P02_SetCopyPriorityAndAssignCopywriter">
    <task name="bmw:P02_SetCopyPriorityAndAssignCopywriterTask"
      swimlane="studioTeamManagers">
    . . .
    <task-node name="P03_SetDesignPriorityAndAssignDesigner">
      <task name="bmw:P03_SetDesignPriorityAndAssignDesignerTask"
        swimlane="studioTeamManagers">
      . . .
      <task-node name="P04_ApproveWorkAsCompleteAndOutOfStudio">
        <task name="bmw:P04_ApproveWorkAsCompleteAndOutOfStudioTask"
          swimlane="studioTeamManagers"></task>
```

By the look of it, here we are not using the `jobOwner` Swimlane, so why is it defined? It is defined as it needs to be passed into the `Work` subprocess, which has a task that should be assigned to the Job Owner.

There are three decision nodes in the Studio process and each one of them needs a variable to look at to be able to decide which path to take. These three variables are going to be passed in from the parent Job process where the Job Owner has decided what work to be executed as part of the material production (that is specified in the `bmw:workTypes` variable, which in turn is parsed for the value of these three variables). Let's use these variables as follows in the decision nodes:

```
<decision name="ConceptWork?">
  <transition to="fork1" name="no"></transition>
  <transition
    to="P01_SetConceptPriorityAndAssignDesignerOrCopywriter"
    name="yes">
    <condition>#{conceptWork == true}</condition>
  </transition>
</decision>
. . .
<decision name="CopyWork?">
  <transition to="join1" name="no"></transition>
  <transition to="P02_SetCopyPriorityAndAssignCopywriter" name="yes">
    <condition>#{copyWork == true}</condition>
  </transition>
</decision>

<decision name="DesignWork?">
  <transition to="join1" name="no"></transition>
  <transition to="P03_SetDesignPriorityAndAssignDesigner" name="yes">
    <condition>#{designWork == true}</condition>
  </transition>
</decision>
```

A decision node is used when the process needs to make a decision of where to continue in the process graph. There are two ways to specify the decision criteria, the simplest one is to just add `condition` element to the transition(s). Conditions are BeanShell script expressions that return a `boolean`. At runtime, the decision node will loop over its leaving transitions (in the order as specified in the XML), and evaluate each condition. The first transition for which the conditions resolve to `true` will be taken.

The other way that a decision node can calculate the decision is to use a `DecisionHandler`. Then the decision is calculated in a Java class and the selected leaving transition is returned by the `decide` method of the `DecisionHandler` implementation.

Each of the work types also has an associated due date (that is `bmw_conceptWorkDueDate`, `bmw_copywriteWorkDueDate`, and `bmw_designWorkDueDate`) that is set by the Job Owner. They should also be set up for each of the tasks in the Studio workflow definition. Look at the Work process definition for an example of how to do this. Then set up the task variable `dueDate` in the P01, P02, and P03 tasks by adding a `task-create` event handler.

Now we are coming to the `StartConceptWork` node that needs to be updated, so it kicks off the Concept work subprocess for each Studio worker assigned by the Job owner to do concept work. The selected Studio workers will be contained in the `bpm_assignees` variable when we implement the Job process. Update the node as follows:

```
<node name="StartConceptWork">
  <action class="org.alfresco.repo.workflow.jbpm.ForEachFork">
    <foreach>#{bpm_assignees}</foreach>
    <var>bpm_assignee</var>
  </action>
  <transition to="GenericWorkConcept" />
</node>
```

Here we use a special Alfresco workflow action that implements a *for loop* that can be used to walk through a collection (that is `bpm_assignees`) and then assign a variable (that is `bpm_assignee`) to each item in the collection. For each loop, the transition `GenericWorkConcept` is taken with the `bpm_assignee` and assigned the value of current item in the collection.

The `GenericWorkConcept` node is a process-state node type used to initiate a Work subprocess for each concept work that is to be done. We have to update this node with a few things before it will work. Start by adding the name of the subprocess we want to call and set the work type variable:

```
<process-state name="GenericWorkConcept">
  <sub-process name="bmw:workProcess"/>

  <event type="node-enter">
    <script>
      <variable name="bmw_workType" access="write"/>
      <expression>
        bmw_workType = "Concept";
      </expression>
    </script>
  </event>
```


The sub-process name should match the process-definition name specified for the subprocess. We cannot specify a specific version of the subprocess definition, the latest one deployed will be used, and this is also true when we run a normal process without any subprocesses. We set the `bmw_workType` variable to `Concept` so we can set up task description correctly in the Work subprocess. This makes it possible for the worker to differ between concept, copywrite, and design work tasks.



To run a process instance with a previous version of a process definition, we would have to redeploy the previous version of the definition first.

The next thing we need to do before the subprocess call will work properly is to specify what variables we want to pass into the subprocess from the parent process. We can leave this out and the parent process should pass all variables into the subprocess instance. However, this is not usually what we want and sometimes we want to map a variable to a different name in the subprocess.

As soon as we start specifying what variables should be passed into the subprocess, we have to think about all variables that are going to be needed in the subprocess as none of the variables will be passed in automatically in this case. We can divide the variables we pass into the subprocess into three categories:

- **Outbound** – These are variables that will be set up by the subprocess and passed back up to the parent process.
- **Inbound** – These are variables that have been set by the parent process and then passed into the subprocess and used by it.
- **Unused Inbound** – These are variables that have been set by the parent process and then passed into the subprocess, but are never used by the subprocess. However, they are mandatory variables that are needed to be able to complete tasks in the subprocess.

We are going to need one outbound variable to keep the process ID for the Work subprocess that is to be executed as part of the Studio process. Add it as follows:

```
<variable name="studioConceptWorkProcId" access="write" mapped-  
name="procId"/>
```

Outbound variables are specified by using the `access` attribute and setting it to `write`. The outbound variable `studioConceptWorkProcId` has a different name in the subprocess and we specify that with the `mapped-name` attribute. Next, we specify all the inbound variables that are going to be used by the Work subprocess:

```
<variable name="bmc_campaignId" access="read" />  
<variable name="bmc_product" access="read" />  
<variable name="bmc_jobType" access="read" />
```

```

<variable name="bmw_jobStatus" access="read"/>
<variable name="bmw_workTypes" access="read" />
<variable name="bmw_workType" access="read" />
<variable name="bmw_conceptWorkDueDate" access="read" mapped-
  name="bpm_dueDate"/>
<variable name="bpm_package" access="read"/>
<variable name="bpm_assignee" access="read"/>
<variable name="bpm_priority" access="read"/>
<variable name="jobOwner" access="read"/>
<variable name="studioTeamMgrsGroupName" access="read"/>

```

The last couple of inbound variables that we need to pass into the work subprocess are not really used by it in any way, but they cannot be null, as in the workflow model they are defined as mandatory. A task cannot be completed if there are mandatory variables that are null:

```

<variable name="bmw_briefSignOffCategory" access="read"/>
<variable name="bmw_productionSignOffCategory" access="read"/>
<variable name="bmw_assigneeApprover3Person" access="read"/>

```

Okay, that is all we need to pass into the Concept Work subprocess. When we call the Copy and Design Work subprocesses, there are only a few things that differ in the definitions. The due date is passed in from another variable and the bpm_assignee is set from a fixed variable (that is not from the bpm_assignees collection). So we can copy the sub-process, event, and variables from the GenericWorkConcept node definition into both the GenericWorkCopy and GenericWorkDesign node definitions. Then change the following for the GenericWorkCopy node:

- `bmw_workType = "Concept" => bmw_workType = "Copy"`
- `bmw_conceptWorkDueDate => bmw_copywriteWorkDueDate`
- `<variable name="bpm_assignee" access="read"/> => <variable name="bmw_assigneeCopywriter" access="read" mapped-name="bpm_assignee"/>`

And for the GenericWorkDesign node change the following:

- `bmw_workType = "Concept" => bmw_workType = "Design"`
- `bmw_conceptWorkDueDate => bmw_designWorkDueDate`
- `<variable name="bpm_assignee" access="read"/> => <variable name="bmw_assigneeDesigner" access="read" mapped-name="bpm_assignee"/>`

This is it; we are now finished with the definition of the Studio subprocess.

Studio process—workflow content model

To define the supporting types for the Studio subprocess, open up the `workflow-model.xml` file. Define the new task types to support the workflow task node definitions as follows:

```
<type
  name="bmw:P01_SetConceptPriorityAndAssignDesignerOrCopywriterTask">
  <parent>bmw:baseAssignMultipleJobTask</parent>
</type>
<type name="bmw:P02_SetCopyPriorityAndAssignCopywriterTask">
  <parent>bmw:baseJobTask</parent>
  <mandatory-aspects>
    <aspect>bmw:assigneeCopywriter</aspect>
  </mandatory-aspects>
</type>
<type name="bmw:P03_SetDesignPriorityAndAssignDesignerTask">
  <parent>bmw:baseJobTask</parent>
  <mandatory-aspects>
    <aspect>bmw:assigneeDesigner</aspect>
  </mandatory-aspects>
</type>
<type name="bmw:P04_ApproveWorkAsCompleteAndOutOfStudioTask">
  <parent>bmw:baseJobTask</parent>
</type>
```

The P01 task type uses a new base task that allows for multiple assignments. Define it as follows:

```
<type name="bmw:baseAssignMultipleJobTask">
  <parent>bmw:baseJobTask</parent>
  <mandatory-aspects>
    <aspect>bpm:assignees</aspect>
  </mandatory-aspects>
</type>
```

The P02 and P03 tasks go on in parallel so we cannot use the `bpm_assignee` for the worker person assignment. We need to define two new variables that will hold the copywriter person and the designer person (you could also use the existing `bpm_assignee` and just define one more variable, but our approach is clearer). Define the `bmw:assigneeCopywriter` aspect and the `bmw:assigneeDesigner` aspect in the same way we did defined the `bmw:assigneeApprover1Person` aspect.

Studio process—property files for UI labels

For the Studio process UI label resources create a new property file called `labels-studio-workflow.properties` and put it in the `com_bestmoney_module_cms/ui` directory. Then add the following properties to it:

```
bmw_studioProcess.workflow.title=Studio
bmw_studioProcess.workflow.description=Do the Studio work
bmw_workflowModel.type.bmw_P01_
SetConceptPriorityAndAssignDesignerOrCopywriterTask.title=P01-Set
Concept Priority and Assign Worker(s)
bmw_workflowModel.type.bmw_P01_SetConceptPriorityAndAssignDesignerOr
CopywriterTask.description=Set concept work priority and assign
designer and/or copywriter
bmw_workflowModel.type.bmw_P02_SetCopyPriorityAndAssignCopywriterTask
.title=P02-Set Copy Priority and Assign Copywriter
bmw_workflowModel.type.bmw_P02_SetCopyPriorityAndAssignCopywriterTask
.description=Set copy work priority and assign copywriter
bmw_workflowModel.type.bmw_P03_SetDesignPriorityAndAssignDesignerTask
.title=P03-Set Design Priority and Assign Designer
bmw_workflowModel.type.bmw_P03_SetDesignPriorityAndAssignDesignerTask
.description=Set design work priority and assign designer
bmw_workflowModel.type.bmw_P04_ApproveWorkAsCompleteAndOutOfStudio
Task.title=P04-Approve Studio Work
bmw_workflowModel.type.bmw_P04_ApproveWorkAsCompleteAndOutOfStudio
Task.description=Approve studio work as completed and out of studio
```

Studio process—task property sheets

For the property sheets related to the Studio process tasks, we will use a separate web client configuration file called `web-client-config-studio-workflow.xml`.

The Studio process property sheets are very much like the ones we used for the Work process. The thing that differs is what is being displayed under the second header separator called `priorityAndWorkerAssigneeHeader`. There we display the priority, due date, and assignee(s) fields, so the Studio Manager can fill those in.

The property sheet for the P01 task looks like this:

```
<config evaluator="node-type" condition=
  "bmw:P01_SetConceptPriorityAndAssignDesignerOrCopywriterTask"
  replace="true">
  <property-sheet>
    <separator name="sep-1" display-label-id="jobWorkflowHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus" display-label-id="jobStatus"
      read-only="true"/>
```

```
<separator name="sep-2" display-label-  
  id="priorityAndWorkerAssigneeHeader" component-  
  generator="HeaderSeparatorGenerator"/>  
<show-property name="bpm:priority" display-label-  
  id="workPriority"/>  
<show-property name="bmw:conceptWorkDueDate" display-label-  
  id="conceptWorkDueDate" />  
<show-association name="bpm:assignees" display-label-  
  id="workers"/>  
</property-sheet>  
</config>
```

The bpm:assignees association will enable the Studio manager to assign one or more users to complete work. We use the bmw:conceptWorkDueDate property from the bmw:job aspect to keep the due date for concept work. You might be thinking that we could have used the already available bpm:dueDate property, but we want to keep more than one due date so we need to define more due date properties.

The next property sheet is for the P02 task and looks almost identical to the previous one:

```
<config evaluator="node-type" condition="bmw:P02_  
SetCopyPriorityAndAssignCopywriterTask" replace="true">  
  <property-sheet>  
    <separator name="sep-1" display-label-id="jobWorkflowHeader"  
      component-generator="HeaderSeparatorGenerator"/>  
    <show-property name="bmw:jobStatus" display-label-id="jobStatus"  
      read-only="true"/>  
    <separator name="sep-2" display-label-  
      id="priorityAndWorkerAssigneeHeader" component-  
      generator="HeaderSeparatorGenerator"/>  
    <show-property name="bpm:priority" display-label-  
      id="workPriority"/>  
    <show-property name="bmw:copywriteWorkDueDate" display-label-  
      id="copywriteWorkDueDate" />  
    <show-association name="bmw:assigneeCopywriter" display-label-  
      id="worker"/>  
  </property-sheet>  
</config>
```

The only thing that differs is that here we collect the due date for copywrite work with the bmw:copywriteWorkDueDate property and we input the copywriter person via the bmw:assigneeCopywriter association. The next property sheet is used to collect data for the design work specified by task P03:

```
<config evaluator="node-type" condition="bmw:P03_  
SetDesignPriorityAndAssignDesignerTask" replace="true">  
  <property-sheet>
```

```

    <separator name="sep-1" display-label-id="jobWorkflowHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus" display-label-id="jobStatus"
      read-only="true"/>
    <separator name="sep-2" display-label-
      id="priorityAndWorkerAssigneeHeader" component-
      generator="HeaderSeparatorGenerator"/>
    <show-property name="bpm:priority" display-label-
      id="workPriority"/>
    <show-property name="bmw:designWorkDueDate" display-label-
      id="designWorkDueDate" />
    <show-association name="bmw:assigneeDesigner" display-label-
      id="worker"/>
  </property-sheet>
</config>

```

The last property sheet for the Studio process should present the Studio work that has been used as an input by the previous tasks, so it can be approved. All properties are therefore read-only:

```

<config evaluator="node-type"
  condition="bmw:P04_ApproveWorkAsCompleteAndOutOfStudioTask"
  replace="true">
  <property-sheet>
    <separator name="sep-1" display-label-id="jobWorkflowHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:jobStatus" display-label-id="jobStatus"
      read-only="true"/>
    <separator name="sep-2" display-label-id="workItemsHeader"
      component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:workTypes" display-label-id="workTypes"
      read-only="true"/>
    <show-property name="bmw:conceptWorkDueDate" display-label-
      id="conceptWorkDueDate" read-only="true"/>
    <show-property name="bmw:designWorkDueDate" display-label-
      id="designWorkDueDate" read-only="true"/>
    <show-property name="bmw:copywriteWorkDueDate" display-label-
      id="copywriteWorkDueDate" read-only="true"/>
  </property-sheet>
</config>

```

Finally, we need to add the label properties used by the property sheets to a UI property file. Open up the `webclient-workflow.properties` file and add the following label properties to it:

```
priorityAndWorkerAssigneeHeader=Priority and Worker assignee(s)
conceptWorkDueDate=Concept Work Due Date
designWorkDueDate=Design Work Due Date
copywriteWorkDueDate=Copy Work Due Date
workers=Workers
workTypes=Work Types
workItemsHeader=Work items
```

In the next section, we will see how the property sheet file and associated UI label files are loaded via Spring configuration.

Studio process—bootstrapping UI property files and property sheets configuration

The workflow bootstrapping beans now need to be updated with the two new files that we have created for the Studio process. This is done in the `bootstrap-context.xml` Spring configuration file located in the `com_bestmoney_module_cms\context` directory.

Update the following bean configuration to include the Studio process properties:

```
<bean id="com.bestmoney.marketing.properties.workflowBootstrap"
  parent="workflowDeployer">
  <property name="labels">
    <list>
      <value>
        alfresco.module.com_bestmoney_module_cms.ui.labels-work-
        workflow</value>
      <value>
        alfresco.module.com_bestmoney_module_cms.ui.labels-studio-
        workflow</value>
    </list>
  </property>
</bean>
```

Then bootstrap the Studio process task property sheets:

```
<bean id="com.bestmoney.webclient.configBootstrap"
  class="org.alfresco.web.config.WebClientConfigBootstrap"
  init-method="init">
  <property name="configs">
    <list>
```

```

    <value>classpath:alfresco/module/com_bestmoney_module_cms/ui/
      web-client-config-custom.xml</value>
    <value>classpath:alfresco/module/com_bestmoney_module_cms/ui/
      web-client-config-work-workflow.xml</value>
    <value>classpath:alfresco/module/com_bestmoney_module_cms/ui/
      web-client-config-studio-workflow.xml</value>
  </list>
</property>
</bean>

```

Studio process—testing it

So now we are ready to test this new Studio process. However, this process is also designed to be part of a bigger parent process, so we need a way of testing it without depending on information being passed in from the parent process. A couple of changes to it are necessary before we can test it standalone. Create a copy of the `processdefinition.xml` file located in the `com_bestmoney_module_cms\workflows\studio` directory and name it `processdefinition_test.xml`. Then update the first Swimlane part of the Studio processes definition in this copy as follows:

```

<swimlane name="initiator"/>
  <swimlane name="jobOwner">
    <assignment
      class="org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
      <actor>#{people.getPerson('jobOwner')}</actor>
    </assignment>
  </swimlane>
  <swimlane name="studioTeamManagers">
    <assignment
      class="org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
      <pooledactors>
        #{people.getGroup('GROUP_STUDIO_TEAM_MANAGERS')}
      </pooledactors>
    </assignment>
  </swimlane>

```

These are exactly the same changes as we did for the Work process test definition. Update the start node so it contains a start task and initialization of the necessary variables:

```

<start-state name="Start">
  <task name="bmw:startTestProcessTask" swimlane="initiator"></task>
  <event type="node-leave">
    <action
      class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">

```



```
<script>
    executionContext.setVariable("bmc_campaignId", "Campaign 1");
    executionContext.setVariable("bmc_jobType", "Poster");
    executionContext.setVariable("bmw_jobStatus", "Start Up");
    executionContext.setVariable("bmw_workTypes",
        "[Concept,Design]");
    executionContext.setVariable("conceptWork", true);
    executionContext.setVariable("designWork", true);
    executionContext.setVariable("studioTeamMgrsGroupName",
        "GROUP_STUDIO_TEAM_MANAGERS");
    executionContext.setVariable("jobOwner",
        people.getPerson("jobOwner").getNodeRef());
    executionContext.setVariable("bmw_assigneeApprover3Person",
        people.getPerson("admin").getNodeRef());
    var procId = executionContext.processInstance.getId();
    executionContext.setVariable("procId", procId);
</script>
</action>
</event>
<transition to="ConceptWork?"></transition>
</start-state>
```

There are quite a few differences to the Work process start node, as we can see. We do not set the `bmw_workType`, as it is set by the Studio process, and we do not set the `bpm_assignee` (that is worker), as this is also set by the Studio process. We have added a work type Design so we can test with more than one work type. The `bmw_workTypes` collection is split up into separate Boolean variables in the Job process, so we need to set them (that is `conceptWork` and `designWork`) as well.

We also set the `studioTeamMgrsGroupName` as it is needed by the Studio process. Finally, the `jobOwner` variable is needed as we are not using the default initiator user that is normally assigned automatically to the user who starts a workflow.

Now we can build and deploy this new workflow. Stop Alfresco and run the ant target `deploy-alfresco-amp` and that should build the Best Money AMP and deploy it into the Alfresco installation that we have configured in `build.properties`.

It's now time to deploy the new Studio workflow. However, we need to deploy the Work process first as currently the test version of the Work process is deployed. So deploy it as follows:

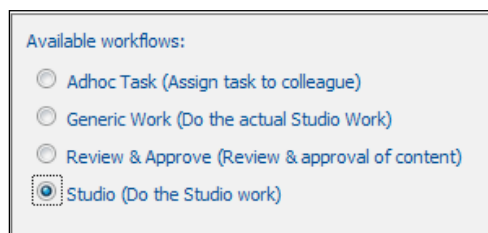
```
deploy alfresco/module/com_bestmoney_module_cms/workflows/work/
processdefinition.xml
```

And then deploy the Studio test process as follows:

```
deploy alfresco/module/com_bestmoney_module_cms/workflows/studio/
processdefinition_test.xml
```

Note that the order in which these workflow definitions are deployed is very important. If we were to deploy the Studio workflow definition first, it would use the test version of the Work process, even if we deployed a newer version of the Work process afterwards.

The Studio test workflow definition is now ready to be used. What we now need is just a document from which to start the workflow. So upload some document and then select **Start Advanced Workflow** from the drop-down pop up menu:



In the first screen of the Start Advanced Workflow wizard, select the new **Studio** workflow that we just deployed. Then click **Next** and then **Next** again, followed by clicking on the **Finish** button to start the workflow. This creates a new P01_SetConceptPriorityAndAssignDesignerOrCopywriter task and assigns it to every member of the STUDIO_TEAM MANAGERS group. Log in with the studiomgr1 user and you will see the following task in the **My Pooled Tasks** Dashlet:

My Pooled Tasks			
Description	Type	Id Created	
 P01-Set Concept Priority and Assign Worker(s)	P01-Set Concept Priority and Assign Worker(s)	17	22 August 2010 10:25

To complete this task, and move the workflow instance forward, click on the description to open the task dialog screen:

Here, the Studio Manager can select the Concept work priority, due date, and the workers that should complete the Concept.

 If we do not see the Campaign Id, Associated Product, and Job Type properties then we might have to specify them manually for each Studio task property sheet. Sometimes Alfresco picks up the `bmc:job` property sheet and other times it does not. In most cases, we have to duplicate the specification for these properties in every task property sheet.

Now select the worker user that we previously created and choose a due date for the task. Then complete the task. This should kick off the Work subprocess and if we log in as the worker user we should see that it has been assigned the `W01_ProduceWork` task, and the due date is set properly to the date that was selected by the Studio Manager. Let the worker complete this task that creates the `W02` task and assigns it to Studio Manager 1. When the `W02` task is completed the `W03` task is created and assigned to the Job Owner.

When the Job Owner completes the P03 task the P03 task is created where the Studio Manager is supposed to assign the Designer, set due date, and priority. That kicks off a new Work subprocess for the Design and when that is finished the Studio Manager will assign the last task P04 off the Studio process. When P04 is completed the Studio process is also completed.



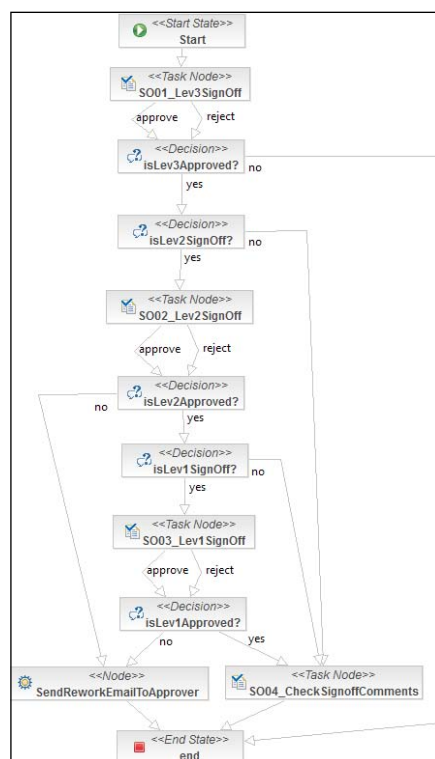
Note that if we change the Work process we also have to redeploy the Studio process so it picks up the new Work process.

Implementing the Sign-off subprocess

The last sub-process that is used by the Job process is the Sign-off process. Start up Eclipse and add a new process definition for the Sign-off process and call it **signoff**.

Sign-off process—workflow definition (jPDL)

Define a process definition graphically for the Sign-off subprocess. It is usually a good idea to have the Swimlane diagram printed out or on a second screen when doing this. The process diagram should look something like this when we are done:



The sign-off workflow definition does not use any new node types; we are familiar with all of them. Worth noticing is that the two Send Email to Approver nodes in the Swimlane diagram have been replaced by one node in the jPDL diagram. Sometimes it is possible to pass in variables to a node instead of having two almost duplicate nodes.

This process will use a new feature that is available to us and that is sending e-mails from the workflow process. Whenever a level 1 or level 2 approver rejects a marketing brief or marketing production (that is the sign-off process is used by the Job process in two places) a rework e-mail is sent to the previous approver to let him or her know that it was not signed off in the end.

Now set the process definition name manually to `bmw:signoffProcess`. The jPDL XML Schema version also needs to be updated to 3.2. The Studio process is going to need a `jobOwner` Swimlane, as defined for the Studio and Work processes, and one Swimlane for each approver. The approver Swimlanes all look like this:

```
<swimlane name="lev1Approver">
  <assignment
    class="org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
    <actor>#{bmw_assigneeApprover1Person}</actor>
  </assignment>
</swimlane>
```

The only thing that differs between the approver Swimlanes are the numbers 1, 2, or 3. The approvers are assigned by the Job Owner when the marketing job is created.

For this subprocess, we are also going to save the process ID in the `procId` variable. This is done in the `start-state` node just before we leave it. The definition is the same as in the Studio and Work process.

The task nodes need tasks and they in turn need to be assigned to a Swimlane. Add task elements for all task nodes in the Sign-off process. When the task elements are added, define a `task-create` event that sets the task description. We also need a variable to keep track of the approver's decision (that is approve or reject), initialize it in a `task-enter` event. The following shows the updated `SO01_Level3SignOff` node:

```
<task-node name="SO01_Lev3SignOff">
  <event type="node-enter">
    <script>
      <variable name="lev3Approved" access="write"/>
      <expression>
        lev3Approved = false;
      </expression>
    </script>
  </event>
```

```

<task name="bmw:S001_Lev3SignOffTask" swimlane="lev3Approver">
  <event type="task-create">
    <script>
      <variable name="bmw_signOffPhase" access="read"/>
      <expression>
        taskInstance.description = "S003-Sign-off Phase (" +
          bmw_signOffPhase + ")";
      </expression>
    </script>
  </event>
</task>
<transition to="isLev3Approved?" name="reject"></transition>
<transition to="isLev3Approved?" name="approve">
  <script>
    <variable name="lev3Approved" access="read,write"/>
    <expression>
      lev3Approved = true;
    </expression>
  </script>
</transition>
</task-node>

```

By dynamically updating the task description the approver can see what phase he or she is signing off (that is marketing brief or produced material). The `bmw_signOffPhase` is set by the Job process depending on what is being signed off.

The `S002_Level2SignOff` and `S003_Level1SignOff` nodes should look the same as the `S001_Level3SignOff` node, so they will not be displayed here. After each sign-off node there is a decision node that checks if it was approved or rejected, update it as follows:

```

<decision name="isLev3Approved?">
  <transition to="end" name="no"></transition>
  <transition to="isLev2SignOff?" name="yes">
    <condition>#{lev3Approved == true}</condition>
  </transition>
</decision>

```

After Level 2 and Level 1 approval, the decision nodes also need to set up a couple of variables in case the approver rejected the job:

```

<decision name="isLev2Approved?">
  <transition to="SendReworkEmailToApprover" name="no">
    <script>
      <variable name="rejectPerson" access="write"/>
      <variable name="sendRejectEmailToPerson" access="write"/>
    </script>
  </transition>
</decision>

```

```
<expression>
    rejectPerson = bmw_assigneeApprover2Person;
    sendRejectEmailToPerson = bmw_assigneeApprover3Person;
</expression>
</script>
</transition>
<transition to="isLev1SignOff?" name="yes">
    <condition>#{cat2Approved == true}</condition>
</transition>
</decision>
```

In case the job was rejected, we set up who did the rejection and who was the previous approver. This information will be used when sending the rework e-mail in case of a rejection. The `isLev1Approved` decision node looks the same as the `isLev2Approved` node.

Before Level 2 and Level 1 sign-off, we need to check if the particular level of sign-off has been selected by the Job Owner. This information is kept in the `signOffLevel` variable, update the nodes as follows:

```
<decision name="isLev2SignOff?">
    <transition to="CheckSignoffComments" name="no"></transition>
    <transition to="Lev2SignOff" name="yes">
        <condition>#{signOffLevel == "LEV2" or signOffLevel == "LEV1"}</condition>
    </transition>
</decision>
. . .
<decision name="isLev1SignOff?">
    <transition to="S004_CheckSignoffComments" name="no"></transition>
    <transition to="S003_Lev1SignOff" name="yes">
        <condition>#{signOffLevel == "LEV1"}</condition>
    </transition>
</decision>
```

When we get to the `S004_CheckSignOffComments` node, we consider the job as successfully signed off and set the `signedOff` variable to `true`. We also add the task and it should be completed by the Job Owner:

```
<task-node name="S004_CheckSignoffComments">
    <event type="node-enter">
        <script>
            <variable name="signedOff" access="write"/>
            <expression>
                signedOff = true;
            </expression>
        </script>
    </event>
</task-node>
```

```

    </script>
  </event>
  <task name="bmw:S004_CheckSignoffCommentsTask" swimlane="jobOwner">
    <event type="task-create">
      <script>
        <variable name="bmw_signOffPhase" access="read"/>
        <expression>
          taskInstance.description = "S004-Check Sign-off (" +
            bmw_signOffPhase + ")";
        </expression>
      </script>
    </event>
  </task>
  <transition to="end"></transition>
</task-node>

```

The last thing we need to sort out for the sign-off process is the `SendReworkEmailToApprover` node that will send the e-mail to the previous approver in case of a rejection. Add a node-enter event as follows:

```

<node name="SendReworkEmailToApprover">
  <event type="node-enter">
    <action
      class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
      <script>
        var template = companyhome.childByNamePath("Data
          Dictionary/Email Templates/Best Money/Marketing/signoff-
          rework-email.ftl");
        var args = [];
        args["signOffPhase"] = bmw_signOffPhase;
        args["campaignId"] = bmc_campaignId;
        args["rejectorPersonNodeRef"] =
          rejectPerson.nodeRef.toString();
        var txtMail =
          bpm_package.children[0].processTemplate(template, args);
        var mail = actions.create("mail");
        mail.parameters.to =
          sendRejectEmailToPerson.properties.email;
        mail.parameters.from = "alfresco@bestmoney.com";
        mail.parameters.subject = "The " + bmw_signOffPhase + " for
          the " + bmc_campaignId + " campaign has been rejected";
        mail.parameters.text = txtMail;
        mail.execute(bpm_package);
      </script>
    </action>
  </event>
  <transition to="end"></transition>
</node>

```


The first thing we do in the script is fetching the e-mail template that we are going to use. It will be stored in the /Company Home/Data Dictionary/Email Templates/Best Money/Marketing folder and called signoff-rework-email.ftl.



The companyhome root variable that usually is available without us having to do anything will not be available in a subprocess. It is important that we do not forget to pass it in from the parent Job process. This is because we are specifically setting what variables should be used in the subprocess and this also includes Alfresco JavaScript root variables.

Next we set up the parameters that will be passed into the template when we process it via one of the documents in the content package that is associated with the workflow instance (that is bmp_package). To send the e-mail we use an Alfresco mail action.

Sign-off process—workflow content model

By now we are quite familiar with how to add supporting workflow task types to the Workflow Content Model. Add the following types to the workflow-model.xml file:

```
<type name="bmw:S001_Lev3SignOffTask">
  <parent>bmw:baseSignOffTask</parent>
</type>
<type name="bmw:S002_Lev3SignOffTask">
  <parent>bmw:baseSignOffTask</parent>
</type>
<type name="bmw:S003_Lev3SignOffTask">
  <parent>bmw:baseSignOffTask</parent>
</type>
<type name="vmmw:S004_CheckSignoffCommentsTask">
  <parent>bmw:baseSignOffTask</parent>
</type>
```

All these task types use a new base task type called `bmw:baseSignOffTask` that has an extra property to store the sign-off phase that we are in, add it as follows:

```
<type name="bmw:baseSignOffTask">
  <parent>bmw:baseJobTask</parent>
  <properties>
    <property name="bmw:signOffPhase">
      <title>Current sign-off phase such as for example Brief</title>
      <type>d:text</type>
    </property>
  </properties>
</type>
```

The `bmw:signOffPhase` property will be used in the description of sign-off tasks and in the property sheets, so users know which phase of the material production they are actually signing off.

Sign-off process—create and bootstrap the e-mail template

The e-mail template that is used by the `SendReworkEmailToApprover` node needs to be specified and registered in a way so it is loaded automatically. Create the `signoff-rework-email.ftl` Freemarker template file and put it in the `com_bestmoney_module_cms\templates` directory. Open it up and add the following text as the template:

```
<#assign signOffPhase=args["signOffPhase"]/>
<#assign campaignId=args["campaignId"]/>
<#assign rejectorPersonNodeRef=args["rejectorPersonNodeRef"]/>
<#assign rejectorPerson=companyhome.nodeByReference[rejectorPersonNodeRef]/>
```

```
The ${signOffPhase} for the ${campaignId} campaign has been
rejected by ${rejectorPerson.properties.firstName} ${rejectorPerson.
properties.lastName}.
```

```
Best Money
```

We also need to bootstrap the template so it is loaded automatically into the `/Company Home/Data Dictionary/Email Templates/Best Money/Marketing` folder. For this create a file called `signoff-rework-email-template.xml` and put it in the `com_bestmoney_module_cms\bootstrap` directory. Add the following bootstrapping XML to it:

```
<?xml version="1.0" encoding="UTF-8"?>
<view:view xmlns:view="http://www.alfresco.org/view/repository/1.0">
  <cm:content
    xmlns:view="http://www.alfresco.org/view/repository/1.0"
    xmlns:sys="http://www.alfresco.org/model/system/1.0"
    xmlns:app="http://www.alfresco.org/model/application/1.0"
    xmlns:cm="http://www.alfresco.org/model/content/1.0"
    xmlns="" view:childName="cm:signoff-rework-email.ftl">
    <view:aspects>
      <cm:titled></cm:titled>
      <app:inlineeditable></app:inlineeditable>
    </view:aspects>
    <view:properties>
```

```
<sys:store-protocol>workspace</sys:store-protocol>
<sys:store-identifier>SpacesStore</sys:store-identifier>
<sys:node-uuid>signoff_rework_email_template_1_0</sys:node-
  uuid>
<app:editInline>true</app:editInline>
<cm:description>Sign-Off Rejection Email Template when rework
  is required</cm:description>
<cm:content>
  contentUrl=classpath:alfresco/module/
    com_bestmoney_module_cms/templates/signoff-rework-
    email.ftl|mimetype=text/plain|size=668|encoding=UTF-8
</cm:content>
<cm:title>signoff-rework-email.ftl</cm:title>
<cm:name>signoff-rework-email.ftl</cm:name>
<cm:created>2010-08-11T15:00:00.000+01:00</cm:created>
</view:properties>
<view:associations></view:associations>
</cm:content>
</view:view>
```

This XML file just defines how the template should be imported but it actually does not import it. We need to add an Importer bean for that in the `bootstrap-context.xml` Spring configuration file (located in the `com_bestmoney_module_cms\context` directory):

```
<bean id="com.bestmoney.cms.bootstrapEmailTemplates"
class="org.alfresco.repo.module.ImporterModuleComponent"
  parent="module.baseComponent">
  <property name="moduleId" value="com_bestmoney_module_cms"/>
  <property name="name"
    value="com.bestmoney.cms.bootstrapEmailTemplates"/>
  <property name="description" value="Initial email template
    requirements"/>
  <property name="sinceVersion" value="1.0"/>
  <property name="appliesFromVersion" value="1.0"/>
  <property name="importer" ref="spacesBootstrap"/>
  <property name="bootstrapViews">
    <list>
      <props>
        <prop key="path">
          /${spaces.company_home.childname}
          /${spaces.dictionary.childname}
          /${spaces.templates.email.childname}
          /cm:Best_x0020_Money/cm:Marketing
        </prop>
      </props>
    </list>
  </property>
</bean>
```

```

        <prop key="location">
            alfresco/module/com_bestmoney_module_cms/bootstrap/
            signoff-rework-email-template.xml
        </prop>
    </props>
</list>
</property>
</bean>

```

For this import to work, the .../Best Money/Marketing folders have to exist.

Sign-off process—property files for UI labels

For the Sign-off process, UI label resources create a new property file called `labels-signoff-workflow.properties` and put it in the `com_bestmoney_module_cms/ui` directory. Then add the properties in the same way as we have done previously. We are not showing it here as it is straightforward, have a look in the source code.

Sign-off process—task property sheets

For the property sheets related to the Sign-off process tasks, we will use a separate web client configuration file called `web-client-config-signoff-workflow.xml`.

Start by defining the following property sheet for the S001 task:

```

<config evaluator="node-type" condition="bmw:S001_Lev3SignOffTask">
    <property-sheet>
        <separator name="sep-0" display-label-id="signOffPhaseHeader"
            component-generator="HeaderSeparatorGenerator"/>
        <show-property name="bmw:signOffPhase" display-label-id="signOffPhase"
            read-only="true"/>
        <separator name="sep-1" display-label-id="jobWorkflowHeader"
            component-generator="HeaderSeparatorGenerator"/>
        <show-property name="bmc:campaignId" display-label-id="campaignId"
            read-only="true"/>
        <show-property name="bmc:product" display-label-id="product"
            read-only="true"/>
        <show-property name="bmc:jobType" display-label-id="jobType"
            read-only="true"/>
        <show-property name="bmw:jobStatus" display-label-id="jobStatus"
            read-only="true"/>
        <separator name="sep-2" display-label-id="signoffCatAndAssigneesHeader"
            component-generator="HeaderSeparatorGenerator"/>
    </property-sheet>
</config>

```

```
<show-property name="bmw:briefSignOffCategory" display-label-id="briefSignOffCategory" read-only="true"/>
<show-property name="bmw:productionSignOffCategory" display-label-id="productionSignOffCategory" read-only="true"/>
<show-association name="bmw:assigneeApprover1Person" display-label-id="approver1Person" read-only="true"/>
<show-association name="bmw:assigneeApprover2Person" display-label-id="approver2Person" read-only="true"/>
<separator name="sep-3" display-label-id="workItemsHeader" component-generator="HeaderSeparatorGenerator"/>
<show-property name="bmw:workTypes" display-label-id="workTypes" read-only="true"/>
<show-property name="bmw:conceptWorkDueDate" display-label-id="conceptWorkDueDate" read-only="true"/>
<show-property name="bmw:designWorkDueDate" display-label-id="designWorkDueDate" read-only="true"/>
<show-property name="bmw:copywriteWorkDueDate" display-label-id="copywriteWorkDueDate" read-only="true"/>
</property-sheet>
</config>
```

The property sheets for the S002 and S003 tasks are pretty much the same except that we display approver 1 and 3 for S002 and approver 2 and 3 for S003.

The last property sheet looks like this for S004:

```
<config evaluator="node-type" condition="bmw:S004_CheckSignoffCommentsTask">
  <property-sheet>
    <separator name="sep-0" display-label-id="signOffPhaseHeader" component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmw:signOffPhase" display-label-id="signOffPhase" read-only="true"/>
    <separator name="sep-1" display-label-id="jobWorkflowHeader" component-generator="HeaderSeparatorGenerator"/>
    <show-property name="bmc:campaignId" display-label-id="campaignId" read-only="true"/>
    <show-property name="bmc:product" display-label-id="product" read-only="true"/>
    <show-property name="bmc:jobType" display-label-id="jobType" read-only="true"/>
    <show-property name="bmw:jobStatus" display-label-id="jobStatus" read-only="true"/>
  </property-sheet>
</config>
```

There are new labels that we use for these property sheets and they need to be added as follows to the `webclient-workflow.properties` file:

```
signOffPhaseHeader=Sign-off Phase
signOffPhase=Sign-off Phase
signoffCatAndAssigneesHeader=Sign-off categories and assignees
briefSignOffCategory=Brief Sign-off Category
productionSignOffCategory=Production Sign-off Category
approver1Person=Level 1 Approver (Director)
approver2Person=Level 2 Approver (Marketing Manager)
approver3Person=Level 3 Approver (Group Manager)
```

Sign-off—bootstrapping UI property files and property sheets configuration

The workflow bootstrapping beans now need to be updated with the two new files that we have created for the Sign-off process. This is done in the `bootstrap-context.xml` Spring configuration file located in the `com_bestmoney_module_cms\context` directory.

Update the following bean configuration to include the Sign-off process properties:

```
<bean id=
  "com.bestmoney.marketing.properties.workflowBootstrap"
  parent="workflowDeployer">
  <property name="labels">
    <list>
      . . .
      <value>
        alfresco.module.com_bestmoney_module_cms.ui.labels-signoff-
        workflow
      </value>
    </list>
  </property>
</bean>
```

Then bootstrap the Sign-off process task property sheets:

```
<bean id="com.bestmoney.webclient.configBootstrap"
  class="org.alfresco.web.config.WebClientConfigBootstrap"
  init-method="init">
  <property name="configs">
    <list>
      . . .
```

```
<value>
  classpath:alfresco/module/com_bestmoney_module_cms/ui/web-
  client-config-signoff-workflow.xml
</value>
</list>
</property>
</bean>
```

Sign-off process—testing it

So now we are ready to test this new Sign-off process. This process is also designed to be part of a bigger parent process so we need a way of testing it without depending on information being passed in from the parent process. We have done this for the Work and Studio processes so we are not going to show it again here. The source code contains a `processdefinition_test.xml` file that is located in the `com_bestmoney_module_cms\workflows\signoff` directory and it can be used to test the Sign-off process individually. It requires the `approver1`, `approver2`, and `approver3` users to be created for it to work. It has the `signOffLevel` set to `LEV3` so only one approver on the group level will be needed to approve when we test. If you want to test other levels then change this variable.

To deploy the Test version of the Sign-off process do as follows:

```
deploy alfresco/module/com_bestmoney_module_cms/workflows/signoff/
processdefinition_test.xml
```

If approver 2 or approver 1 rejects the job then an e-mail should be sent. For this to work we have to configure Alfresco with what SMTP server to use. This can be done in the `tomcat/shared/classes/alfresco-global.properties` file:

```
mail.host=smtp.bestmoney.com
mail.username=anonymous
mail.password=
mail.encoding=UTF-8
mail.from.default=alfresco@bestmoney.com
mail.smtp.auth=false
```

If you get an error when starting Alfresco it is probably because the `/Data Dictionary/Email Templates/Best Money/Marketing` folder structure does not exist. Comment out the bean called `com.bestmoney.cms.bootstrapEmailTemplates` in the `bootstrap-context.xml` file and restart Alfresco. Then create the required folders. Stop Alfresco and enable the bean again and you should be fine.

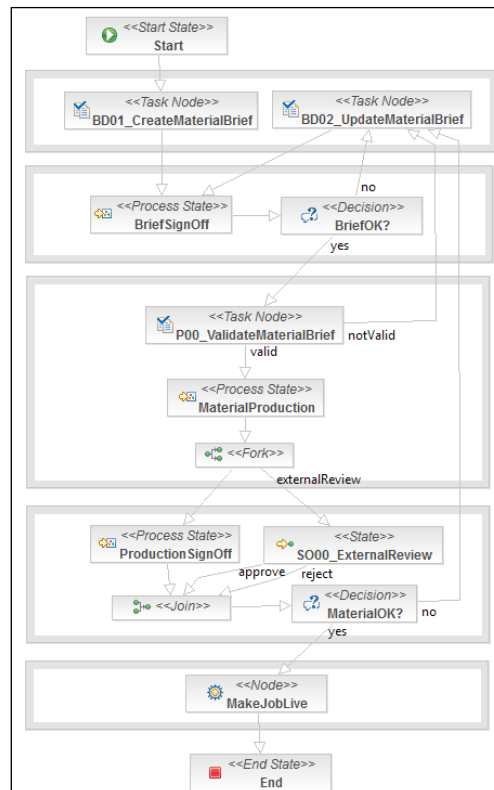
Implementing the Job process

Okay, we are finally at the parent Job process that is going to tie everything together. Start up Eclipse and add a new process definition for the Job process and call it **job**.

Job process—workflow definition (jPDL)

In the Job process Swimlane diagram, we use phases to divide the process into different stages or phases. The Job process definition will make use of a new feature in jPDL called 'Super States' to implement this. A super state is used to enclose one or more nodes and is usually used to depict a certain *phase* of the business process.

The Job process diagram looks like this:



When designing the workflow definition with phases there are a couple of things that are good to think about:

- **Always start with the super states:** You can put new nodes in a super state but not drag-and-drop existing nodes into a super state.
- **Name the super state as soon as you have created it:** This is useful as otherwise it can be tricky to remember to change the name in all places where it is used, and then you will be wondering why the process is not behaving the way you want.

In the previous diagram, we will name our phases/super states as follows, top to down:

- BriefDefinitionPhase
- BriefSignOffPhase
- ProductionPhase
- ProductionSignOffPhase
- LivePhase

The easiest way to name a super-state is to switch to the Source window in Eclipse and do it as soon as you have dragged-and-dropped it in the diagram window. We can also see the use of another node type called `state` that will be used as a waiting state for the external reviewer to decide if the produced material should be approved or not.

Now set the process definition name manually to `bmw:jobProcess`. The jPDL XML Schema version also needs to be updated to 3.2. The Job process is going to need an `initiator` Swimlane, which is always needed for a process that is not a subprocess. It will contain the person that started the workflow, in our case this will be the Job Owner. To define this Swimlane do this:

```
<swimlane name="initiator"/>
```

We then update the `start-state` node as follows:

```
<start-state name="Start">
  <task name="bmw:startJobProcessTask" swimlane="initiator"/>
  <transition to="BriefDefinitionPhase/BD01_CreateMaterialBrief"/>
  <event type="node-leave">
    <script>
      <variable name="studioTeamMgrsGroupName" access="write"/>
      <expression>
        studioTeamMgrsGroupName = "GROUP_STUDIO_TEAM_MANAGERS";
      </expression>
    </script>
  </event>
</start-state>
```

```

        </expression>
      </script>
    </event>
  </start-state>

```

The first thing we have done here is to add a task and assign it to the `initiator` Swimlane. This is the special start task needed to get the workflow going and we will only require the user to enter a comment when he or she starts the workflow.

 We could have defined the start node in a way so the start task required the user to enter all `bmc:job` and `bmw:job` data. This also works, only problem is if the user has filled in almost all data and then wants to save it and start the workflow later on, this is not possible with a start task. And that is why we do this in the `BD01` task instead, where the user can save the task if he or she wants to before completing it.

Then we add a `node-leave` event handler that sets up the name of the Studio Team Managers group. This name will be used throughout the processes and it is good to set it in one place only.

Notice how transitions are specified now when we have phases involved, the phase name needs to be included like a path to the node we are transitioning to.

Now let's start adding a few things to the `BriefDefinitionPhase`:

```

<super-state name="BriefDefinitionPhase">
  <event type="superstate-enter">
    <script>
      <variable name="bmw_jobStatus" access="write"/>
      <expression>
        bmw_jobStatus = "Brief Definition";
      </expression>
    </script>
  </event>

```

Here we have used a new event handler for the super-state called `superstate-enter`. We use it to set the status for the job. The event handlers on the phase level are good when we want to set variables that are relevant for all nodes in the phase.

Let's continue updating the `BD01_CreateMaterialBrief` task node:

```

<task-node name="BD01_CreateMaterialBrief">
  <task name="bmw:BD01_CreateMaterialBriefTask" swimlane="initiator">
    <event type="task-create">

```

```
<action
  class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
  <script>
    taskInstance.setVariable(
      "bmw_briefSignOffCategory", "LEV3");
    taskInstance.setVariable(
      "bmw_productionSignOffCategory", "LEV3");
  </script>
</action>
</event>
</task>
<transition to="../BriefSignOffPhase/BriefSignOff"/>
</task-node>
```

As usual, we are adding a task for the task node. The task has a handler for task-create events. We use the handler to set default values for two of the variables. So why do we not just set these default values in the Workflow content model? This is because when the property sheets for a subprocess, such as Sign-off, are displayed they will overwrite any property value with the default value – if any is defined in the content model.

Also add a task to the BD02_UpdateMaterialBrief node:

```
<task-node name="BD02_UpdateMaterialBrief">
  <task name="bmw:BD02_UpdateMaterialBriefTask"
    swimlane="initiator"/>
  <transition to="../BriefSignOffPhase/BriefSignOff"/>
</task-node>
```

Note how the "../" is used to navigate out of the current phase to be able to step into the phase we are going to next. We finish up the BriefDefinitionPhase by setting a couple of variables that control what type of work should be done:

```
<event type="superstate-leave">
  <script>
    <variable name="bmw_workTypes" access="read"/>
    <variable name="conceptWork" access="write"/>
    <variable name="designWork" access="write"/>
    <variable name="copyWork" access="write"/>
    <expression>
      conceptWork = false;
      designWork = false;
      copyWork = false;
      if (bmw_workTypes.contains("Concept")) {
        conceptWork = true;
      }
    </expression>
  </script>
</event>
```

```

        if (bmw_workTypes.contains("Design")) {
            designWork = true;
        }
        if (bmw_workTypes.contains("Copywrite")) {
            copyWork = true;
        }
    </expression>
</script>
</event>
</super-state>

```

For this we use a handler for the `superstate-leave` event. The work that should be done (that is concept, design, and/or copywrite) is specified in a multi-value variable called `bmw_workTypes` and here we check what specific work types it contains and set the `conceptWork`, `designWork`, and `copyWork` process variables accordingly.



If we for example, select Concept and Copywrite work then the `bmw_workTypes` variable value will look like `[Concept, Copywrite]`.

Next up is the `BriefSignOffPhase` where we should call the Sign-off subprocess to sign off the Material Brief. Start by setting the job status as follows:

```

<super-state name="BriefSignOffPhase">
  <event type="superstate-enter">
    <script>
      <variable name="bmw_jobStatus" access="write"/>
      <expression>
        bmw_jobStatus = "Brief Sign-Off";
      </expression>
    </script>
  </event>

```

Then update the `BriefSignOff` process-state so it calls the Sign-off process:

```

<process-state name="BriefSignOff">
  <sub-process name="bmw:signoffProcess"/>
  <event type="node-enter">
    <script>
      <variable name="bmw_signOffPhase" access="write"/>
      <expression>
        bmw_signOffPhase = "Material Brief";
      </expression>
    </script>
  </event>

```

In the `node-enter` event handler, we set up a variable `bmw_signOffPhase` with a name that distinguishes this particular sign-off process from the other sign-off processes further down in the Job Process. In this case, we call it `Material Brief`, which will be reflected in the task descriptions that the users will see, for example: `"SO03-Sign-off Phase (Material Brief)"`.

There are quite a few variables we need to pass into the sign-off process, but let's start with the variables that should be set by the subprocess and passed back to the Job process:

```
<variable name="briefSignedOff" access="write" mapped-
  name="signedOff"/>
<variable name="briefSignOffProcId" access="write" mapped-
  name="procId"/>
```

The first variable `briefSignedOff` will be set when the subprocess finishes and it will contain the outcome of the sign-off. The other `briefSignOffProcId` variable contains the process ID for the Sign-off process if we should need it for reporting.

Next we pass in all the variables from the `bmc:job` and `bmw:job` aspects:

```
<variable name="bmc_campaignId" access="read"/>
<variable name="bmc_product" access="read"/>
<variable name="bmc_jobType" access="read"/>
<variable name="bmw_jobStatus" access="read"/>
<variable name="bmw_workTypes" access="read"/>
<variable name="bmw_conceptWorkDueDate" access="read"/>
<variable name="bmw_designWorkDueDate" access="read"/>
<variable name="bmw_copywriteWorkDueDate" access="read"/>
<variable name="bmw_briefSignOffCategory" access="read"/>
<variable name="bmw_productionSignOffCategory" access="read"/>
<variable name="bmw_assigneeApprover1Person" access="read"/>
<variable name="bmw_assigneeApprover2Person" access="read"/>
<variable name="bmw_assigneeApprover3Person" access="read"/>
<variable name="bmw_externalReviewerEmail" access="read"/>
<variable name="bmw_signOffPhase" access="read"/>
<variable name="bmw_briefSignOffCategory" access="read" mapped-
  name="signOffCategory"/>
```

The last `bmw_briefSignOffCategory` variable that we pass in tells the Sign-off process what level of approval we need (default is LEV3). We also need to pass in a few variables that might not be obvious at first:

```
<variable name="bpm_package" access="read"/>
<variable name="companyhome" access="read"/>
<variable name="initiator" access="read" mapped-name="jobOwner"/>
<transition to="BriefOK?"/>
</process-state>
```

As we mentioned before, when we start specifying what variables should be passed into the sub-process we have to specify all that can possibly be needed. The last thing we need to do in the `BriefSignOffPhase` is to check in the `BriefOk?` node whether the material brief was signed-off or not:

```
<decision name="BriefOK?">
  <transition to="../BriefDefinitionPhase/BD02_UpdateMaterialBrief"
    name="no"/>
  <transition to="../ProductionPhase/P00_ValidateMaterialBrief"
    name="yes">
    <condition>#{briefSignedOff == true}</condition>
  </transition>
</decision>
</super-state>
```

If the brief was signed off then we enter into the `ProductionPhase` where we start by setting the job status and adding a task to the `P00` task node:

```
<super-state name="ProductionPhase">
  <event type="superstate-enter">
    <script>
      <variable name="bmw_jobStatus" access="write"/>
      <expression>
        bmw_jobStatus = "Production";
      </expression>
    </script>
  </event>
  <task-node name="P00_ValidateMaterialBrief">
    <task name="bmw:P00_ValidateMaterialBriefTask">
      <assignment
        class="org.alfresco.repo.workflow.jbpm.AlfrescoAssignment">
        <pooledactors>#{people.getGroup(studioTeamMgrsGroupName)}
        </pooledactors>
      </assignment>
    </task>
    <transition to="../BriefDefinitionPhase/BD02_UpdateMaterialBrief"
      name="notValid"/>
    <transition to="MaterialProduction" name="valid"/>
  </task-node>
```

We do not always have to use Swimlanes when assigning tasks to users. Here we can see how the special `AlfrescoAssignment` class can be used to assign tasks to users or groups of users. In this case, we use the Studio Team Managers group name and assign this task to all members of this group.

The material specified in the brief is produced in the Studio process, so configure it as follows:

```
<process-state name="MaterialProduction">
  <sub-process name="bmw:studioProcess"/>

  <variable name="studioProcId" access="write" mapped-name="procId"/>
  <variable name="studioConceptWorkProcId" access="write"/>
  <variable name="studioCopyWorkProcId" access="write"/>
  <variable name="studioDesignWorkProcId" access="write"/>
```

The first couple of variables we define are the outbound variables that will store the process IDs for all the subprocesses. Some of these variables might be empty, for example, if the Job Owner has not selected any Copywrite work to be done. Then the studioCopyWorkProcId variable would be null.

Then pass in the required variables from bmc:job and bmw:job aspects:

```
<variable name="bmc_campaignId" access="read"/>
<variable name="bmc_product" access="read"/>
<variable name="bmc_jobType" access="read"/>
<variable name="bmw_jobStatus" access="read"/>
<variable name="bmw_workTypes" access="read"/>
<variable name="bmw_conceptWorkDueDate" access="read"/>
<variable name="bmw_designWorkDueDate" access="read"/>
<variable name="bmw_copywriteWorkDueDate" access="read"/>
```

We also pass in the Studio Team Managers Group name as there are tasks in the Studio and Work processes that should be assigned to the members of this group. Also pass in the work type variables that we set up in the BriefDefinitionPhase so the studio knows what to do:

```
<variable name="studioTeamMgrsGroupName" access="read"/>
<variable name="conceptWork" access="read"/>
<variable name="designWork" access="read"/>
<variable name="copyWork" access="read"/>
```

Then pass in the content package and the initiator person:

```
<variable name="bpm_package" access="read"/>
<variable name="initiator" access="read" mapped-name="jobOwner"/>
```

The bpm_package contains the document from which the workflow was started. Users can also decide to add more documents to this package as the workflow progresses. But nobody is forced to add any documents to this package during the process. In fact all the material that is produced by the studio might just be stored in the folder from where the workflow was started (that is, where the document is from when the workflow was started).

The following properties are also defined:

```
<variable name="bmw_briefSignOffCategory" access="read"/>
<variable name="bmw_productionSignOffCategory" access="read"/>
<variable name="bmw_assigneeApprover3Person" access="read"/>
<variable name="bmw_externalReviewerEmail" access="read"/>
<transition to="fork1" name=""/>
</process-state>
```

None of these properties are actually used or displayed by the Studio process or the Work process. However, they need to be passed into the Studio process anyway as they are mandatory in the `bmw:job` aspect definition. If not passed in, it would not be possible to complete a task that is based on a type that has this aspect.

When the material is produced and approved as okay out of the Studio, we end up in the `ProductionSignOffPhase` phase where there is another sign-off of the new material. The call to the Sign-off subprocess is pretty much the same as when the material brief was signed off so we are not repeating it here (see the source code). However, there is an extra sign off (that is, `S000_ExternalReview`) by an external person going on in parallel. So let's have a look at that and see how it is defined and implemented.

The way the external sign-off works is that the Job Owner specifies an e-mail address for an external reviewer when the brief is created. You might have noticed that there is a new property called `bmw_externalReviewerEmail` that we need to add to the `bmw:job` aspect.

When we get to the `S000_ExternalReview` node we call a custom action from a script in a node-enter event handler. This custom action sends an e-mail based on a template to the e-mail address specified with the `bmw_externalReviewerEmail` property. Here is how it looks:

```
<state name="S000_ExternalReview">
  <event type="node-enter">
    <script>
      <variable name="externallySignedOff" access="write"/>
      <variable name="notificationRecipient"
        access="read,write"/>
      <variable name="bmw_externalReviewerEmail"
        access="read"/>
      <expression>
        externallySignedOff = false;
        notificationRecipient = bmw_externalReviewerEmail;
      </expression>
    </script>
```



```
<action class="com.bestmoney.cms.workflow.actionhandler
.ExternalSignOffNotification"/>
</event>
<transition to="join1" name="reject"/>
<transition to="join1" name="approve">
  <script>
    <variable name="externallySignedOff"
      access="read,write"/>
    <expression>
      externallySignedOff = true;
    </expression>
  </script>
</transition>
</state>
```

The ExternalSignOffNotification action is implemented as a JBPMSpringActionHandler and Alfresco Mail action to send the e-mail. See the source code delivered with this chapter for more info on how to write these kinds of jBPM actions. The action constructs an e-mail looking something like this:

You have been assigned to a task named SO00_ExternalReview. To sign-off click on the link below.

Documents that are waiting sign-off:

Alfresco Training Guide.pdf

Reject:[http://<hostname>:8080/alfresco/service/bestmoney/marketing/bpm/signoff?id=jbpm\\$34-@externalReview2&action=reject&guest=true](http://<hostname>:8080/alfresco/service/bestmoney/marketing/bpm/signoff?id=jbpm$34-@externalReview2&action=reject&guest=true)

Approve:[http://<hostname>:8080/alfresco/service/bestmoney/marketing/bpm/signoff?id=jbpm\\$34-@externalReview&2action=approve&guest=true](http://<hostname>:8080/alfresco/service/bestmoney/marketing/bpm/signoff?id=jbpm$34-@externalReview&2action=approve&guest=true)

There are two links, one for each transition out of the SO00_ExternalReview state node. So when the external reviewer clicks one of these links it moves the process forward by signaling one of the transitions. The way the process is signaled is handled by a Web Script that accompanies the custom action. It is registered as follows:

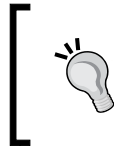
```
<webscript>
  <shortname>External Sign-off</shortname>
  <description>External sign-off via email of produced
    material</description>
  <url>/bestmoney/marketing/bpm/signoff?id={argument}&action={
    transArgument}</url>
  <format default="html">extension</format>
```

```

    <authentication>guest</authentication>
    <transaction>none</transaction>
  </webscript>

```

And it kicks off a class called `ExternalSignOffWebScript`, see source code, that is a `DeclarativeWebScript`. This java-based web script takes the *id* parameter from the URL and the *action* parameter from the URL and calls the `workflowService.signal(id, action)` method.



Note that in a production deployment we would want to change the Web Script authentication from `guest` to `user` so not everyone can approve or reject by just getting their hands on the URLs.

The `S000_ExternalReview` node keeps track of the external review outcome with the `externallySignedOff` variable. Then we just have to also check the material production sign-off outcome to know if the material should go live:

```

<decision name="MaterialOK?">
  <transition to="../BriefDefinitionPhase/BD02_UpdateMaterialBrief"
    name="no"></transition>
  <transition to="../LivePhase/MakeJobLive" name="yes">
    <condition>#{productionSignedOff == true and
      externallySignedOff == true}</condition>
  </transition>
</decision>
</super-state>

```

We now enter into the last `LivePhase` where we classify the produced material, if it is stored in the folder from where the Job workflow was started. The `MakeJobLive` node looks as follows:

```

<super-state name="LivePhase">
  . . .
  <node name="MakeJobLive">
    <event type="node-enter">
      <action
        class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
        <runas>admin</runas>
        <script>
          var props = new Array();
          props["bmc:campaignId"] = bmc_campaignId;
          props["bmc:product"] = bmc_product;
          props["bmc:jobType"] = bmc_jobType;

```

```

        var jobfolder = bpm_package.children[0].parent;

        for (var i = 0; i < jobfolder.children.length; i++) {
            var document = jobfolder.children[i];
            if (document.isDocument) {
                document.addAspect("bmc:job", props);
                document.save();
            }
        }
    </script>
</action>
</event>
<transition to=" ../End"/>
</node>
</super-state>

```

Here, we apply the `bmc:job` aspect to all the produced material. Note that we do not loop through the items in the `bpm_package` as it might just contain the initial document that was used to start the process. Instead, we use the `bpm_package` variable to get to the folder where the new material hopefully has been uploaded.



When setting variables in an Array be careful with date variables and check that they are defined (that is, not null) as follows:

```
if (typeof(bmc_someDate) !== 'undefined') props["bmc:someDate"] = bmc_someDate;
```

And any multi-value variable needs to be handled as follows, let's say that we could specify multiple products for example:

```
bmc_products = bmc_products.replace('[', ' ').
replace(']', '');
```

```
props["bmc:products"] = bmc_products.split(',');
```

If we take a look at one of the documents produced via this job process, we will see that it has been classified with the `bmc:job` aspect. Here is an example of what we should see in the **View Details** page:

Marketing Production Material	
Campaign Id:	TestCampaign7
Associated Product:	Credit Card
Job Type:	Web Page



When using JavaScript it is important to think about how we use comments. If we comment with `"/ /"`, we will effectively comment out all the code after that as Alfresco JavaScript is formatting code on a single line only.

The action scripts are stored in the `JBPM_DELEGATION` table in the `CONFIGURATION_` column. And the size is just 4,000 bytes when using the Derby database. When using MySQL, it is a `TEXT` type so no problem with size there. Any comments inside the script will be stored too, even if you do an XML comment (`<!--`) when inside the action tag.

Job process—workflow content model

As usual, we need to create Types for each task in the workflow definition, so the data used by the tasks has to be stored somewhere. Add the following types to the `workflow-model.xml` file:

```
<type name="bmw:startJobProcessTask">
  <parent>bpm:startTask</parent>
</type>
<type name="bmw:BD01_CreateMaterialBriefTask">
  <parent>bmw:baseJobTask</parent>
</type>
<type name="bmw:BD02_UpdateMaterialBriefTask">
  <parent>bmw:baseJobTask</parent>
  <overrides>
    <property name="bpm:packageItemActionGroup">
      <default>edit_package_item_actions</default>
    </property>
  </overrides>
</type>
<type name="bmw:P00_ValidateMaterialBriefTask">
  <parent>bmw:baseJobTask</parent>
</type>
```

If a task is rejected and needs updating we allow the `bpm:package` content to be updated. This is done by setting the `bpm:packageItemActionGroup` as shown previously.

Job process—property file, property sheets, and bootstrapping

We have done this a few times now, so we are not going to repeat it. Have a look at the source code and specifically the `labels-job-workflow.properties` and `web-client-config-job-workflow.xml` files. For information about how these files are bootstrapped see the `bootstrap-context.xml` Spring configuration file.

The configuration produces the following UI for creating a new Job:

Task Properties

Owner:

jobOwner

General information about this job

Ⓢ Campaign Id:

Test Campaign

Ⓢ Associated Product:

Credit Card

Ⓢ Job Type:

Advert

Job Status:

Brief Definition

Sign-off categories and assignees

Brief Sign-off Category:

LEV3

Production Sign-off Category:

LEV3

Level 1 Approver (Director):

Select...

Level 2 Approver (Marketing Manager):

Select...

Ⓢ Level 3 Approver (Group Manager):

Approver Number 3 [approver3]  

Ⓢ External Reviewer Email:

martin.bergjung@opsera.com

Work Items

Concept

Add to List

Ⓢ Work Types:

Name

Concept 

Design 

Concept Work Due Date:

27

September

2010

10

:

19

Today

None

Design Work Due Date:

29

October

2010

10

:

19

Today

None

Copywrite Work Due Date:

None

Job process—testing it

We have now finally completed the whole Job process, including all subprocesses. We have tested the subprocesses, so now we just have to try out the whole job process and see if everything works together.

The first thing we should do is log in as administrator and redeploy all workflows, and do this in the correct order. The following list is in the order in which the workflows need to be deployed:

```
deploy alfresco/module/com_bestmoney_module_cms/workflows/work/
processdefinition.xml
deploy alfresco/module/com_bestmoney_module_cms/workflows/studio/
processdefinition.xml
deploy alfresco/module/com_bestmoney_module_cms/workflows/signoff/
processdefinition.xml
deploy alfresco/module/com_bestmoney_module_cms/workflows/job/
processdefinition.xml
```

The way we should always deploy workflows is in the other way around from how they are used. For example, the Job process uses the Studio process, which in turn uses the Work process. So we need to deploy them in the order Work, Studio, and Job. The Job process also uses the Sign-off process, which then needs to be deployed before the Job process. The order in which the Studio and Sign-off process definitions are deployed does not matter.

Extending the workflow solution

Now people can start using the Best Money Job Workflow to control and manage the production of marketing material. However, it will not be long before we will get questions and requests like:

- How about e-mail notifications, can the system send me an e-mail when I am assigned a new task?
- Is it possible to have some custom fields in the My Tasks To Do, My Pooled tasks, and so on?
- We need a management Dashlet where we can see all assigned tasks for all ongoing workflows, is it available?
- Is it possible to have a list of all tasks completed for a Job workflow, for auditing reasons?

And the list goes on. This section will go through a couple of solutions for these kinds of requests. The scope of this chapter does not allow us to go through all the source code for the solutions, just how they are used in the workflow. Download the source code for this chapter for more detailed information about the implementation.

Adding e-mail notification

One really useful feature that we are going to need is the possibility to have e-mail notifications sent out automatically when a user or users of a group has been assigned a task. This can be done easily with a jBPM action handler called `TaskAssignmentNotificationActionHandler`. It is used as follows in the process definition to send an e-mail when a user has been assigned a task:

```
<task name="bmw:BD01_CreateMaterialBriefTask" swimlane="initiator">
  <event type="node-enter">
    <action class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
      <script>
        executionContext.setVariable(
          "taskNotificationRecipientGroup", "");
        executionContext.setVariable(
          "taskNotificationRecipientEmail",
            initiator.properties.email);
      </script>
    </action>
    <action class="com.bestmoney.cms.workflow.actionhandler.
TaskAssignmentNotificationActionHandler"/>
  </event>
```

There are two variables that are set in the execution context and used by the action handler. The `taskNotificationRecipientGroup` variable should be used when the e-mail should be sent to members of a group. And the `taskNotificationRecipientEmail` should be used when the e-mail should be sent to one individual, usually the Swimlane person.

The `BD01_CreateMaterialBriefTask` has this configured, so you can see how it works.

Remember to redeploy the workflow definition after a change like this and make sure that the user has a valid e-mail address specified.

Using customized task dashlets

The My Task To Do and My Pooled Tasks Dashlets do not really display any information about what Campaign the job is associated with or what Job type it is. It would be good to add this to the task listing. The source code comes with three new dashlets that adds the Campaign ID and Job Type columns:

Best Money Tasks-to-do							
Campaign Id	Job Type	Description	Created	Due Date	Status	Priority	Actions
		BD01-Create a Material Brief	26 August 2010 10:18		Not Yet Started	3	 
TestCamp1	Advert	BD02-Update a Material Brief	24 August 2010 17:28		Not Yet Started	3	 
Campaign 1	Poster	W03-Validate Completed Work (Concept)	20 August 2010 13:35		Not Yet Started	3	 
Page 1 of 1   1  							
Best Money Pooled Tasks-to-do							
No tasks found.							
Best Money Tasks completed							
Campaign Id	Job Type	Description	Created	Completed on	Outcome		Actions
		Start Marking Job Process	26 August 2010 10:18	26 August 2010			 
TestCampaign6	PR	SO04-Check Sign-off (Produced Material)	25 August 2010 15:26	25 August 2010			 
TestCampaign6	PR	W03-Validate Completed Work (Design)	25 August 2010 15:25	25 August 2010	Approve		 
TestCampaign6	PR	W03-Validate Completed Work (Concept)	25 August 2010 15:18	25 August 2010	Approve		 
TestCampaign6	PR	SO04-Check Sign-off (Material Brief)	25 August 2010 15:17	25 August 2010			 
TestCampaign6	PR	BD02-Update a Material Brief	25 August 2010 15:15	25 August 2010			 
TestCampaign6	PR	SO04-Check Sign-off (Produced Material)	25 August 2010 15:04	25 August 2010			 
TestCampaign6	PR	W03-Validate Completed Work (Design)	25 August 2010 15:02	25 August 2010	Approve		 
TestCampaign6	PR	W03-Validate Completed Work (Concept)	25 August 2010 15:01	25 August 2010	Approve		 
TestCampaign6	PR	SO04-Check Sign-off (Material Brief)	25 August 2010 15:00	25 August 2010			 
Page 1 of 4   1 2 3 4  							

These dashlets are available after installing the Best Money AMP. If you want to inspect the Dashlet JSP files, they are available in the `alf_extensions\trunk\_alfresco\web\jsp\bestmoney\dashlet` directory. Basically, the standard task dashlet JSPs are just copied to new filenames and then two columns are added to each one of them. They are added to the Web Client via configuration in the `web-client-config-custom.xml` file.

Management dashlets

When the workflow project goes live, it will not take long until there are lots of Job processes running simultaneously, making it harder and harder for managers to get an overview of what is going on.

All assigned tasks for all jobs dashlet

One very useful dashlet is one that displays what tasks have been assigned for each workflow instance (that is job), and to whom they have been assigned, and so on.

So sooner or later we will get requests for a dashlet that can show all assigned tasks for all ongoing workflows. This dashlet will be useful in the following situations:

- When we want to answer questions like "who is currently assigned to do work on this job"?
- Could you reassign this task to user X, the user it is assigned to, is on vacation, is sick, has left the company, and so on.

The source code comes with a dashlet that supports this request. It looks like this:



Campaign Id	Job Type	Description	Created	Due Date	Owner	Actions
		BD01-Create a Material Brief	26 August 2010 10:18		jobOwner	
TestCampaign5	Poster	P01-Set Concept Priority and Assign Worker(s)	25 August 2010 14:50	28 August 2010		
TestCampaign4	Booklet	SO03-Sign-off Phase (Material Brief)	25 August 2010 14:27		approver3	
TestCampaign3	Advert	P04-Approve Studio Work	25 August 2010 14:03			
TestCampaign2	Advert	P00-Validate a Material Brief	24 August 2010 18:37			
TestCamp1	Advert	BD02-Update a Material Brief	24 August 2010 17:28		jobOwner	
Campaign 1	Poster	SO03-Sign-off Phase (Brief)	22 August 2010 18:11		approver3	
Campaign 1	Poster	SO03-Sign-off Phase (Brief)	22 August 2010 17:55		approver3	
Campaign 1	Poster	W04-Amend Completed Work (Concept)	22 August 2010 11:43	30 August 2010	worker	
Campaign 1	Poster	W02-Validate Completed Work (Concept)	22 August 2010 11:20	25 August 2010		
Campaign 1	Poster	W01-Produce Work (Concept)	22 August 2010 11:03	28 August 2010	worker	
Campaign 1	Poster	W01-Produce Work (Concept)	22 August 2010 10:35	18 August 2010	worker	
Campaign 1	Poster	W04-Amend Completed Work (Concept)	20 August 2010 14:55		worker	
Campaign 1	Poster	W03-Validate Completed Work (Concept)	20 August 2010 13:35		jobOwner	
Campaign 1	Poster	W01-Produce Work (Concept)	15 August 2010 19:20		worker	

Page 1 of 1

There are a couple of groups that this dashlet relies on when it displays its UI. There is one group called **Marketing-All Users** that needs to be created and contains the user(s) that should have access to this Dashlet. A user that adds this Dashlet to his or her Dashboard but is not a member of this group will not be able to access this Dashlet, an error message is displayed.

There are also functionalities in the dashlet such as view Task dialog and Reassign task that require the user to be a member of the **Marketing-Admins** group. If the user is not a member of this group, the dashlet will display the UI with limited functionality.

There is a backing bean for this dashlet JSP that is called `CustomWorkflowBean` and it uses SQL queries to get the task information. It is also possible to use the `WorkflowService` and `query` for information, as in the following example:

```
List<Node> allTasks = new ArrayList<Node>();

WorkflowTaskQuery query = new WorkflowTaskQuery();
query.setTaskState(WorkflowTaskState.IN_PROGRESS);
query.setProcessName(processDefinition);
// Do not use order by as then the sorting by clicking on
// column header does not work
//      query.setOrderBy(new WorkflowTaskQuery.OrderBy[] {
//          WorkflowTaskQuery.OrderBy.TaskCreated_Desc,
//          WorkflowTaskQuery.OrderBy.TaskActor_Asc});

// Do the query
List<WorkflowTask> tasks = getWorkflowService().queryTasks(query);

// Loop thru each workflow task and create task object to
// hold task data
for (WorkflowTask task : tasks) {
    Node node = createTask(task);
    allTasks.add(node);
}
```

This works but it will be very difficult to get acceptable performance. For example, you would have to make the calls mentioned above for each process definition and it would not be possible to do any Joins like you can with SQL.

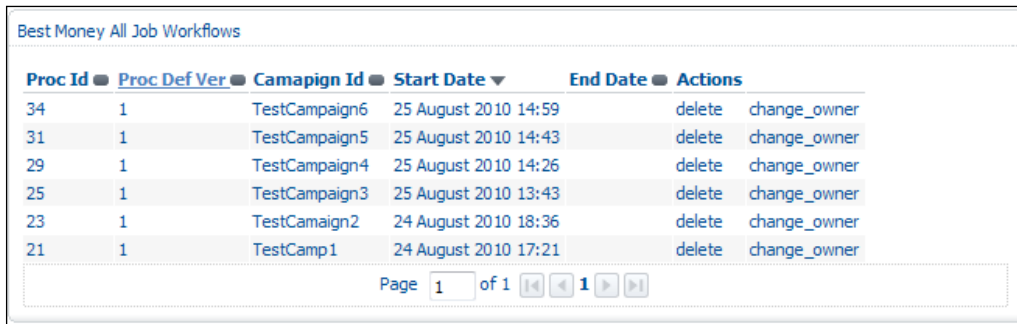
The query response time will pretty soon be unmanageable with requests taking as much as minutes. If this dashlet is used a lot it will not be acceptable.

All job workflows dashlet

Another request that will come up after a while is for a dashlet that can show all ongoing Job workflows, and have features for deleting a workflow, and changing the owner of a workflow. Out of the box you cannot delete a workflow so the tasks disappear from the Completed Tasks Dashlet, which means it will fill up pretty quickly and make the system unmanageable. Therefore, it is important to have a way of deleting a workflow instance that has been completed and is no longer needed.

It is also very useful to be able to change the owner of the workflow, basically change the initiator of the workflow. Sometimes people go on vacation, get sick, leave the company and so on, and in those cases we need a way to continue with the process anyway. This is not the same thing as re-assigning a single task. When we change the workflow owner, every task afterwards will be assigned to the new owner if it is associated with the `initiator` Swimlane.

Here is how this Dashlet looks:



Best Money All Job Workflows

Proc Id	Proc Def Ver	Camapign Id	Start Date	End Date	Actions
34	1	TestCampaign6	25 August 2010 14:59		delete change_owner
31	1	TestCampaign5	25 August 2010 14:43		delete change_owner
29	1	TestCampaign4	25 August 2010 14:26		delete change_owner
25	1	TestCampaign3	25 August 2010 13:43		delete change_owner
23	1	TestCamaign2	24 August 2010 18:36		delete change_owner
21	1	TestCamp1	24 August 2010 17:21		delete change_owner

Page 1 of 1

If we click on the delete action link, the following screen is displayed:



Best Money All Job Workflows

*** Delete Workflow ***

Proc Id: 31

Camapign Id: TestCampaign5

Proc Def Ver: 1

<= Go Back To List Delete

Click on the **Delete** button to delete this Job workflow permanently. This will automatically delete all involved subprocesses (that is, sign-off, studio, and work) by looking at the saved `procIds` for them.

If we click on the **Change Owner** link, the following screen will be displayed:

Best Money All Job Workflows

*** Change Workflow Owner (Initiator) ***

Proc Id: 34

Camapign Id: TestCampaign6

Proc Def Ver: 1

Current Owner (initiator): jobOwner

New Owner (initiator):

[<= Go Back To List](#) [Change Owner](#)

Here, we can specify the username for the new job owner and then click the **Change Owner** button. This changes the owner of the Job workflow and all its subprocesses.

Exporting the task summary list in an Excel spreadsheet

As Best Money is a financial institution, they want to have an audit trail of exactly what tasks were completed to produce marketing material. It should be available as an Excel spreadsheet and created in the folder where the material is at the end of the process.

We can use a jBPM action called `CreateTaskListForJobActionHandler` and add it to the `MakeJobLive` node in the beginning of the `node-enter` event:

```
<event type="node-enter">
  <action
    class="com.bestmoney.cms.workflow.actionhandler
      .CreateTaskListForJobActionHandler"/>
```

After running through a job process we should find an Excel spreadsheet in the materials folder, as in the following screenshot:

 TestCampaign7-Web Page.xls 	7 KB	27 August 2010 09:15	27 August 2010 09:15

The task list will look something like this:

TestCampaign7-Web Page.xls [Read-Only] [Compatibility Mode]							
	A	B	C	D	E	F	G
1	Phase	Task	Decision	Owner	Created	Due	Priority
2		Start Marketing Job Process		jobOwner	27/08/2010		2
3	Brief Definition	BD01-Create a Material Brief		jobOwner	27/08/2010		3
4	Brief Sign-Off	SO03-Sign-off Phase (Material Brief)	approve	approver3	27/08/2010		3
5	Brief Sign-Off	SO04-Check Sign-off (Material Brief)		jobOwner	27/08/2010		3
6	Production	P00-Validate a Material Brief	valid	studiomgr1	27/08/2010		3
7	Production	P01-Set Concept Priority and Assign Worker(s)		studiomgr1	27/08/2010	28/08/2010	3
8	Production	W01-Produce Work (Concept)		worker	27/08/2010	28/08/2010	3
9	Production	W02-Validate Completed Work (Concept)	approve	studiomgr1	27/08/2010	28/08/2010	3
10	Production	W03-Validate Completed Work (Concept)	approve	jobOwner	27/08/2010	28/08/2010	3
11	Production	P02-Set Copy Priority and Assign Copywriter		studiomgr1	27/08/2010	30/08/2010	3
12	Production	P03-Set Design Priority and Assign Designer		studiomgr1	27/08/2010	29/08/2010	3
13	Production	W01-Produce Work (Design)		worker	27/08/2010	29/08/2010	3
14	Production	W01-Produce Work (Copy)		worker2	27/08/2010	30/08/2010	3
15	Production	W02-Validate Completed Work (Copy)	approve	studiomgr1	27/08/2010	30/08/2010	3
16	Production	W02-Validate Completed Work (Design)	approve	studiomgr1	27/08/2010	29/08/2010	3
17	Production	W03-Validate Completed Work (Design)	approve	jobOwner	27/08/2010	29/08/2010	3
18	Production	W03-Validate Completed Work (Copy)	approve	jobOwner	27/08/2010	30/08/2010	3
19	Production	P04-Approve Studio Work		studiomgr1	27/08/2010		3
20	Production Sign-Off	SO03-Sign-off Phase (Produced Material)	approve	approver3	27/08/2010		3
21	Production Sign-Off	SO04-Check Sign-off (Produced Material)		jobOwner	27/08/2010		3

Note that this report contains all the tasks from all involved subprocesses.

Remember to redeploy the workflow definition after a change like this.

Material folder link

For the Studio team users to be able to easily locate the folder where they should upload the produced material, a folder link in the property sheets would be helpful. We can easily add a folder link to all the property sheets as follows. First start off by adding a new property to the base job type:

```
<type name="bmw:baseJobTask">
  <parent>bpm:workflowTask</parent>
  <properties>
    <property name="bmw:jobFolderLink">
      <title>A link/URL to the job folder</title>
      <type>d:text</type>
    </property>
  </properties>
</overrides>
```

This job folder property is set up when we enter the BriefSignOffPhase as follows:

```
<super-state name="BriefSignOffPhase">
  <event type="superstate-enter">
    <script>
      <variable name="bmw_jobStatus" access="write"/>
      <expression>
```

```

        bpm_jobStatus = "Brief Sign-Off";
    </expression>
</script>
<action
    class="org.alfresco.repo.workflow.jbpm.AlfrescoJavaScript">
    <script>
        <variable name="bpm_jobFolderLink" access="write"/>
        <expression>
            var jobfolder = bpm_package.children[0].parent.parent;
            if (jobfolder != null) {
                bpm_jobFolderLink = "/alfresco" +
                    jobfolder.getUrl();
            } else {
                logger.log("jobfolder = null");
            }
        </expression>
    </script>
</action>
</event>

```

Then add this property to all the property sheets as needed:

```

<show-property name="bpm:jobFolderLink" display-label-id="jobFolder"
    read-only="true" component-generator="CustomLinkGenerator"/>

```

This property uses a new component generator called `CustomLinkGenerator`, see chapter source code for how it has been implemented.

When a property sheet is displayed it will now contain the folder link such as in this example:

Job Folder (materials etc):	/alfresco/n/browse/workspace/SpacesStore/26cef395-d9c5-40f7-b1f4-6e4954462adb

Remember to redeploy the workflow definition after a change like this.

Summary

In the previous chapter, we saw how to implement a basic workflow with jBoss jBPM. In this chapter, we have taken this further to see the full potential of Alfresco's embedded workflow engine. Implementing parallel flows was explored and we saw how super-states could be used to implement phases. Super-states were good because we could just move a node inside a super state and it would have access to the variables defined inside the super state.

We also looked at how to use subprocesses to reuse common process definitions, using subprocesses also makes it easier to overview a business process. However, there are also things to think about when using subprocesses, such as how to be able to generate a task list for a parent process and all involved subprocesses. For this we used a special process identifier that was propagated up to the parent process so it could keep track of all subprocesses that were part of the execution.

The jBPM workflow engine is very well-integrated with Alfresco and we have seen that we can directly assign users and groups from Alfresco when creating the process definition. The Alfresco JavaScript language is also available to use in the process definition, which gives access to a lot of the Alfresco JavaScript root objects such as `companyhome` and `bpm_package` (that is, all documents managed by the workflow).

E-mail notifications do not come out of the box but we saw how that can be implemented with some custom Java coding and using Actions to call the code from the process definition. E-mails can also be sent directly from a process definition by using JavaScript.

Alfresco does not come with a lot of features for managing workflows when everything is deployed into production. We saw how custom Dashlets can be built to manage all active tasks, manage active workflows, and much more.

In the next chapter, we will look at enterprise application integration and see how we can integrate Alfresco content with other enterprise content. We will see how to use Portlets to display Alfresco content and how to use CMIS to extract the content from Alfresco.

12

Enterprise Application Integration (EAI) Solutions

These days people are often used to applications that aggregate content from many sources of information in the enterprise. Sometimes, these applications are referred to as *mashups*. Typically, there is one central portal deployed where users can see content from many different enterprise applications, if they have the permission to do so. So, it is becoming more and more important to have an idea of how to extract content from Alfresco and implement **Enterprise Application Integration (EAI)** solutions.

Best Money has deployed the Liferay portal as its main information centre, where content from many different financial applications is aggregated. Now, they are interested in creating a couple of new portlets to show information from Alfresco. They would like to start with a portlet, which shows recently created or added documents during the day.

In this chapter, we will look at:

- Building a web script that returns documents and folders modified during the current day
- Building a Java/HTML portlet that fetches recently modified content from a remote Alfresco repository
- Building a Java/GWT/GXT portlet that fetches recently modified content from a remote Alfresco repository
- Deploying these portlets in the Liferay portal
- Managing authentication so that users do not have to log in to both the portal and to Alfresco

Introducing portlets

The main feature of portals is to aggregate content from different sources. Each piece of content is displayed in a small window called a **portlet**. Each portlet is usually independent of other portlets and can run an application. We can say that portlets are small web applications that run within a so-called portlet container. We can compare it to servlets running inside a servlet container.

Portlet standards

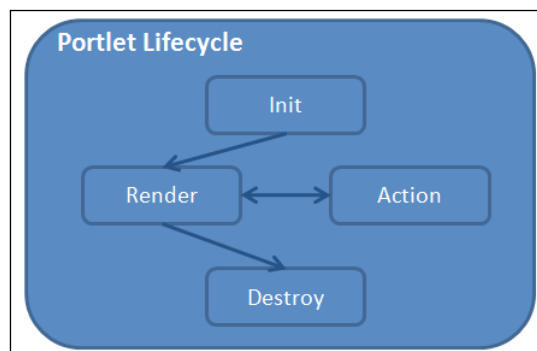
Portlets are standardized by the **Java Community Process (JCP)**, as opposed to portals, which are not. There are a couple of different portlet standards:

- **JSR168**: The basic standard for portlets (also known as Portlet version 1.0).
- **JSR286**: Adds support for inter-portlet communication via events. So, if you have a portlet showing the Alfresco folder structure, you could have that portlet sending out events when someone selects a new folder. Then another portlet displaying documents in a folder could change the view automatically when receiving events (also known as Portlet version 2.0).
- **JSR301** and **JSR329**: Supports displaying JSF content in the portlet.

The Liferay portal server implements both the JSR-168 specification and the JSR-286 specification. We will be focusing on basic portlet implementations in this chapter and will therefore not cover inter-portlet communication.

Portlet lifecycle

Portlets are living in the portlet container where they go through a clearly defined lifecycle. It is quite similar to the servlet lifecycle but it has some special features:



Before portlets are made available they run through the initialization phase and the container calls the `init` method in the portlet implementation (implements the `javax.portlet.Portlet` interface). The `init` method is called only once. When a portlet should be rendered and displayed, the `render` method is called by the container. This can happen for many reasons:

- The portlet has just been initialized and is about to be displayed for the first time
- An action such as someone clicking on a button in the portlet is executed
- Another portlet is rendered, which also renders this portlet again (if Ajax was used this might not happen)

The action and render phase is implemented separately so the `render` method can produce only markup and the `processAction` method focuses on business logic processing. When the portlet is removed from the portal view the `destroy` method is called by the container once.

Portlet modes and window states

A portlet can be rendered in different modes depending on what the user wants to do. There are three modes that need to be handled:

- **View:** This is the mandatory display mode that every portlet needs to support.
- **Edit:** A portlet can also support edit mode where the user can input some data.
- **Help:** A portlet can support a help page.

When a portlet is displayed in the container, a different amount of content can be displayed depending on the current window state for the portlet. There are three window states as follows:

- **Normal:** This is the normal view state where you see the portlet next to the other portlets.
- **Minimized:** In this state, the portlet will be displayed as a single line of text with only the title of the portlet visible.
- **Maximized:** In this state, the portlet takes over the portal completely, and no other portlets are displayed.

Portlet implementation and deployment

A portlet is usually implemented by extending the `javax.portlet.GenericPortlet` as it has predefined methods for the different modes, and it implements the `javax.portlet.Portlet` interface.

The simplest portlet implementation looks something like this:

```
import javax.portlet.GenericPortlet;
import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import java.io.IOException;
import java.io.PrintWriter;

public class HelloWorldPortlet extends GenericPortlet {
    protected void doView(RenderRequest request,
        RenderResponse response) throws PortletException, IOException {
        response.setContentType("text/html");
        PrintWriter writer = response.getWriter();
        writer.println("Hello World!");
    }
}
```

Just as we need to tell a servlet container about a servlet via the `web.xml` file, a portlet also needs to be specified in a deployment descriptor to be picked up by the portlet container. A **portlet descriptor** is specified in the `portlet.xml` file and an example for the `HelloWorldPortlet` looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<portlet-app xmlns=
    "http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/
        portlet-app_2_0.xsd http://java.sun.com/xml/ns/portlet/
        portlet-app_2_0.xsd" version="2.0 ">
    <portlet>
        <description>Hello World</description>
        <portlet-name>HelloWorldPortlet</portlet-name>
        <portlet-class>
            com.bestmoney.cms.web.ui.portlet.HelloWorldPortlet
        </portlet-class>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>VIEW</portlet-mode>
```

```

    </supports>
    <supported-locale>en</supported-locale>
  <portlet-info>
    <title>HelloWorld</title>
    <keywords>Hello, world, test</keywords>
  </portlet-info>
</portlet>
</portlet-app>

```

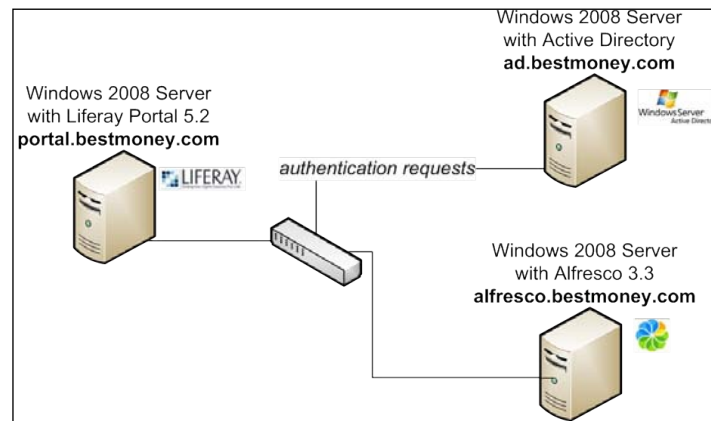
Implementing portlets that display Alfresco content

Here, we will look at implementing two different portlets that fetch recently modified content from Alfresco. These portlets will be configured to deploy in the Liferay portal.

Portal architecture

Best Money uses the Liferay portal running on a dedicated server and it should talk to the remote Alfresco server to fetch document and folder content. The Alfresco system offers an easy way to fetch information remotely through the web scripts framework. With web scripts we can build complete HTML fragments that can be displayed inside the portlet or we can have the web script send back JSON objects or Atom entries, which we can process and display as we like in the portlet.

The following figure shows the Best Money portal architecture:



When the Liferay portlets fetch content from Alfresco they first need to authenticate with the Alfresco server. Both the Liferay portal and the Alfresco server are configured to do all authentication via the Microsoft Active Directory.

Alfresco portlet implementation approaches

When we implement a new portlet that should display content from the Alfresco repository this can be done in a couple of different ways:

- **Alfresco presentation web script:** This approach means that the Alfresco developer will develop both the business logic and the presentation and return an HTML fragment that can be directly displayed in the portlet.
 - **Advantages:** Usually quick to implement as Alfresco comes with many presentation web scripts out of the box; so if one fits the requirement it might be a way to go.
 - **Disadvantages:** Portal developer has no control over look and feel; paging and other links takes you outside the portal.
- **Alfresco data web script:** Here, the Alfresco developer returns just the content in XML or JSON format and the portlet developer has to create the UI for the content returned from Alfresco.
 - **Advantages:** Any type of content can be fetched from Alfresco; data can be formatted in any way using XML, JSON, and so on; portal developer can use any UI framework such as JSP, Grails, Seam, GWT/GXT, and so on; and links can be generated to work within the portal.
 - **Disadvantages:** Portal developers have to code custom parsing of response in some cases.
- **CMIS web script:** This is a standardized way to call a CMS system and fetch data in ATOM publishing format.
 - **Advantages:** Standardized common API based on web scripts, standard XML parsing based on ATOM publishing protocol. Portal developer has control over UI.
 - **Disadvantages:** Usually more complicated APIs, the APIs do not cover all of the features in Alfresco, does not return JSON.

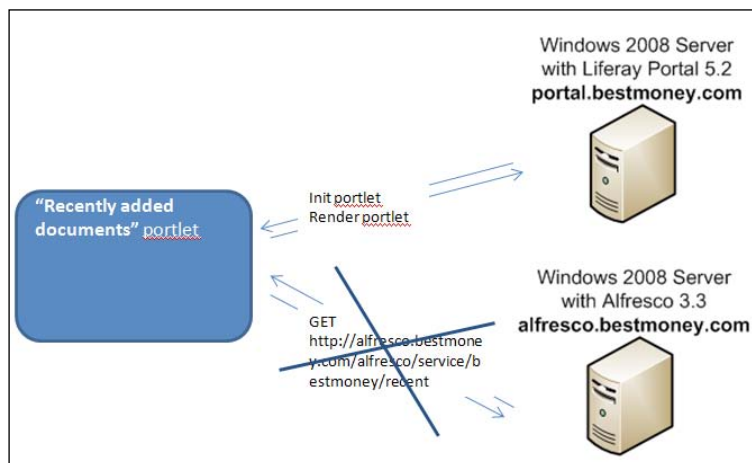
The approach that we will use depends on a lot of different factors. If we, for example, are building a portlet that should display the Alfresco folder hierarchy then it is probably a good idea to use the CMIS approach and build a standard folder hierarchy portlet that can be reused for many different CMS systems. The same goes for a portlet that lists documents in a folder.

On the other hand, if we want to build a portlet that displays workflow information such as assigned tasks, then we would have to use proprietary Alfresco web scripts, as this is not part of the CMIS standard.

The UI framework that we use will also play a major role in deciding what type of web scripts to use. Best Money, for example, has done a lot of its portlet development, and other web application development with **GXT** – a commercial UI Framework built on top of GWT – and wonders if that framework can be used to develop the portlets that should display the content from Alfresco. GXT works very well with JSON or XML, so it is good if the web script can return that as then we would not need to do any manual parsing of responses.

So if we want to use a particular UI framework for developing the portlets, then we will have to use data web scripts. The question then is, should it be a CMIS-based data web script or not? In the case of GXT, we would be better off building our own web scripts returning JSON, as then we can also choose to wrap the returned JSON in a JavaScript function call. And this leads us into the next issue with cross-domain calls.

If the UI framework uses a lot of Ajax calls such as GXT, we also have another issue on our hands. Web applications cannot make HTTP calls to web servers other than the one from which they were loaded – that breaks the security model. So this is a problem in the case of Best Money, how do we use GXT to display Alfresco content when direct calls to the Alfresco server will not be allowed. See the following figure:



In the preceding figure, the `GET http://alfresco.bestmoney.com/...` call will not be allowed by the web browser, not even as a sub-domain to `bestmoney.com`. So that is something we have to get around with a proxy or using a script tag.

If the UI should be run in a mobile or wireless environment, we might have to start thinking about bandwidth consumption and see to it that we send content as efficiently as possible over the air. Then again data web scripts returning JSON might be a good approach as it is not as chatty as Atom.

Implementing the "recently added documents" portlet

This portlet will display all documents and folders that have been created or modified during the day. We will implement two different versions of the portlet—one without GXT, using only Java and HTML and one with GWT/GXT as requested by Best Money.

Implementing the "recently added documents" web script

The first thing that we need to do is create a web script that can be used by the portlet to fetch the content modified today. Create a new web script descriptor called `recent.get.desc.xml` and put it in the `bestmoney\alf_extensions\trunk\_alfresco\config\alfresco\extension\templates\webscripts\com\bestmoney\cms` directory. Open the file and define the web script as follows:

```
<webscript>
  <shortname>Get recently added documents and folders</shortname>
  <url>/bestmoney/recent</url>
  <authentication>user</authentication>
  <transaction allow="readonly"/>
  <format default="json">argument</format>
  <family>BestMoney</family>
</webscript>
```

This web script is *read-only* and returns recently added or modified documents and folders in a JSON structure, if the user has permission to see them (that is, authentication is set to user). Now create the controller for the web script. We will implement the controller with JavaScript in a file called `recent.get.js`:

```
function main() {
  var store_type = "workspace";
  var store_id = "SpacesStore";
  var store = store_type + "://" + store_id;
  var query = "+PATH:\"/app:company_home/*\" AND
    (TYPE:\"cm:content\" OR TYPE:\"cm:folder\") AND NOT
    TYPE:\"cm:systemfolder\" AND @cm\\:modified:NOW";
```

```

    var itemsCreatedToday = search.luceneSearch(store, query);
    var results = {};
    results.content = itemsCreatedToday;
    results.content.sort(sortNames);
    results.totalCount = results.content.length;
    model.results = results;
}
function sortNames(a, b) {
    var nameA = a.name.toLowerCase(), nameB = b.name.toLowerCase();
    if (nameA < nameB) //sort string ascending
        return -1;
    if (nameA > nameB)
        return 1;
    return 0; //default return value (no sorting)
}
main();

```

The way this controller script works is that it does a Lucene search for all content under the /Company Home folder that is of type `cm:content` (that is, documents) or type `cm:folder` (that is, standard folders) and that is created or modified today. When that is done, the content nodes are sorted on the `cm:name` property.

The sorted nodes (that is, the `model`) are then passed on to the script that generates the presentation or view, or as in this case the JSON response, which we will create with a FreeMarker template called `recent.get.json.ftl`:

```

<#assign content = results.content>
<#assign contentLength = results.totalCount>
{ "contentLength":"${contentLength}", "content":[
<#list content as item>
{
    "locked":"${item.isLocked?string}",
    "cmName":"${item.properties.name}",
    "cmTitle":"${item.properties.title!''}",
    "cmDescription":"${item.properties.description!''}",
    "cmModified":"${item.properties.modified?datetime}",
    "cmModifier":"${item.properties.modifier!''}",
    "path":"${item.displayPath!''}",
    "id":"${item.id}",
    "childrenSize":"${item.children?size}"
}
<#if (item_has_next)>,</#if>
</#list>
]
}

```


This FreeMarker template will loop through all the content nodes (that is, results) in the model and create a JSON structure looking like this:

```
{
  "contentLength":"1", "content":[
    {
      "locked":"false",
      "cmName":"Alfresco_Getting_Started_Guide.pdf",
      "cmTitle":"",
      "cmDescription":"",
      "cmModified":"05-Sep-2010 08:03:59",
      "cmModifier":"admin",
      "path":"/Company Home/docs/Alfresco",
      "id":"2c712a49-28a8-448f-87e3-de95c5ff8d52",
      "childrenSize":"0"
    }
  ]
}
```

This is all we need for the web script and we can now test it. Build the Best Money AMP using the `deploy-alfresco-amp` Ant target. The web script will be deployed automatically when the AMP is installed.

To test the web script, it is best to use some external command-line tool, as we might not get it right the first time around. Download the *curl* tool (<http://curl.haxx.se/>) and use the following syntax to test the new web script:

```
C:\tools\curl7.21>curl -v -u admin:admin "http://localhost:8080/alfresco/
service/bestmoney/recent"

* About to connect() to localhost port 8080 (#0)
*   Trying 127.0.0.1... connected
* Connected to localhost (127.0.0.1) port 8080 (#0)
* Server auth using Basic with user 'admin'
> GET /alfresco/service/bestmoney/recent HTTP/1.1
> Authorization: Basic YWRtaW46YWRtaW4=
> User-Agent: curl/7.21.1 (i386-pc-win32) libcurl/7.21.1 OpenSSL/0.9.8o
zlib/1.2.5
> Host: localhost:8080
> Accept: */*
>
< HTTP/1.1 200 OK
< Server: Apache-Coyote/1.1
< Cache-Control: no-cache
```

```
< Pragma: no-cache
< Content-Type: application/json;charset=UTF-8
< Transfer-Encoding: chunked
< Date: Sun, 05 Sep 2010 12:33:50 GMT
<
```

```
{ "contentLength":"8",
  "content": [
    {
      "locked":"false",
      "cmName":"Alfresco",
      "cmTitle":"",
      "cmDescription":"",
      "cmModified":"05-Sep-2010 08:06:59",
      "cmModifier":"admin",
      "path":"/Company Home/docs",
      "id":"e845ac7a-f8c8-4045-bbcd-afd2a8a9d2a5",
      "childrenSize":"5"
    }
  ],
  . . .
}
```

Notice how you can pass in authentication information with the `-u` switch and the `-v` switch will print the request and response information. The *curl* tool is invaluable during development as it gives us a very easy and quick way of testing our new web scripts.



Note that the web script will not return anything if you have not added or modified any documents or folders today.

Now if we see that we need to change something in the FreeMarker templates then we do not have to go through the whole procedure of generating a new AMP and then deploying it and restarting Alfresco. The only thing that we need to do is to open up the `recent.get.json.ftl` template file directly in the exploded war directory and make the necessary changes and save it (see the `tomcat\webapps\alfresco\WEB-INF\classes\alfresco\extension\templates\webscripts\com\bestmoney\cms` directory). Now we can run *curl* again to see how the change affected the output, without restarting Alfresco.

If we need to update the controller then we have to do a bit more. Make the update to the recent `.get.js` file and save it. Then go to `http://localhost:8080/alfresco/service/index` page and click on the **Refresh Web Scripts** button. Now we can run `curl` again to see the changes take effect.

Before going on to develop the portlet, stop Alfresco and configure debug logging for web scripts. This is important when we are not calling Alfresco from the command line with `curl` and have full control over what is being passed on in the URL. When an application is constructing the URL to the web script it is good to know exactly how that URL looks like when it reaches Alfresco. In that way, we will be able to spot any errors if the response is not what we were expecting.

Change the following settings in the `log4j.properties` file (located in the `tomcat/webapps/alfresco/WEB-INF/classes` directory):

```
log4j.logger.org.springframework.extensions.webscripts=debug
log4j.logger.org.springframework.extensions.webscripts.ScriptLogger=debug
log4j.logger.org.alfresco.repo.web.scripts=debug
log4j.logger.org.alfresco.repo.jscript=debug
log4j.logger.org.alfresco.repo.jscript.ScriptLogger=debug
```

This also turns on debug logging for JavaScript.

Implementing a Java-based "recently added documents" portlet

The first version of this portlet will be implemented as a simple Java-based portlet that just creates HTML directly in the `doView` method.

Tools for calling the web service and parsing the response

We are going to need some tool to use when calling the web script and another tool to parse the response. For this, we will use **Apache HTTP Commons** (<http://hc.apache.org/httpclient-3.x/>) and the **Jackson JSON parser** (<http://jackson.codehaus.org/>). In the `bestmoney\alf_clients\portal\web\WEB-INF\lib` directory, we have added the following libraries to be able to use these tools:

```
commons-codec-1.3.jar
commons-httpclient-3.1.jar
commons-logging-1.1.jar
jackson-core-lgpl-1.4.2.jar
```

```
jackson-jaxrs-1.4.2.jar
jackson-mapper-lgpl-1.4.2.jar
jackson-xc-1.4.2.jar
```

Creating the portlet class

Now create a new portlet java class called `RecentlyAddedDocumentsPortlet` and put it in the `com.bestmoney.cms.web.ui.portlet.rad` package in the `bestmoney\alf_clients\portal\source` directory. Start by implementing the `doView` method as follows:

```
public class RecentlyAddedDocumentsPortlet extends GenericPortlet {
    @Override
    protected void doView(RenderRequest renderRequest,
        RenderResponse renderResponse) throws PortletException,
        IOException {
        PrintWriter writer = renderResponse.getWriter();
        String json = callAlfrescoWebScript(
            "http://localhost:8080/alfresco/service/bestmoney/recent",
            writer);
        List<ContentModel> contentItems = readContentJSON(json);
        writer.write(convert2HtmlTable(contentItems));
        writer.close();
    }
}
```

The first thing that we do is to get a writer so we can write HTML to the response stream. We then call the Alfresco web script that we just created and get JSON back with the recently added documents and folders. This JSON is then parsed into a list of `ContentModel` objects (explained later). Finally, we write the list of content items out as an HTML table.

Let's have a look at the method that calls the web script, it looks like this:

```
private String callAlfrescoWebScript(String url, PrintWriter writer) {
    HttpClient client = new HttpClient();
    Credentials defaultcreds = new UsernamePasswordCredentials("admin",
        "admin");
    client.getState().setCredentials(AuthScope.ANY, defaultcreds);
    GetMethod getMethod = new GetMethod(url);
    getMethod.setDoAuthentication(true);

    try {
        int statusCode = client.executeMethod(getMethod);
        if (statusCode == HttpStatus.SC_OK) {
            return getMethod.getResponseBodyAsString();
        } else {

```

```
        writer.write("Got Error" + statusCode + "when calling  
        Alfresco: " + getMethod.getURI());  
    }  
    } catch (HttpException e) {  
        writer.write("Fatal protocol violation: " + e.getMessage());  
    } catch (IOException e) {  
        writer.write("Fatal transport error: " + e.getMessage());  
    } finally {  
        getMethod.releaseConnection();  
    }  
  
    return "";  
}
```

Here we use the Apache HTTP client to make a HTTP GET call to the URL that is mapped to the web script. If we get HTTP Status 200 (`HttpStatus.SC_OK`) back from Alfresco, we return the JSON as is.



Note that normally we should fetch current username and password from the portlet session and then use that instead of admin/admin. As an administrator you have access to everything in Alfresco, which might not be what we want.

The next method is the one that parses the JSON string into a list of content items. It uses the `ContentModel` object, so let's look at this first:

```
public class ContentModel {  
    public String id;  
    public String locked;  
    public String path;  
    public String childrenSize;  
    public String cmName;  
    public String cmTitle;  
    public String cmDescription;  
    public String cmModified;  
    public String cmModifier;  
}
```

It is basically a class with members for each attribute we parse from each entry in the JSON string. The parsing method looks as follows:

```
private List<ContentModel> readContentJSON(String json) {  
    List<ContentModel> contentItems = new ArrayList<ContentModel>();  
  
    if (json == null || json.trim().length() == 0) {
```

```

        return contentItems;
    }

    try {
        ObjectMapper mapper = new ObjectMapper();
        JsonNode rootNode = mapper.readValue(json, JsonNode.class);
        JsonNode contentNodes = rootNode.path("content");

        for (JsonNode itemNode : contentNodes) {
            ContentModel contentItem = new ContentModel();
            contentItem.id = itemNode.path("id").getTextValue();
            contentItem.locked = itemNode.path("locked").getTextValue();
            contentItem.path = itemNode.path("path").getTextValue();
            contentItem.childrenSize =
                itemNode.path("childrenSize").getTextValue();
            contentItem.cmName = itemNode.path("cmName").getTextValue();
            contentItem.cmTitle = itemNode.path("cmTitle").getTextValue();
            contentItem.cmDescription =
                itemNode.path("cmDescription").getTextValue();
            contentItem.cmModifier =
                itemNode.path("cmModifier").getTextValue();
            contentItem.cmModified =
                itemNode.path("cmModified").getTextValue();
            contentItems.add(contentItem);
        }
    } catch (IOException e) {
        System.err.println("Fatal JSON Parsing error: " +
            e.getMessage());
        e.printStackTrace();
    }
    return contentItems;
}

```

It starts off by checking that there is actually some JSON passed in, otherwise the processing will fail. Then we create a mapper that is used to read values from the JSON structure. We start off by getting a reference to the root node in the JSON structure. With the `rootNode` we can just navigate into the structure and get to the content node that contains the list of documents and folders. For each JSON object node, we create a `ContentModel` object and set it up with the values from the JSON node.

When we get the list of content items created, we just need to turn it into HTML, which we will do with the last method that looks like this:

```
private String convert2HtmlTable(List<ContentModel> contentItems) {
    StringBuffer sb = new StringBuffer("<table id=\"docsTable\">");
    sb.append("<thead>");
    sb.append("<tr>");
    sb.append("<th scope=\"col\">Name</th>");
    sb.append("<th scope=\"col\">Path</th>");
    sb.append("<th scope=\"col\">Modified</th>");
    sb.append("<th scope=\"col\">Modifier</th>");
    sb.append("</tr>");
    sb.append("</thead>");
    sb.append("<tbody>");
    for (ContentModel item : contentItems) {
        sb.append("<tr>");
        sb.append("<td>");sb.append(item.cmName);
        sb.append("</td>");
        sb.append("<td>");sb.append(item.path);
        sb.append("</td>");
        sb.append("<td>");sb.append(item.cmModified);
        sb.append("</td>");
        sb.append("<td>");sb.append(item.cmModifier);
        sb.append("</td>");
        sb.append("</tr>");
    }
    sb.append("</tbody>");
    sb.append("</table>");

    return sb.toString();
}
```

This method creates an HTML table and populates it with the list of content items.

Creating the standard portlet deployment descriptor

The portlet is now completed but to be able to deploy it we need to fill in a couple of descriptor files. Let's start with the standard `portlet.xml` file located in the `bestmoney\alf_clients\portal\web\WEB-INF` directory, and add the following portlet descriptor:

```
<portlet>
  <description>Recently added documents presented in simple HTML
    table</description>
  <portlet-name>RecentlyAddedDocumentsPortlet</portlet-name>
```

```

    <portlet-class> com.bestmoney.cms.web.ui.portlet.rad.
        RecentlyAddedDocumentsPortlet
    </portlet-class>
    <supports>
        <mime-type>text/html</mime-type>
        <portlet-mode>VIEW</portlet-mode>
    </supports>
    <portlet-info>
        <title>Recently Added Documents</title>
        <keywords>Recently, added, documents</keywords>
    </portlet-info>
    <security-role-ref><role-name>administrator</role-name>
    </security-role-ref>
    <security-role-ref><role-name>guest</role-name>
    </security-role-ref>
    <security-role-ref><role-name>power-user</role-name>
    </security-role-ref>
    <security-role-ref>
        <role-name>user</role-name>
    </security-role-ref>
</portlet>

```

Here we tell the portlet container that we have a new portlet called `RecentlyAddedDocumentsPortlet` and that it is implemented in the specified class. We also tell the container what security roles should have access to the portlet.

Creating the Liferay portlet deployment descriptor

Now to the Liferay-specific Portlet descriptors; first add an entry into the `liferay-portlet.xml` file:

```

<portlet>
    <portlet-name>RecentlyAddedDocumentsPortlet</portlet-name>
    <instanceable>true</instanceable>
</portlet>

```

We also map the security roles in this file:

```

<role-mapper>
    <role-name>administrator</role-name>
    <role-link>Administrator</role-link>
</role-mapper>
<role-mapper>
    <role-name>guest</role-name>
    <role-link>Guest</role-link>
</role-mapper>

```



```
<role-mapper>
  <role-name>power-user</role-name>
  <role-link>Power User</role-link>
</role-mapper>
<role-mapper>
  <role-name>user</role-name>
  <role-link>User</role-link>
</role-mapper>
```

There is also another file that we need to add the portlet to, so that it is displayed in the UI and available for users to add it via the **Personalize Pages** screen. Open up the `liferay-display.xml` file and add the following:

```
<display>
  <category name="Best Money">
    <portlet id="RecentlyAddedDocumentsPortlet" />
  </category>
</display>
```

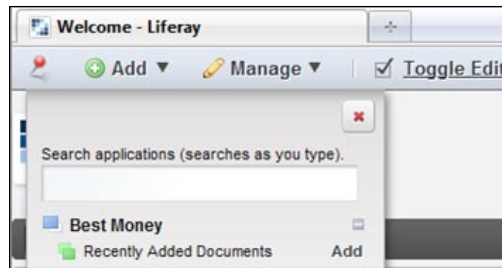
This file also tells the portal which category the portlet will be listed in, such as in this case when it will be listed under the Best Money category.

Building and testing the portlet

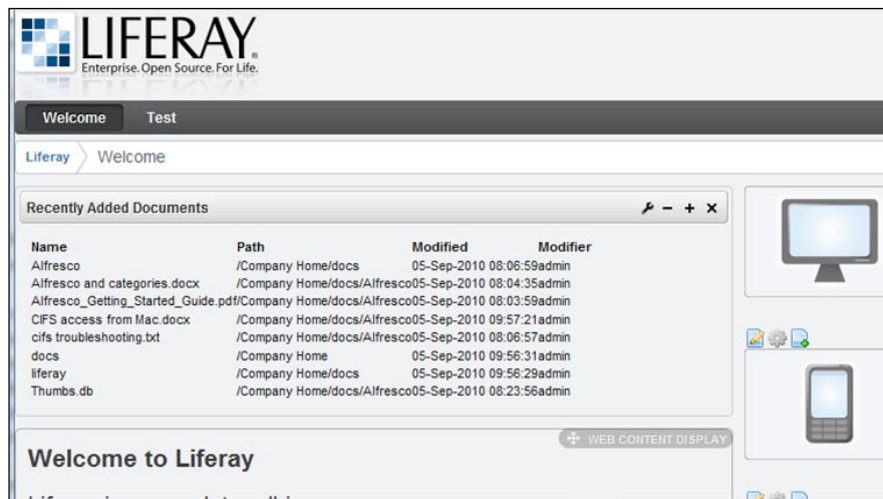
The portlet is now implemented and described for the portlet container, so the last thing to do is to build the Portlet WAR file and deploy it in Liferay. First, set up the properties in the `build.properties` file and then run the `war` Ant target in the `build.xml` file. Both files are located in the `bestmoney\alf_clients\portal` directory.

When we get the `RecentlyAddedDocumentsPortlet.war` we can upload it via the Liferay UI (this is in Liferay version 6.0.5). Log in as the admin user and select **Add | More...**, which displays a window with a list of portlet applications. At the bottom of the list, you can select **Install More Applications**. This displays the Control Panel where you can select the **Upload File** link under **Plugin Installer**. Select the WAR file that we just created and it will install the portlets in it.

Now to add the portlet to a page, select the **Back to Liferay** link at the top. Then select **Add | More...** again, which should now display a new Best Money portlet application as follows:



Select the portlet by clicking on **Add** and it will be added and displayed on the current page:



If you want to run Liferay and Alfresco on the same box, then open the Liferay config file—`server.xml` file located in the `liferay-portal-6.0.5\tomcat-6.0.26\conf` directory and change port numbers by swapping 8 -> 9, so 8080 becomes 9090, and so on.

Implementing a GWT/GXT-based "recently added documents" portlet

In the case of Best Money, they want to base their portlet development on GWT/GXT, so we need to develop the portlet based on this framework. We can still use the same web script.

Creating the portlet class

Now create a new portlet Java class called `RecentlyAddedDocumentsPortletGXT` and put it in the `com.bestmoney.cms.web.ui.portlet.rad` package in the `bestmoney\alf_clients\portal\source` directory. Start by implementing the `doView` method as follows:

```
public class RecentlyAddedDocumentsPortletGXT extends GenericPortlet {
    @Override
    protected void doView(RenderRequest renderRequest,
        RenderResponse renderResponse) throws PortletException,
        IOException {
        renderResponse.setContentType("text/html");
        PrintWriter writer = renderResponse.getWriter();
        String ticket = getAlfrescoTicket("admin", "admin", writer);
        writer.println("<script language='javascript'>" +
            "function getAlfrescoTicket() {\n" +
            "    return '" + ticket + "';" + "}\n</script>");
        writer.println("<script language='javascript' src='" +
            renderRequest.getContextPath() +
            "/recentlyAddedDocumentsApp.nocache.js'></script>");
        writer.println("<div id='" +
            RecentlyAddedDocumentsApp.HTML_DIV_ID + "'></div>");
        writer.close();
    }
}
```

As with the other portlet, we start by getting a writer so we can write HTML to the response stream. We then call a method that will get the admin user a ticket that can be used in subsequent web script calls as an authentication token.

We then create a JavaScript function called `getAlfrescoTicket` that can be called to get current authentication ticket. This is necessary as we can then use this ticket from the GWT code when doing Ajax calls to the web script.

After this, we insert another JavaScript that basically starts the GWT application. The GWT application will insert itself into the `div` with the ID set by the `HTML_DIV_ID` constant defined in the GWT module class `RecentlyAddedDocumentsApp`, which we will implement next.

We also need to implement the `getAlfrescoTicket` method, as follows, using the Apache HTTP client library:

```
private String getAlfrescoTicket(String userName, String password,
    PrintWriter writer) {
    HttpClient client = new HttpClient();
```

```

GetMethod getMethod = new GetMethod(
    "http://localhost:8080/alfresco/service/api/login?u=" +
        userName + "&pw=" + password);

try {
    int statusCode = client.executeMethod(getMethod);
    if (statusCode == HttpStatus.SC_OK) {
        return getMethod.getResponseBodyAsString()
            .replace("<?xml version=\"1.0\" encoding=\"UTF-8\"?>", "")
            .replace("<ticket>", "")
            .replace("</ticket>", "").trim();
    } else {
        writer.write("Got Error " + statusCode + " when calling
            Alfresco: " + getMethod.getURI());
    }
} catch (HttpException e) {
    writer.write("Fatal protocol violation: " + e.getMessage());
} catch (IOException e) {
    writer.write("Fatal transport error: " + e.getMessage());
} finally {
    getMethod.releaseConnection();
}

return "";
}

```

We use the `http://localhost:8080/alfresco/service/api/login` URL, which is standard for getting an Alfresco ticket that can be used as an authentication token, so we do not have to log in when doing the web script call. The ticket is returned in some XML, so we strip that off before we return the ticket as a string.

Create the GWT module class

Now create a new class that will represent the GWT entry point and implement the `EntryPoint` interface, call it `RecentlyAddedDocumentsPortletApp` and put it in the `com.bestmoney.cms.web.ui.portlet.rad.gwt.client` package in the `bestmoney\alf_clients\portal\source` directory.

Start by defining some constant that we will need for JSON properties and div ID, and so on:

```

public class RecentlyAddedDocumentsApp implements EntryPoint {
    public static final AppIcons ICONS = GWT.create(AppIcons.class);
    public static final String HTML_DIV_ID = "recentlyAddedDocs";
    public static final String ROOT_JSON_PROPERTY = "content";
}

```

```
public static final String LENGTH_JSON_PROPERTY = "contentLength";
public static final String ID_PROPERTY = "id";
public static final String LOCKED_PROPERTY = "locked";
public static final String PATH_PROPERTY = "path";
public static final String CHILDREN_SIZE_PROPERTY = "childrenSize";
public static final String CM_NAME_PROPERTY = "cmName";
public static final String CM_TITLE_PROPERTY = "cmTitle";
public static final String CM_DESC_PROPERTY = "cmDescription";
public static final String CM_MODIFIED_DATE_PROPERTY =
    "cmModified";
public static final String CM_MODIFIER_PROPERTY = "cmModifier";
public static final int MAX_ROWS_PER_PAGE = 25;
public static final String DATETIME_FORMAT =
    "dd-MMM-yyyy HH:mm:ss";
```

Now implement the `onModuleLoad` method as follows, to create the complete UI for the portlet:

```
public void onModuleLoad() {
    String url =
        "http://localhost:8080/alfresco/service/bestmoney/recent";
    ScriptTagProxy<PagingLoadResult<ModelData>> proxy =
        new ScriptTagProxy<PagingLoadResult<ModelData>>(url);

    ModelType type = getModelType();
    JsonPagingLoadResultReader<PagingLoadResult<ModelData>> reader =
        new JsonPagingLoadResultReader<PagingLoadResult<ModelData>>(type);
    final PagingLoader<PagingLoadResult<ModelData>> loader =
        new BasePagingLoader<PagingLoadResult<ModelData>>(proxy, reader);

    setupBeforeLoadListener(loader);
    loader.setSortDir(Style.SortDir.DESC);
    loader.setSortField(CM_NAME_PROPERTY);
    loader.setRemoteSort(true);

    final ListStore<ModelData> store =
        new ListStore<ModelData>(loader);

    Grid<ModelData> grid = createGrid(store, loader);
    ContentPanel panel = createPanelForGrid(grid, loader);
    RootPanel.get(HTML_DIV_ID).add(panel);
}
```

The most important bit to note here is the use of the `ScriptTagProxy`, which enables us to do cross-domain HTTP calls. It will require the web script that we use to be updated a little bit, so that it can wrap the JSON in a JavaScript function call. The script tag proxy will create a `<script src=...` and put the URL to call the web service as the source. Then when the JSON is returned as a JavaScript, it needs to be wrapped in a function.

We then go on to create a `ModelType` that will represent the format of the JSON returned from the web script. The GXT grid that we use will use a store and a loader and the loader in turn will use the script tag proxy to get to the JSON data. The result returned from the script tag proxy is paged via the reader that we have defined.

We also set up a load listener (that is, `setupBeforeLoadListener`) that will be called just before the script tag proxy will make its call to the web script, giving us a chance to add extra parameters to the HTTP Get.

The last thing that we do is create the grid and a panel for it and then add the panel to the div we defined in the portlet class implementation.

The `setupBeforeLoadListener` method is implemented as follows to add the extra `alf_ticket` parameter:

```
private void setupBeforeLoadListener(
    PagingLoader<PagingLoadResult<ModelData>> loader) {
    loader.addListener(Loader.BeforeLoad,
        new Listener<LoadEvent>() {
            public void handleEvent(LoadEvent loadEvent) {
                BasePagingLoadConfig loadConfig =
                    loadEvent.<BasePagingLoadConfig>getConfig();
                loadConfig.set("alf_ticket",
                    getAlfrescoTicket());
            }
        });
}
```

And the `getAlfrescoTicket` method is implemented as a native JavaScript method that calls the JavaScript method that we defined in the portlet class:

```
public native String getAlfrescoTicket()
/*-{
    return $wnd.getAlfrescoTicket();
}-*/
;
```

The rest of the methods that we call in the `onModuleLoad` method are pure GXT code and if you are interested, it can be downloaded together with the build file for the whole GWT/GXT project from the book website.

Updating the "recently added documents" web script

The script tag proxy that we are using will send a parameter with the `callback` function name that it wants the web script to wrap the JSON in. We need to update the web script controller and FreeMarker template so that it can handle this.

Start with the controller `recent.get.js` and add the following:

```
function main() {
    var store_type = "workspace";
    var store_id = "SpacesStore";
    var callbackFunctionName = args["callback"];
    var store = store_type + "://" + store_id;
    var query = "+PATH:\"/app:company_home/*\" AND
        (TYPE:\"cm:content\" OR TYPE:\"cm:folder\") AND NOT
        TYPE:\"cm:systemfolder\" AND @cm\\:modified:NOW";
    var itemsCreatedToday = search.luceneSearch(store, query);

    var results = {};
    results.callbackFunctionName = callbackFunctionName;
    ...
}
```

Then update the FreeMarker template, `recent.get.json.ftl`, and add the following:

```
<#assign content = results.content>
<#assign contentLength = results.totalCount>
<#if results.callbackFunctionName??>
    <#assign callback = results.callbackFunctionName>
</#if>

<#if (callback??)>${callback}(</#if>
    . . .
    "childrenSize": "${item.children?size}"
}
    <#if (item_has_next)>,</#if>
</#list>
]
}
<#if (callback??)>);</#if>
```

Now, test the new web script as follows:

```
C:\tools\curl7.21>curl -v -u admin:admin http://localhost:8080/alfresco/service/bestmoney/recent?callback=testFunc
```

```
testFunc(
{ "contentLength":"8",
"content": [
{
"locked":"false",
"cmName":"Alfresco",
"cmTitle":"",
"cmDescription":"",
"cmModified":"05-Sep-2010 08:06:59",
"cmModifier":"admin",
"path":"/Company Home/docs",
"id":"e845ac7a-f8c8-4045-bbcd-afd2a8a9d2a5",
"childrenSize":"5"
}}]);
```

Creating the standard portlet deployment descriptor

The portlet is now completed, but to be able to deploy it we need to fill in a couple of descriptor files. Let's start with the standard `portlet.xml` file located in the `bestmoney\alf_clients\portal\web\WEB-INF` directory; add the following portlet descriptor:

```
<portlet>
  <description>Recently added documents presented with
    GWT/GXT</description>
  <portlet-name>RecentlyAddedDocumentsPortletGXT</portlet-name>
  <portlet-class>
    com.bestmoney.cms.web.ui.portlet.rad.
    RecentlyAddedDocumentsPortletGXT
  </portlet-class>
  <supports>
    <mime-type>text/html</mime-type>
    <portlet-mode>VIEW</portlet-mode>
  </supports>
  <portlet-info>
    <title>Recently Added Documents (GXT)</title>
    <keywords>Recently, added, documents, GXT</keywords>
  </portlet-info>
  <security-role-ref>
    <role-name>administrator</role-name>
```



```
</security-role-ref>
<security-role-ref>
  <role-name>guest</role-name>
</security-role-ref>
<security-role-ref>
  <role-name>power-user</role-name>
</security-role-ref>
<security-role-ref>
  <role-name>user</role-name>
</security-role-ref>
</portlet>
```

Here we tell the portlet container that we have a new portlet called `RecentlyAddedDocumentsPortletGXT` and that it is implemented in the specified class. We also tell the container what security roles should have access to the portlet.

Creating the Liferay portlet deployment descriptor

Now to the Liferay-specific portlet descriptors; first add an entry into the `liferay-portlet.xml` file:

```
<portlet>
  <portlet-name>RecentlyAddedDocumentsPortletGXT</portlet-name>
  <instanceable>true</instanceable>
  <header-portlet-css>/css/gxt-all.css</header-portlet-css>
  <header-portlet-css>/css/gxt-gray.css</header-portlet-css>
  <header-portlet-css>/gwt/standard/standard.css</header-portlet-css>
</portlet>
```

There are a couple of extra stylesheets used by the GXT and GWT frameworks, so we add them here. There is also another file that we need to add the portlet to, so that it is displayed in the UI and available for users to add it via the personalize pages screen. Open up the `liferay-display.xml` file and add the following:

```
<display>
  <category name="Best Money">
    <portlet id="RecentlyAddedDocumentsPortlet" />
    <portlet id="RecentlyAddedDocumentsPortletGXT" />
  </category>
</display>
```

This file also tells the portal which category the portlet will be listed in, such as in this case when it will be listed under the Best Money category.

Building and testing the portlet

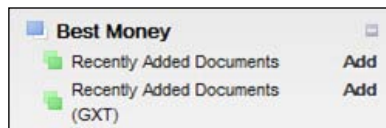
The portlet is now implemented and described for the portlet container, so the last thing to do is to build the Portlet WAR file and deploy it in Liferay. First, set up the properties in the `build.properties` file and then run the `war` Ant target in the `build.xml` file. Both files are located in the `bestmoney\alf_clients\portal` directory.

When we get the `RecentlyAddedDocumentsPortlet.war` we can upload it via the Liferay UI. Log in as the admin user and upload as we did with the other portlet.

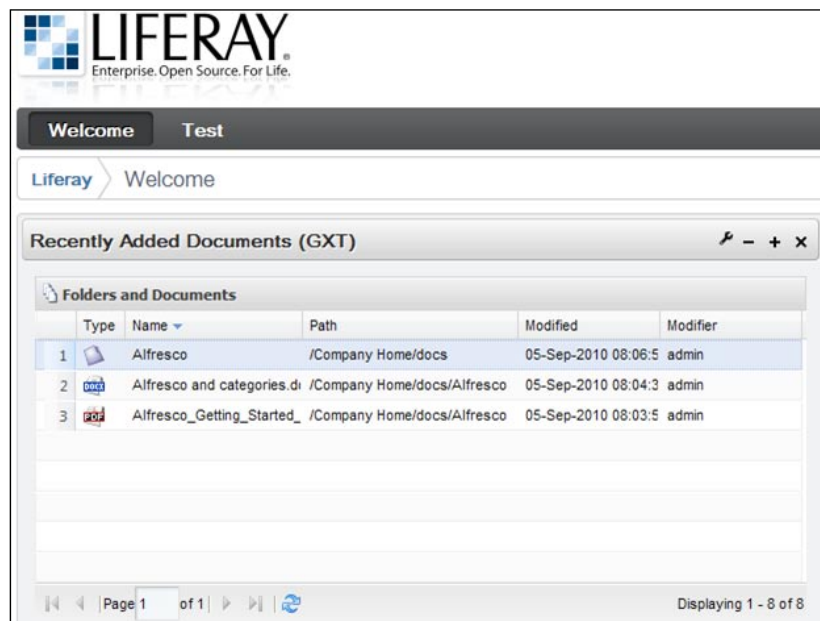


To update an already installed portlet application, remove it from the `tomcat/webapps` directory and then install it again.

Now to add the new portlet to a page, select **Add | More...** again, which should now display a new Best Money portlet application as follows:



Select the GXT version of the portlet by clicking on **Add** and it will be added and displayed on the current page:



When working with GWT/GXT, it is useful to install Mozilla Firebug JavaScript debugger for Mozilla Firefox. In this way, we can see if something goes wrong, what HTTP calls are being made; do some debugging, and so on.

Summary

This chapter has shown us that it is quite easy to extract content from Alfresco and build *mashups* in a portal like Liferay. We created two different types of portlets – one that displayed content with plain HTML and another portlet that used the more Ajax-friendly GWT/GXT framework.

The HTML portlet fetched content from Alfresco with Apache HTTP Client and Jackson JSON parser. All requests were made from the server-side, so there was no problem with cross-domain requests. The GXT portlet on the other hand made Ajax calls from the client to get the Alfresco content, this caused problems as these calls are cross-domain calls, which are not allowed. We solved this by using a `script` tag proxy.

Both these portlets used a custom web script and the controller for it used a JavaScript to return content created or modified the same day by using a Lucene query. We used the *curl* tool to test the web script and we could see that this tool is quite handy when it comes to testing web scripts during development.

We also saw how we can handle authentication so the user does not have to log into Alfresco when he or she is already logged into the portal.

In the next chapter, we will have a look at how we can also integrate and manage e-mail content in Alfresco, which is becoming more and more important in the enterprise.

13

Types of E-mail Integration Solutions

It is becoming more and more common that an ECM solution should include the possibility of storing e-mails in the repository, so they can be managed and searched in the same way all other content can. The long-term vision for most ECM systems is to be able to handle almost any kind of content, and for many organizations e-mails are often next in line to manage after content such as documents, images, web content, and records.

When we talk about managing e-mails in the content management system, it is important to know exactly what we mean by that. Today most companies and organizations want to use Alfresco for e-mail archiving, which is not something that is easily supported out of the box. Alfresco can be used as an e-mail management solution and we will discuss what the difference is between that and an e-mail archiving solution.

In this chapter, we will look at the advantages and disadvantages between three different e-mail integration solutions:

- E-mail client talking directly to Alfresco via the IMAP protocol
- E-mail client talking to Alfresco via custom built plugin and Web Scripts
- E-mail server talking to Alfresco via custom module and Web Scripts

You will also learn how to use Alfresco's built in IMAP solution to:

- Enable dragging-and-dropping of e-mails into the Alfresco repository
- Enable e-mail attachment extraction
- Enable viewing of document metadata from the e-mail client
- Set up different folder mount points
- Enable e-mail management in an Alfresco Share site

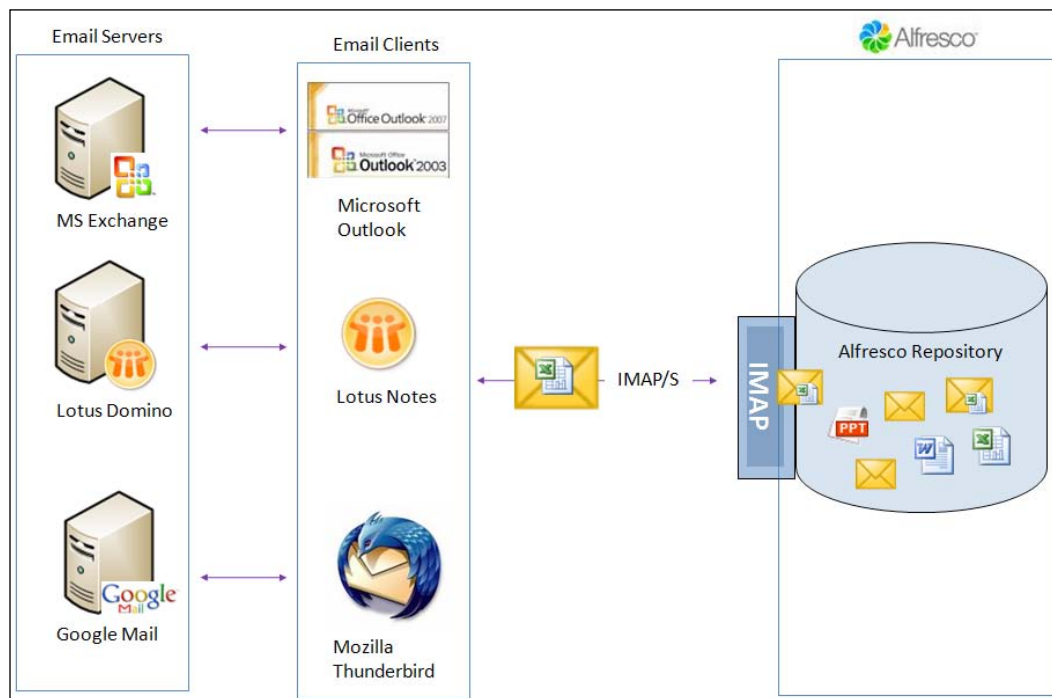
E-mail integration solutions

There are a number of different ways that an e-mail system can be integrated with the Alfresco CMS system. We will look at three of these and present advantages and disadvantages with each one.

E-mail client talking directly to Alfresco via the IMAP protocol

This is the solution that is available out of the box with Alfresco. From version 3.2 and onwards Alfresco supports the IMAP protocol, which is one way an e-mail client can talk to e-mail servers (the other way is POP). So, with this solution Alfresco can behave like an IMAP e-mail server.

The following image illustrates how this solution works:



The e-mail clients typically receive an e-mail in their Inbox and then they can drag-and-drop that e-mail into an Alfresco folder via the IMAP channel. Any attachment can be extracted and handled separately to the e-mail in the Alfresco repository.

This is a manual process that requires the end user to manage what e-mails he or she wants to be stored in Alfresco. Nothing happens automatically and no e-mails are stored in Alfresco unless a user manually drag-and-drops them there.



To achieve automatic archiving of e-mails, a user could set up an e-mail rule in their e-mail client that automatically files some or all e-mails into an Alfresco folder. However, we would still have to manually set up this rule on all users' e-mail clients. So we could not say that this would be an archiving solution that is transparent to the user, as it does not automatically force all e-mails to be saved for auditing purposes. Further on, the e-mail client has to be running in order for the e-mail rule to execute.

This solution is best thought of as an e-mail management solution where users collaborate and share information in e-mails.

The advantages of this solution are:

- **No client installation:** In most e-mail clients we can set up an extra IMAP account connecting to Alfresco without the need to install any extra software on the client workstation. This includes Outlook, Lotus Notes, Mozilla Thunderbird, and GroupWise.
- **Users don't have to change working style:** This is a big thing, users do not want to start learning a complete new way of managing e-mails, they just want to work in the same way they always have. The Alfresco account just shows up as another e-mail inbox in the e-mail client. Users can drag-and-drop e-mails between mailboxes just as they normally do. They do not have to learn any extra functionality.
- **Supported out-of-the-box:** No need to install any extra Alfresco modules, just configure some properties and the solution is ready to go.

The disadvantages of this solution are:

- **No document search:** Users cannot search for documents in Alfresco and then attach them to an e-mail they want to send.
- **Cannot set custom metadata:** Because this solution does not use any custom plugin on the e-mail client side there is no possibility of setting custom metadata for an e-mail, such as for example customer ID, before it is stored in Alfresco. However, you can often solve this problem by creating business rules on the server-side and apply custom metadata based on which folder an e-mail is dropped into.

- **No archiving solution:** This is an e-mail collaboration and e-mail sharing solution, it does not force e-mail to be stored in the repository for compliance and regulatory reasons.

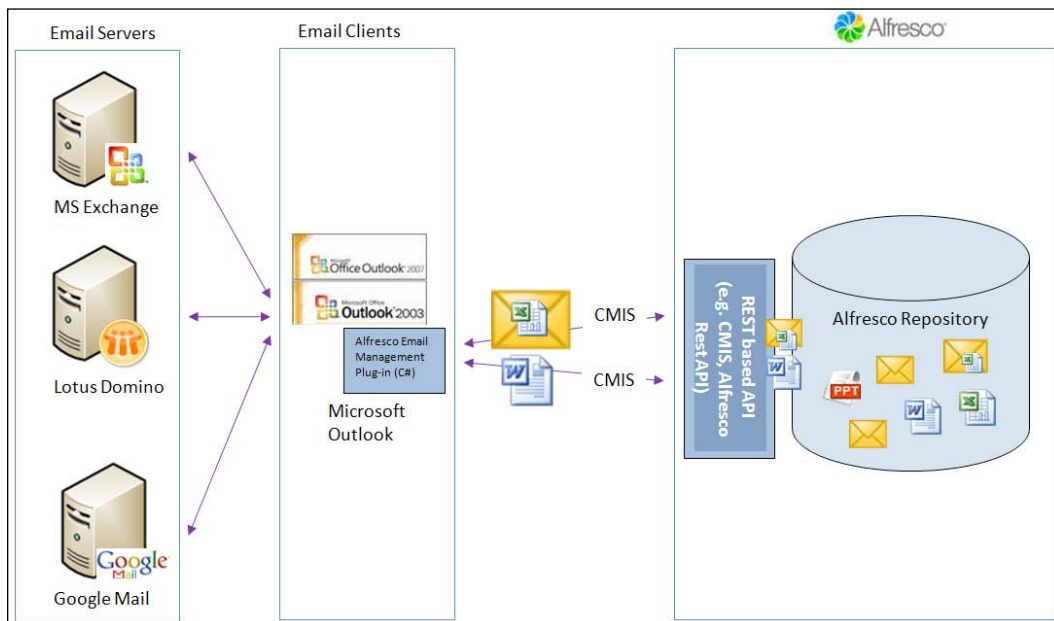
Because this solution doesn't require any client installation, or updates to the Alfresco server, it will probably be the most popular e-mail management solution. It can also easily be extended with folder rules to create sophisticated e-mail filing solutions.

E-mail client talking to Alfresco through custom built plugin and Web Scripts

There are one or two products out there that have taken a different approach to integrating e-mail clients with Alfresco. One of these products is Anovio Email Management solution for Outlook 2007 (<http://www.anovio.de/aem>). This product provides a solution that enables you to also work with documents from the e-mail client, and search for documents via the e-mail client.

To do this they had to implement a plugin for the e-mail client that is almost exclusively Outlook, and use Web Scripts to talk to Alfresco. The IMAP channel approach is not used as it can only handle e-mails.

The following picture gives us an overview of this solution:



This solution is also an e-mail management solution as it is up to the end user to actually save the e-mail into the repository. There is no automatic archiving going on.

The advantages of this solution are:

- **Document search:** You can do a full text search for documents in the repository via the e-mail client. A document can then be attached to an e-mail that is about to be sent.
- **Users don't have to change working style:** Users can drag-and-drop e-mails into the Alfresco repository in a way they are used to. They do not have to use, and learn, the extra document management functionality in the Outlook plugin if they do not want to.
- **Stores attachments directly:** Attachments can be stored directly into the repository without storing the e-mail.
- **Usually works with other CMS systems:** The solution from Anovio, mentioned above, works with other CMS systems as it uses CMIS.

The disadvantages of this solution are:

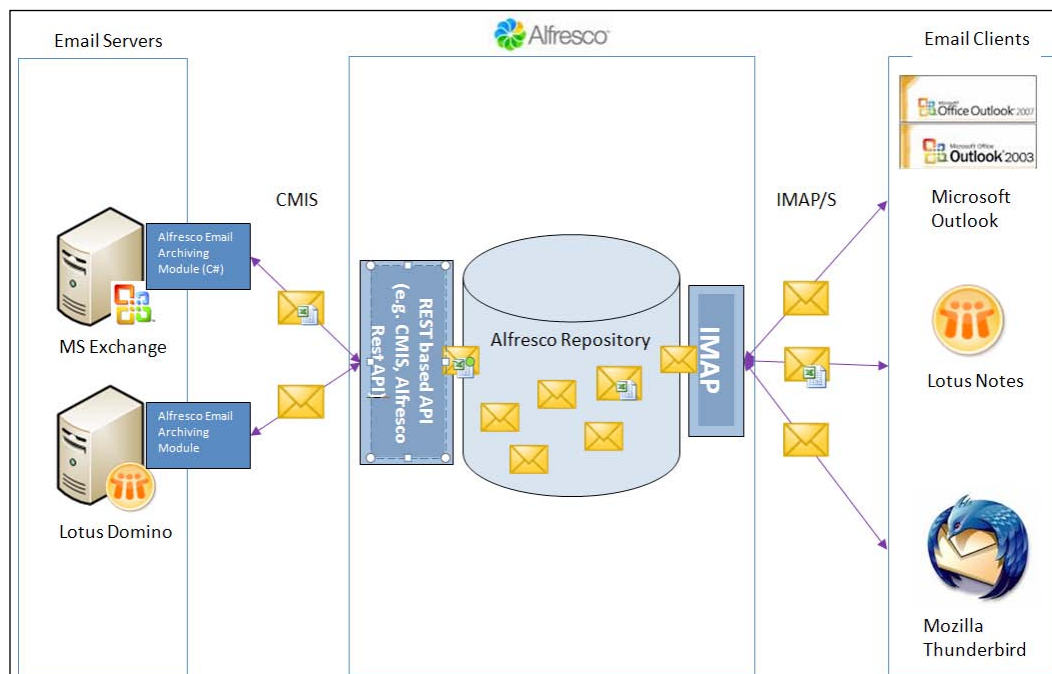
- **Client installation:** It requires you to install this Outlook plugin on every PC that in most cases is a show stopper as it brings along too much support work for the IT support department.
- **Does not work for all e-mail clients:** It works only with certain e-mail clients, such as for example Outlook 2007, in the case of the Anovio product.
- **Users have to learn new functionality:** If users want to handle documents from the e-mail client then they have to learn new functionality. Also, there are usually new menus and features users have to learn even for the standard e-mail management functionality.
- **No archiving solution:** This is an e-mail collaboration and sharing solution, it does not force e-mail to be stored in the repository for compliance and regulation reasons.
- **Not supported out of the box:** It is not a part of the Alfresco package, so will need to be purchased separately.

This kind of solution can be very good for users that frequently need to attach documents from the Alfresco repository to e-mails that they are sending. However, if there is a larger user base, the maintenance burden could be quite substantial as you would need to install the plugin on every user's PC.

E-mail server talking to Alfresco through custom module and Web Scripts

This is the classical e-mail archiving solution where the e-mail system integration has been done on the e-mail server-side. This solution is totally transparent for the end users and usually complies with security regulations. What this means is that all e-mails are archived automatically without the user having to do anything, which guarantees that every incoming and outgoing e-mail has been filed and can be audited later.

There are – unfortunately, at the time of writing – no such solutions available for Alfresco. But for reference purposes this is how such a solution would typically look:



This solution would require us to build an extension module for the e-mail server that captures all inbound and outbound e-mails and stores them in Alfresco without the users having to do anything. So all e-mails are captured and stored for archiving and auditing purposes.

Users can then for example, access the e-mails through the standard IMAP channel, if they are stored as standard MIME messages according to RFC-822 (<http://tools.ietf.org/html/rfc822>).

The advantages of this solution are:

- **Supports archiving and auditing:** This is the only solution that would be compliant with security regulations as users are not involved, and cannot decide if an e-mail should be stored or not.
- **Users don't have to change working style:** Users can use their standard e-mail client to view archived e-mails.

The disadvantages of this solution are:

- **Requires server installation:** We need to have access to the e-mail server and be able to install the integration module. This might be challenging in many situations when you might not be allowed to install anything on the e-mail server, or the e-mail server might be hosted externally so we would not have access to it.
- **Attachments are not extracted:** The attachments would probably not be extracted and sorted into their own subfolder. This is assumed as the purpose of an e-mail archiving solution to store the complete original e-mail for auditing reasons, and not for e-mail management use.
- **Not a collaboration and sharing solution:** E-mails are stored in an archiving structure and not in a project or case structure. Users would have more difficulty in collaborating around e-mail content.
- **Duplicate e-mails exist:** There would be a lot of duplicate e-mails because of security regulations such as Sarbanes-Oxley that requires all e-mails to be stored for auditing purpose, even if it is a duplicate.
- **Not supported out of the box:** It is not a part of the Alfresco package so will need to be purchased separately, if it is available.

This solution is mentioned here so we can easily tell the difference between an e-mail management solution and an e-mail archiving solution when we discuss this with potential clients. There has been a lot of misunderstanding around what e-mail integration solutions are currently available for Alfresco, where they are sometimes referred to as e-mail archiving solutions, which they are not.

Implementing e-mail management solutions

Best Money has a couple of e-mail management requirements that they would like to implement. The first one is that they want to be able to use the MS Outlook e-mail client for dragging-and-dropping e-mails straight into Alfresco folders. They also want to be able to do automatic filing of e-mails based on patterns in the e-mail subject.

The solutions we will implement in this chapter will be based on the IMAP channel integration approach, which is the only solution available out of the box with Alfresco.

Implementing e-mail management solutions with Alfresco IMAP

We will start with a drag-and-drop solution that uses the out of the box IMAP server.

Configure Alfresco to enable the IMAP server

To enable the IMAP server we just have to set a couple of properties as follows in the `alfresco-global.properties` file:

```
imap.server.enabled=true
imap.server.port=143
imap.server.host=mbergljung-PC
```



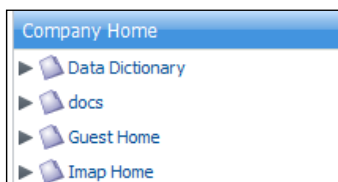
When specifying the `imap.server.host` don't use `localhost` as that will not work. The IMAP server will be bound to the wrong network interface and you will not be able to successfully connect to the server from the e-mail client. An IP address would also work here.

Now start the server and the following should appear in the logs:

```
15:59:06,400 INFO [repo.imap.AlfrescoImapServer] IMAP service
started on host:port mbergljung-PC:143.
```

The Alfresco server is now ready to act as an IMAP server for e-mail clients to connect to. It listens to IMAP port 143 that is for plain non-secure connections. The Alfresco IMAP solution does not, at the time of this writing, support secure connections on port 993.

To test the Alfresco IMAF solution, we first need a user that we can set up an account for in Microsoft Outlook. When we log in to Alfresco Explorer we will also see a new top folder called **Imap Home** as shown in the following screenshot:



This folder will be populated with mailboxes for the different users that connect through IMAF. For this example, we have set up a new user called martin.

Setting up an IMAF account in Outlook 2007

In the Outlook 2007 client, we set up a new IMAF account by selecting **Tools** followed by **Account Settings...** in the drop-down menu. Then select **New...** in the **Account Settings** dialog.

In the **Choose E-mail Service** dialog, we select **Microsoft Exchange, POP3, IMAF,** or **HTTP** and then click **Next...** and here we click the checkbox called **Manually configure server settings or additional server types**. In the next dialog window, choose **Internet E-mail** and then click **Next**. We should now see the following dialog:

Add New E-mail Account

Internet E-mail Settings
Each of these settings are required to get your e-mail account working.

User Information

Your Name: Martin
E-mail Address: martin.bergjung@opsera.com

Server Information

Account Type: IMAF
Incoming mail server: 192.168.0.2
Outgoing mail server (SMTP): 192.168.0.2

Logon Information

User Name: martin
Password: ****
☒ Remember password
☐ Require logon using Secure Password Authentication (SPA)

Test Account Settings
After filling out the information on this screen, we recommend you test your account by clicking the button below. (Requires network connection)

Test Account Settings

When all fields are filled in we can test the connection by clicking this button.

More Settings ...

< Back Next > Cancel

Fill in the values as in the previous screenshot, use whatever username and password you created through Alfresco Explorer. Fill in the same hostname or IP for the SMTP server as for the IMAP server, this field needs to be specified even if we are not going to send e-mails from this account.

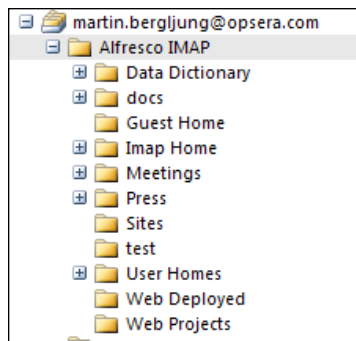
If the IMAP connection test is not successful when you test it via the **Test Account Settings...** button then first try and log in to Alfresco Explorer with the username and password that was used. If that works then check what IP address the IMAP server is bound to as follows:

```
C:\Users\mbergljung>netstat -an|find "143"
```

TCP	192.168.0.2:143	0.0.0.0:0	LISTENING

Now make sure that the same IP address is used as Incoming mail server, and when the connection test is successful click on the **Next** button to finish the setup of the new IMAP account.

We will see something like this in Outlook at this point:



The folders we see correspond to the same folders that user **martin** has permission to see when he logs in via the Alfresco Explorer user interface.

 Note that it might take a minute or two before Outlook has updated the account so you can actually see all these folders. So do not panic if you do not see these folders at once, wait a bit and they should pop up.

Alfresco's IMAP solution will by default subscribe the user to all folders they have access to in Alfresco. This way the user does not have to subscribe to the folders before they are visible (that is in Outlook we can subscribe to folders via the **IMAP Folders...** menu item).



Users cannot unsubscribe to folders on an individual basis as the subscribe/unsubscribe feature in the Alfresco IMAP server is implemented globally for all users. So if for example, the admin user unsubscribes to a folder then all other users will also be unsubscribed to this folder.

If we need users to be able to subscribe and unsubscribe to folders on an individual basis then we can look at the OpsMailmanager product that supports this.

If we now log in to Alfresco Explorer, we will see some new folders created under the **Imap Home** top folder. These folders are only created for the user when he or she accesses the system via an IMAP connection:

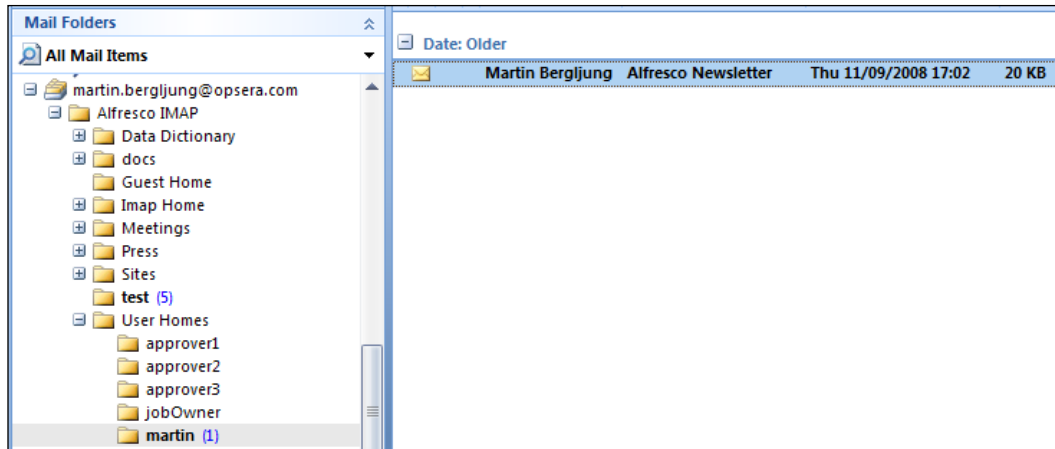


These folders are special folders that are created by the E-mail client, such as for example the **Junk E-mail** mailbox that Outlook creates for all new accounts. Alfresco needs to store these "special" mailboxes somewhere and this is why the **Imap Home** top folder has been created.

Drag-and-drop e-mail into Alfresco folder in Outlook 2007

Now when we have everything set up, it is a simple task of dragging-and-dropping an e-mail from the standard Outlook Exchange Inbox into one of the folders under /Company Home that we have write permissions to.

For the initial testing of the drag-and-drop feature we can use the user's Alfresco home folder, as we can be sure that the user will have write permissions to it. In the following picture, we have dragged an e-mail from the Outlook Exchange Inbox to the user's Alfresco home folder / Company Home/User Homes/martin:



As we can see, the way we use the Alfresco e-mail account is exactly the same way we use any other e-mail account in Outlook. Just drag-and-drop e-mails between the accounts. If you want to keep the e-mail in the Outlook Exchange Inbox, and also store it in Alfresco, then you have to use copy and paste instead of drag-and-drop that will always move the e-mail.

Viewing the e-mail from Alfresco Explorer

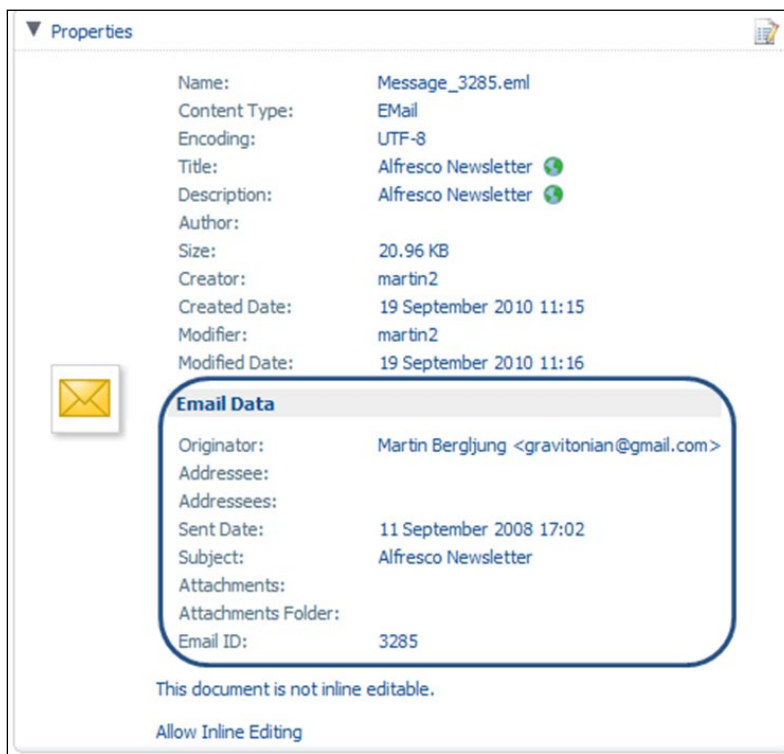
Now, when we have an e-mail stored in the Alfresco repository it would be interesting to see how it is presented there. Log in to Alfresco Explorer with the same username and password that was used to set up the Alfresco IMAP account in Outlook, and then click on the **My Home** link at the top. You should see the e-mail displayed as follows:

Name ▲	Description ▾	Size ▾	Created ▾	Modified ▾	Actions
Message_3285.eml	Alfresco Newsletter	20.96 KB	19 September 2010 11:15	19 September 2010 11:16	   

The name of the e-mail node will have the format Message_<sequence number>.eml and cannot easily be changed without customizing the Alfresco source code. There is no naming plugin that we can implement to customize the naming convention of these e-mail nodes.

If we click on the link, (that is, `Message_3285.eml`) a preview page is displayed with all the properties of the e-mail and the e-mail body. This e-mail will now also be indexed and included in content searches.

If we click on the **View Details** button, we will see that a couple of extra e-mail metadata have been added to the node:



With the OpsMailmanager product a naming plugin can be created where the e-mail node name can be formatted in whatever way we want.

E-mail attachment extraction

The Alfresco IMAP solution supports e-mail attachment extraction so when an e-mail with one or more attachments is dragged-and-dropped into an Alfresco folder all the attachments are extracted into a subfolder:

Name ▲	Description ●	Created ●	Modified ●	Actions
Message_3299.eml-attachments		19 September 2010 11:50	19 September 2010 11:50	
Message_3302.eml-attachments		19 September 2010 11:50	19 September 2010 11:50	

One subfolder per e-mail will be created. E-mail attachment extraction is turned on by default; if you look in `repository.properties` (located in `webapps/alfresco/WEB-INF/classes/alfresco`) you will find the following property that controls this:

```
imap.server.attachments.extraction.enabled=true
```



If for some reason attachment extraction is not working even though it is clearly turned on in the `repository.properties` file, then you can try and put the `imap.server.attachments.extraction.enabled` configuration in `alfresco-global.properties` instead – that has worked for me a couple of times.

If you would like to store all extracted attachments into the same subfolder, control what attachment types are extracted (Word, Excel, PPT, and so on), and what folders are enabled for attachment extraction, then have a look at the OpsMailmanager product.

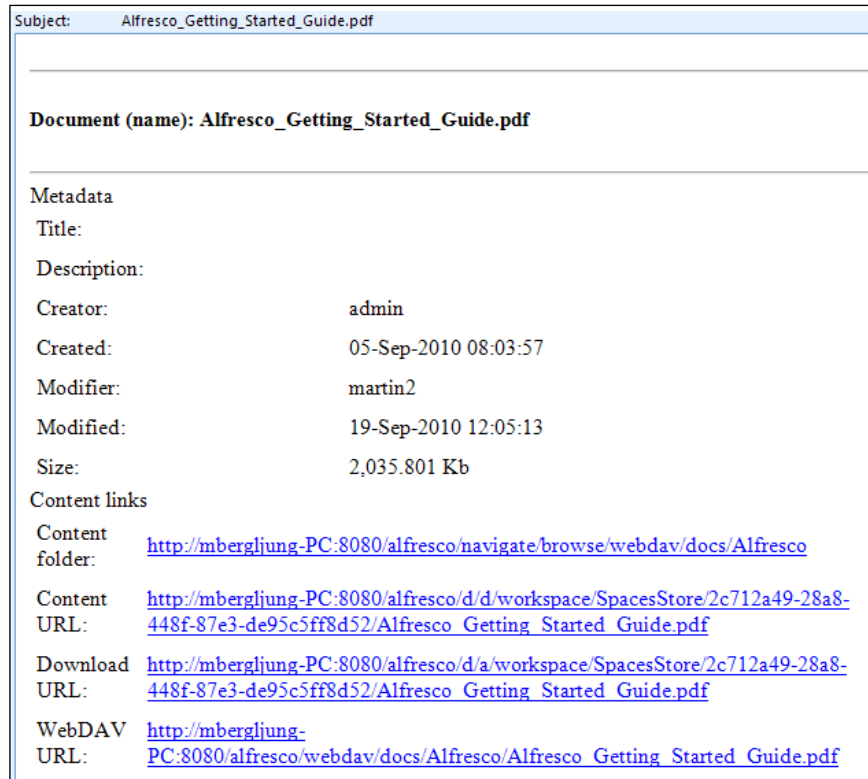
Viewing document metadata from the e-mail client

One thing that is really cool is that you can browse the metadata for documents from the e-mail client and then decide to download a document if you like. This can be quite useful for mobile devices.

The way this works is that whenever a folder contains documents the metadata for them is wrapped into e-mails. This happens dynamically when you click on a folder. The metadata e-mails are then generated and you will see something like this in Outlook:

Mail Folders		Date: Today			
All Mail Items			admin	Alfresco_Getting_Started_Guide.pdf	Sun 19/09/2010 12:05 6 KB
martin.bergljung@opsera.com			admin	Thumbs.db	Sun 19/09/2010 12:05 6 KB
Alfresco IMAP			admin	Alfresco and categories.docx	Sun 19/09/2010 12:05 6 KB
Data Dictionary			admin	cifs troubleshooting.txt	Sun 19/09/2010 12:05 6 KB
docs (2)			mbergljung	CIFS access from Mac.docx	Sun 19/09/2010 12:05 6 KB
Alfresco (5)					

If we open one of these e-mails, we will see the metadata for the document and links to download the document and much more:



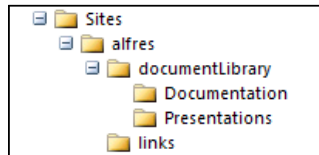
The e-mail body is controlled by a FreeMarker template that we can customize if we do not want to show all the information in the above e-mail body. This template is called `emailbody-textplain.ftl` and can be found in the `/Company Home/Data Dictionary/Imap Configs/Templates` folder.

Dragging-and-dropping e-mails into Alfresco Share site

If we look in the /Company Home/Sites folder we will see that it is empty even if there are Alfresco Share sites that the user is a member of. A user who wishes to upload e-mails into an Alfresco Share site has to select it as an IMAP favorite in the Alfresco Share UI:



Then the Alfresco Share site will show up in the e-mail client as follows:



Here we have set up a Share site called `alfres` and the document library has been updated with two new subfolders. A user will not be able to add any e-mails into these folders until he or she has been invited to the Share site as at least a Contributor.

To be able to view metadata for documents in the document library the user must be at least a Collaborator in the site.



If a Share site does not show up in the e-mail client after we have selected it as an IMAP favorite in Alfresco Share, then we need to manually force an update of the folder list. In Outlook, we can do that by right-clicking on the account name and then selecting **Update Folder List**.

How to use Mount Points

So far a user can see all folders in Alfresco that he or she has read access to and the user can upload e-mails to any folder where he or she has write access. Further on, document metadata will be displayed for documents in any folder where the user is a Collaborator.

This is probably not what we want as e-mail clients (that is, the IMAP protocol) are usually not built to handle thousands of folders. What we want to do is mount certain folders as e-mail management folders and certain folders as document metadata viewing folders.

In Alfresco we can manage this by using the so-called Mount Points. There are three types of Mount Points (http://wiki.alfresco.com/wiki/IMAP#Mount_Point_Modes) that we can use:

- **ARCHIVE:** Using a folder mount point of this type enables the folder for e-mail management (that is, we can write and read e-mails in this folder)
- **VIRTUAL:** This mount point type enables the folder for document metadata viewing by dynamically generating a metadata e-mail/document
- **MIXED:** It is a combination of ARCHIVE and VIRTUAL

The Alfresco IMAP system has one default mount point called `AlfrescoIMAP` defined with mount point type set to `MIXED`. The configuration looks like this:

```
imap.config.server.mountPoints=AlfrescoIMAP
imap.config.server.mountPoints.default.mountPointName=IMAP
imap.config.server.mountPoints.default.modeName=ARCHIVE
imap.config.server.mountPoints.default.store=${spaces.store}
imap.config.server.mountPoints.default.rootPath=/${spaces.company_
home.childname}
imap.config.server.mountPoints.value.AlfrescoIMAP.
mountPointName=Alfresco IMAP
imap.config.server.mountPoints.value.AlfrescoIMAP.modeName=MIXED
```

All properties in the `imap.config.server.mountPoints.default` namespace are default mount point property definitions. If we are happy with them then we do not need to specifically set them when defining a new mount point.

The `AlfrescoIMAP` mount point is using `MIXED` mount point type/mode and as the default mount point type/mode is set to `ARCHIVE` we need to specifically override it and set `MIXED`:

```
imap.config.server.mountPoints.value.AlfrescoIMAP.modeName=MIXED
```

The mount point name is also overridden to be `Alfresco IMAP` instead of just `IMAP`.

Now let's define two new mount points as follows:

- **Meeting e-mails:** `ARCHIVE` mount point with root path set to `/Company Home/Meetings`
- **Documents:** `VIRTUAL` mount point with root path set to `/Company Home/docs`

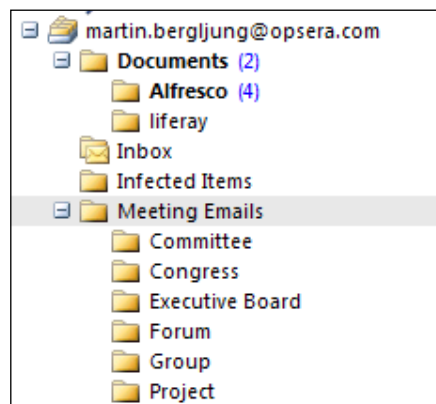
The configuration for these two mount points look like this:

```
imap.config.server.mountPoints=MeetingEmails,Documents
imap.config.server.mountPoints.value.MeetingEmails.
mountPointName=Meeting Emails
imap.config.server.mountPoints.value.MeetingEmails.rootPath=/${spaces.
company_home.childname}/Meetings
imap.config.server.mountPoints.value.Documents.
mountPointName=Documents
imap.config.server.mountPoints.value.Documents.modeName=VIRTUAL
imap.config.server.mountPoints.value.Documents.rootPath=/${spaces.
company_home.childname}/docs
```

The first thing we do when creating new mount points is to give them a name (not the visible name but a name to use when configuring them) and then specify these names in the `imap.config.server.mountPoints` property. By excluding the `AlfrescoIMAP` mount point from the property value it will no longer be active.

Then we override the default mount point values as needed. The `MeetingEmails` mount point has not specified any mount point type/mode, as the default `ARCHIVE` setting is correct.

After restarting the Alfresco server, we will see something like the following in the Outlook client:



As we can see, this view now looks a lot cleaner for the user and filters out all unnecessary folders and content.

Summary

This chapter has explained the different e-mail integration solutions that are available for Alfresco. The most common one is the support of the IMAP protocol in the Alfresco server that makes it easy for e-mail clients to configure a connection to Alfresco and drag-and-drop e-mails into the repository. No client software installation is required.

There are also other products, such as Anovio Email Management Solution, that require a plugin installation on the client-side, but also allows document management and search from the e-mail client.

We have learned that e-mail archiving is not currently supported for Alfresco and the solutions that are available for Alfresco are e-mail management solutions. They require the user to be involved in storing an e-mail in Alfresco, and this is in contrast to an e-mail archiving solution that is transparent to the users and stores all e-mails automatically.

In the next chapter, we will look at what is available for a mobile CMS user who wants to access content in Alfresco.

14

Mobile Phone Access Solutions

The number of people who use mobile phones to access enterprise applications increases every year. IDC (www.idc.com) forecasts that by 2013 the mobile worker population will reach nearly 1.2 billion. This is boosted by the fast-paced development of next generation mobile networks referred to as 4G – the successor of 3G – and the increasing number of public Wi-Fi hotspots. The proliferation of smartphones also contributes by providing a greater experience when accessing applications.

There is also a new generation of workers who are accustomed to always being connected and being able to work with content wherever they are. So it is important that an ECM system like Alfresco can offer multiple ways for users to access its content from a mobile device.

Best Money has a lot of mobile workers and they need access to the documents in Alfresco – when out of the office – from their BlackBerry and iPhone devices. This chapter goes through how you can set up a solution to support this.

In this chapter, you will learn:

- The mobile application support that is available with Alfresco
- Creating a mobile application with Grails that uses CMIS to talk to Alfresco
- Building CMIS requests and responses

Alfresco mobile web application for iPhone

Alfresco comes with a web application that can be used to access content from an iPhone. In this section, we will see how it works and what features are and aren't available.

Installing the Alfresco mobile web application

The Alfresco mobile web application is delivered as part of the standard community download in the `mobile.war` file and when the Alfresco server is started, this web application is also deployed.



Note that the mobile application is not delivered in the Alfresco Community Edition 3.3g or 3.4b download. You would have to resort to Community version 3.3 or compile the source code to get to it. Likewise, for Alfresco Enterprise Edition 3.3.0, 3.3.1, 3.3.2, the mobile application is not delivered with these versions. Alfresco is working on a new mobile client but it is not available at the time of this writing.

Accessing the Alfresco mobile web application

The mobile application is accessible via the `http://<hostname>:8080/mobile` URL and the first screen you will see is the "login" screen as follows:





Some versions of the Alfresco mobile web application require you to access it with the `http://<hostname>:8080/mobile/p` URL.

After logging in, the following screen will be displayed:



Here, we can see that the Alfresco Mobile web application is focused on presenting a mobile interface for the Alfresco Share web application.

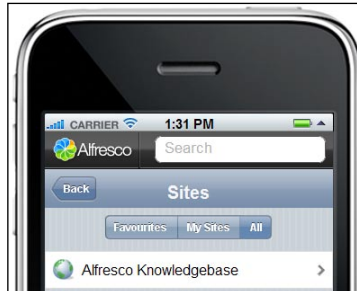


There is no way of accessing content in the standard DM repository. Only content under `/Company Home/sites` can be accessed.

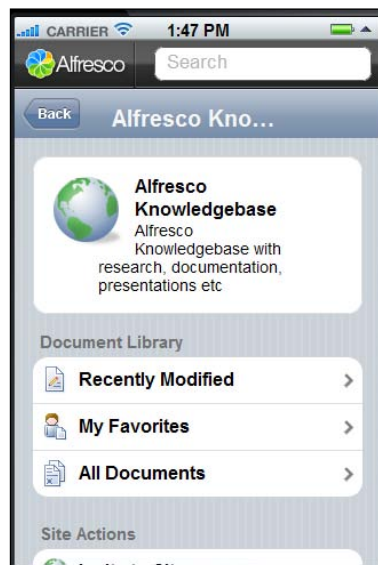
So, to be able to see something when we click on **Sites** in the previous screen, we have to first create a site via Alfresco Share. I have created a site to store information around Alfresco called Alfresco Knowledgebase. If we click on **Sites**, the following will be displayed:



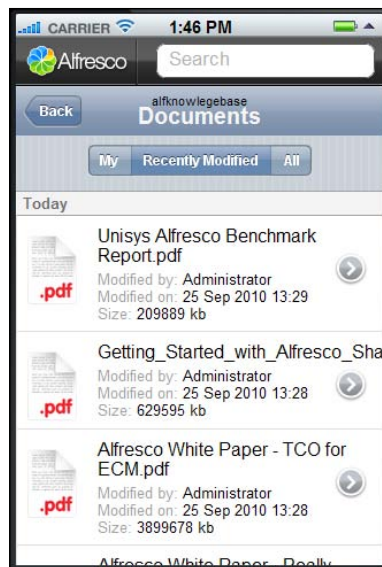
Now click on the **All** button and the Alfresco Knowledgebase site (or whatever sites you have created) should be displayed.



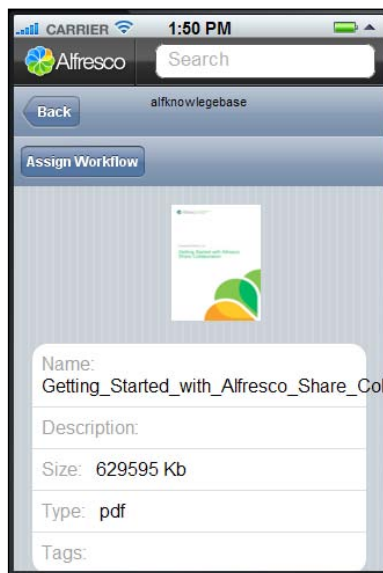
If we click on the **Alfresco Knowledgebase** row, it will take us to the main page for this site, as follows:



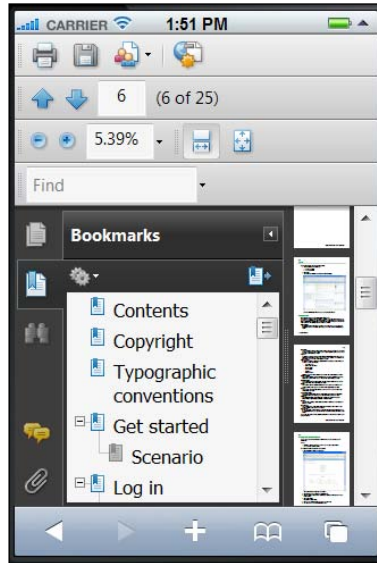
Here, we can see that it is only the **Document Library** that is exposed from all the features available in Alfresco Share. No other features such as wiki, calendar, blogs, data lists, and so on are available. To list available documents in the **Document Library** click on **All Documents** and the following type of page will be displayed:



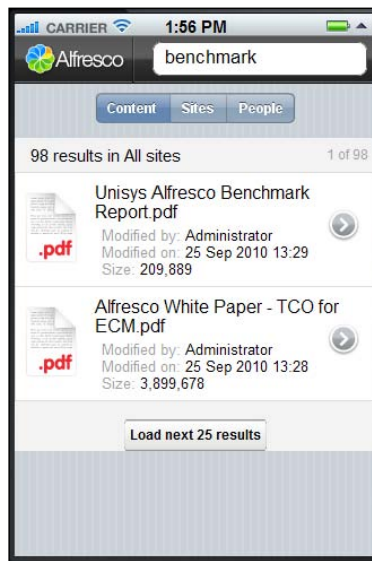
Clicking on one of the documents will display a page with a document thumbnail icon and metadata information:



From this page you can start a "Review & Approve" workflow by clicking on the **Assign Workflow** button, and you can load the document into *preview* mode by clicking on the document icon:



It is also possible to search for documents, if for example, I search for the term **benchmark**, I get the following result:



I can search for people and sites in the same way.

If your company is already using Alfresco Share as a project collaboration platform then the Alfresco mobile web application might be a good starting point—it does not offer all the features of Alfresco Share but at least it gives the users instant access to the documents when they are out and about.

A custom mobile application solution for smartphones

The Alfresco mobile web application for iPhone is the only application available from Alfresco that is targeted at mobile phones. Unfortunately, it does not give us access to the whole repository and it is specifically implemented for the iPhone device. Best Money's BlackBerry users want to be able to access all folders under `/Company Home`, so we need to find another solution.

The best thing in this case is probably to create a custom Alfresco client that can be accessed via most mobile phones. Then we would have the freedom to include whatever functionality the end users want.

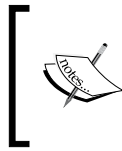
This might sound like a bad idea. Are we not going to end up with lots of extra code to maintain for this new custom application? And are we really going to be product developers? Should we not focus on the business domain and consulting solutions and let Alfresco do the product development?

It sure is going to mean extra code to maintain and develop further. But wouldn't we rather have some extra code to maintain than a very unhappy customer? We can find a middle way—a good tool that enables us to create this kind of application very fast.

In this section, we will use the Grails framework (<http://www.grails.org/>) to develop a small web application that can be used by most smartphones available today. We will call this application `MobileX`.

The combination of Groovy and Grails will enable us to create this `MobileX` application very rapidly as the Grails framework uses convention over configuration, making it really quick to put together web applications. And the Groovy scripting language makes the code very small and neat.

If you are not familiar with Groovy, I recommend spending 1-2 hours reading a tutorial about it before implementing the `MobileX` application (<http://groovy.codehaus.org/Beginners+Tutorial>).



The code used to build the MobileX application is based on an implementation originally developed by Robin Bramley at Ixxus. He has agreed to let me use it as a demonstration for this chapter. Thanks Robin!

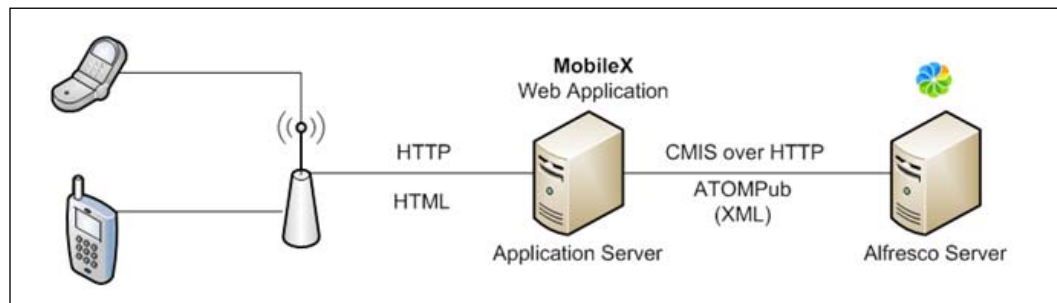
Mobile application architecture overview

When implementing web applications that are targeted at mobile phones it is important to think about what content we send **over-the-air (OTA)**. Mobile devices are restricted in how much memory they have, screen size, CPU speed and so on, so it is important to keep the payload of our communication to a minimum and the screen layout small.

The MobileX web application will be using CMIS to talk to Alfresco and then process the response from the CMIS requests into small HTML pages that are then sent to the mobile device over HTTP.

Most mobile devices today, at least smartphones, are capable of handling HTTP and HTML and this is a lot different from the time (a couple of years back) when you had to use **Wireless Markup Language (WML)** and **Wireless Application Protocol (WAP)** to build web applications for mobile phones.

The architecture for MobileX looks like this:



In this architecture, the MobileX web application and the Alfresco web application are deployed on different servers. This does not have to be the case – they could both be deployed in the Tomcat instance running on the Alfresco Server.

Mobile application feature overview

We will implement the following features in the `MobileX` web application.

User authentication

The first thing that we need to sort out is how users will be authenticating with Alfresco via this new mobile application. All users need to be authenticated before they can access any folders or documents in the repository.

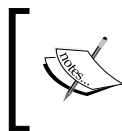
The `MobileX` application will present a login screen whenever the user is trying to access any application URL unauthenticated. The login screen will look like this:



The users will enter their normal Alfresco username and password and then press the **Login** button.

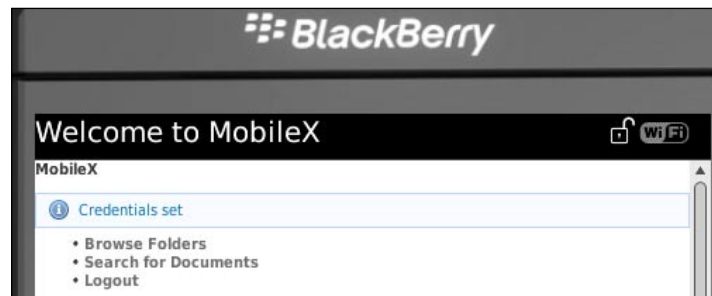
For this application, the BlackBerry Bold 9700 Simulator has been used. It is always best to try out our mobile application in a proper simulator so that we can see exactly how the UI is going to look.

This BlackBerry simulator can also emulate a connection to a 3G mobile network but during development it is easiest to connect to the local WLAN. To do that, we have to scan for the WLAN and then set up a profile; this is done via **WiFi Protected Setup (WPS)**, if the router supports that.



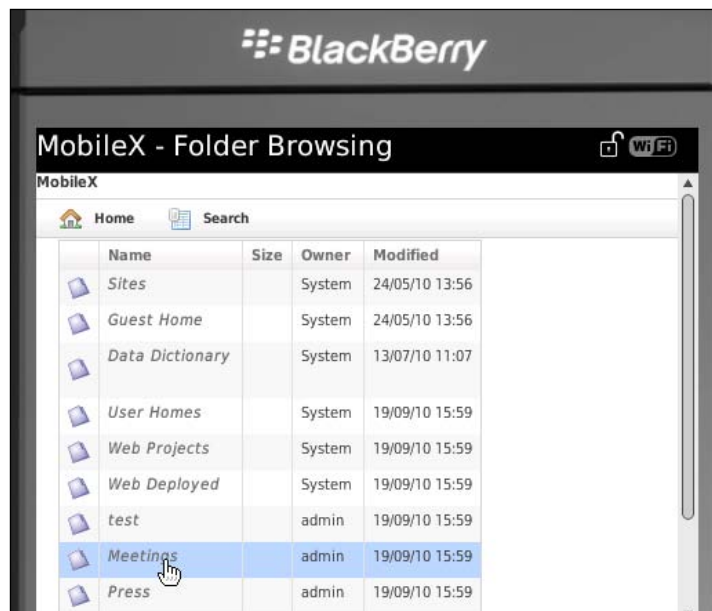
The BlackBerry simulators will not just let us go to the browser and type in an URL as in most other phone emulators. We have to set up a proper network connection first.

When the user has successfully logged in he or she is presented with a menu as follows:

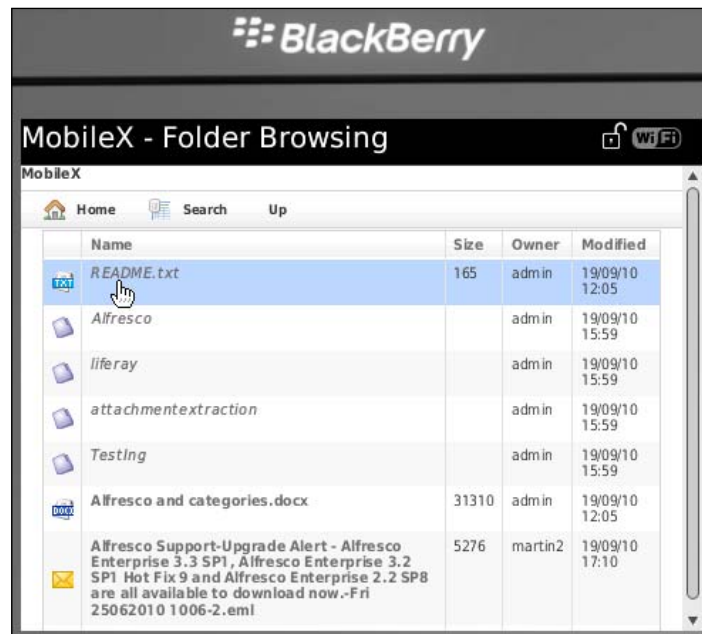


Folder and document browsing

In the main menu, a user can select **Browse Folders** and when first clicked the user is presented with the top-level folders in the Alfresco repository, as follows:



Clicking on one of these folders navigates into that folder. If there are documents and e-mails in the folder then they will also be displayed as follows:

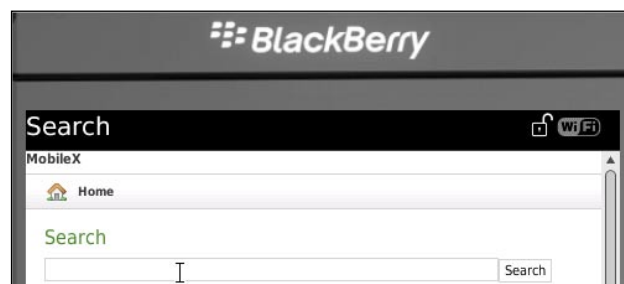


Clicking on one of the content items kicks off the download via the internal mobile device browser and the content item, such as a Word document is processed by the network provider's server, which might be a **BlackBerry Enterprise Server (BES)**. The processed version of the content item is then displayed.

Clicking on the **Up** button navigates to the parent folder and clicking on the **Home** button navigates back to the initial main menu.

Document search

If the user clicks on the **Search** button the following dialog is displayed:



Here the user can do a full text search and content name search for any word entered into the text input field. Clicking on the **Search** button executes the search. The search result is presented in a list.

Setting up the mobile Grails application

Start by downloading Grails (version 1.3.4 was used for this example) and unpacking it into a directory of your choice. Then set the `GRAILS_HOME` variable and add the `%GRAILS_HOME%/bin` directory to the system `PATH` (<http://grails.org/doc/latest/guide/2.%20Getting%20Started.html>).

Now create the Grails MobileX application in the `3340_14_Code\bestmoney\alf_clients` directory:

```
3340_14_Code\bestmoney\alf_clients>grails create-app mobilex
```

After that create the following controllers, filters, and services that we are going to need:

```
3340_14_Code\bestmoney\alf_clients\mobilex>grails create-controller  
com.bestmoney.mobilex.controllers.Authentication
```

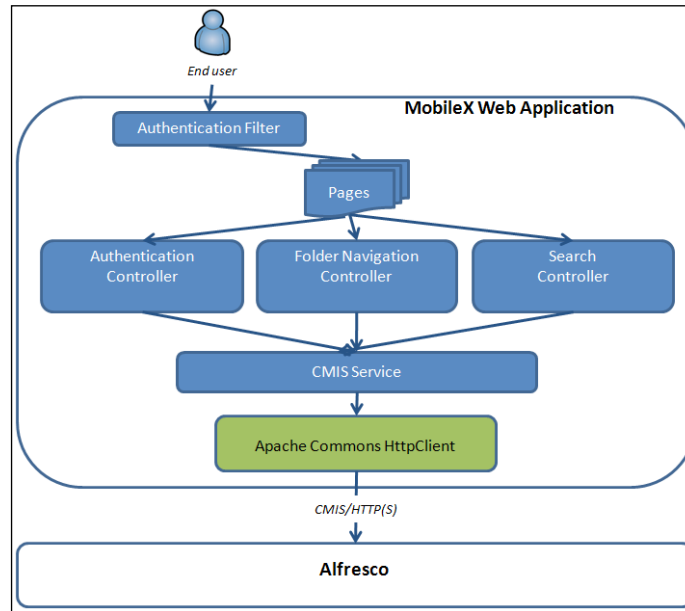
```
3340_14_Code\bestmoney\alf_clients\mobilex>grails create-controller  
com.bestmoney.mobilex. controllers.FolderNavigation
```

```
3340_14_Code\bestmoney\alf_clients\mobilex>grails create-controller  
com.bestmoney.mobilex. controllers.Search
```

```
3340_14_Code\bestmoney\alf_clients\mobilex>grails create-filters  
mobilex.authentication
```

```
3340_14_Code\bestmoney\alf_clients\mobilex>grails create-service  
com.bestmoney.mobilex.services.Cmis
```

The architecture of the MobileX application looks something like this:



Configuring the mobile Grails application

We are going to create our own configuration section called `mobilex` in the `grails-app/conf/Config.groovy` file. It will define well-known Alfresco URLs, XML namespaces, and so on. Put it under the development environment section (<http://grails.org/doc/latest/guide/3.%20Configuration.html>) as follows:

```

environments {
    production {
        grails.serverURL = "http://www.changeme.com"
    }
    development {
        grails.serverURL = "http://localhost:8081/${appName}"

        mobilex {
            serverBase = "http://localhost:8080/alfresco/service"
            alfrescoApiUrl = "${serverBase}/api"
            cmisUrl = "${serverBase}/cmis" // Also gives access to CMIS
            // Service document
            cmisQueryUrl = "${cmisUrl}/queries"
            adminUsername = "admin" // Used to lookup CMIS Service document
            adminPwd = "admin"
        }
    }
}

```

```
// childrenPath will be appended with <node-guid> then
//children
childrenPath = "${cmisUrl}/s/workspace:SpacesStore/i"
cmisNamespaces =
    [cmis: 'http://docs.oasis-open.org/ns/cmis/core/200908/',
     cmisra: 'http://docs.oasis-open.org/
       ns/cmis/restatom/200908/',
     alf: 'http://www.alfresco.org',
     app: 'http://www.w3.org/2007/app',
     opensearch: 'http://a9.com/-/spec/opensearch/1.1/']
    atomVersion = 'http://www.w3.org/2005/Atom'
}
}
test {
    grails.serverURL = "http://localhost:8081/${appName}"
}
}
```

The username and password for the Alfresco administrator is kept in this configuration as it is needed the first time we talk to Alfresco to get the CMIS service document. We also set up the MobileX web application to be started on port 8081, so that we can run the Alfresco server at the same time locally on port 8080.

Implementing the CMIS service

Let's start implementing the CMIS service as it is going to be needed when we implement the controllers. Open up the `CmisService.groovy` class located in `alf_clients\mobilex\grails-app\services\com\bestmoney\mobilex\services`.

There are not going to be any write operations performed by this sample implementation, so set the `transactional` variable to `false`. Make sure a CMIS service instance is created for the scope of the `session`. The CMIS service should also implement the `org.springframework.beans.factory.InitializingBean`, so we can read the CMIS service document when the bean is loaded by Spring. And finally, load the application configuration that we specified in the `Config.groovy` file:

```
class CmisService implements InitializingBean {
    static transactional = false
    static scope = "session"
    def config = ConfigurationHolder.config
```

To communicate with Alfresco via CMIS we are going to use the Apache HTTP client (download the `commons-codec.jar`, `commons-httpclient.jar`, and `commons-logging.jar` and place them in the `alf_clients\mobilex\lib` directory). Create an Apache HTTP client variable like this:

```
def httpClient = new HttpClient()
```

The user's credentials will be stored in the `httpClient` object during the session and that is one reason why the service scope needs to be set as `session`. The last member variable that we are going to define will contain the current root folder (that is, `/Company Home`) node reference for the Alfresco repository instance we are talking to:

```
def rootFolderNodeRef
```

Fetching the folder root node reference from the CMIS service document

The root folder node reference is set up in the Spring initialization method by fetching the so-called CMIS service document, which is available at the `http://<hostname>:<port>/alfresco/service/cmis` URL. This service document looks something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<service xmlns=http://www.w3.org/2007/app
  xmlns:atom=http://www.w3.org/2005/Atom
  xmlns:cmisra=http://docs.oasis-open.org/ns/cmis/restatom/200908/
  xmlns:cmis=http://docs.oasis-open.org/ns/cmis/core/200908/
  xmlns:alf="http://www.alfresco.org">
  <workspace>
    <atom:title>Main Repository</atom:title>
    <collection href=
      "http://localhost:8080/alfresco/service/cmis/s/
        workspace:SpacesStore/i/26cef395-d9c5-40f7-b1f4-
        6e4954462adb/children">
      <atom:title>root collection</atom:title>
      <cmisra:collectionType>root</cmisra:collectionType>
    </collection>
    . . .
```

First, we can see the same namespaces declared that we defined in the configuration file. Then inside the `workspace` element there are a number of `collection` elements that contain URLs for different types of operations that we might want to execute. These are URLs for getting all folders and documents under `/Company Home`, getting the types supported by the repository, doing searches, and so on.

We are interested only in the collection type called `root` that contains the Alfresco node reference for the `root` folder in the repository. This is how we implement the code to extract the `root` folder node reference:

```
void afterPropertiesSet() {
    def usercreds = new
        UsernamePasswordCredentials(config.mobilex.adminUsername,
            config.mobilex.adminPwd)
    httpClient.getState().setCredentials(AuthScope.ANY, usercreds)

    def xmlString = executeGetRequest(config.mobilex.cmisUrl)
    def cmisDocXml = new
        XmlSlurper().parseText(xmlString).declareNamespace(
            config.mobilex.cmisNamespaces)

    def rootFolderChildrenURL
    cmisDocXml.workspace.collection.each {
        if (it.collectionType.text() == "root") {
            rootFolderChildrenURL = it.@href as String
        }
    }

    def rootFolderNodeId =
        rootFolderChildrenURL.substring(
            rootFolderChildrenURL.indexOf('/i/') + 3,
            rootFolderChildrenURL.lastIndexOf('/'))
    rootFolderNodeRef = "workspace://SpacesStore/${rootFolderNodeId}"

    httpClient.getState().clear()
}
```

First, we set up the `httpClient` object with the Alfresco administrator credentials and then we execute a HTTP GET request with the `executeGetRequest` method (which we will implement later) to get the CMIS service document. We then use the very handy Groovy class `XmlSlurper` to parse the returned XML. We loop through all `collection` elements under the `workspace` element until we find the `root` one.

The collection's `href` attribute is then extracted from the XML document as a string. To get the node reference from the `href` URL, we search for `/i/`, which is followed by the node reference identifier. We then put together the complete node reference by prefixing the ID with the store. The store is hardcoded here but could also be extracted from the `href` URL by searching for `/s/`.

The final thing that we do is clear the admin user credentials from the `httpClient`.

We are also going to need a method that can be used by a controller to get to the root folder node reference:

```
def getRootFolderNodeRef() {
    return rootFolderNodeRef
}
```

Authenticating the user with Alfresco

Authentication is not specified in the CMIS specification and it just says "Authentication SHOULD be handled by the transport protocol." We want a custom login screen and not the one presented by the device browser, and the user should be able to log out of the session. This means that we need a login and a logout method that the `AuthenticationController` can use.

The login method will take a username and a password and set them up in the `httpClient`. The `httpClient` will then be used to authenticate the user via the `alfresco/service/api/login?u=${username}&pw=${password}` URL, which is available out of the box from Alfresco.



Calling the `/alfresco/service/api/login` URL does not require any credentials to be set in the `httpClient` but we set them up so that they are available in future request.

If Alfresco successfully authenticates the user, then a ticket will be returned and this ticket will be used to check if the user is logged in or not:

```
def login(username, password) {
    def usercreds = new UsernamePasswordCredentials(username, password)
    httpClient.getState().setCredentials(AuthScope.ANY, usercreds)

    def url =
        "${config.mobilex.alfrescoApiUrl}/login?u=${username}&
        pw=${password}"
    def xmlString = executeGetRequest(url)
    def ticket

    try {
        ticket = new XmlSlurper().parseText(xmlString)
    } catch (org.xml.sax.SAXParseException spe) {
        log.error "Login error: ${spe}"
        httpClient.getState().clear()
    }
}
```



```
        ticket = ""
    }

    return ticket as String
}
```

The `executeGetRequest` method that makes the HTTP GET request will be implemented later as we said. A successful authentication returns the authentication ticket in the following XML format:

```
<ticket>TICKET_b196a552be25dfd0b66ab0be5240f47bdba59b03</ticket>
```

The `XmlSlurper` is used here to extract the ticket value and return it to the caller. If there is an error during the authentication then we just return an empty string.

The `logout` method is fairly simple as it just clears the state of the `httpClient` object:

```
def logout() {
    httpClient.getState().clear()
}
```

The user session will be invalidated in the controller as we will see later on.

Fetching child content for a folder via CMIS

Now let's move on and implement the methods that will be needed by the `FolderNavigationController` class. Only one method is needed and it should take a parent folder node reference and then return a map of content items contained in that folder:

```
def fetchChildrenForFolder(String nodeRef) {
    def nodeGuid = extractNodeGuid(nodeRef)
    def url = "${config.mobilex.childrenPath}/${nodeGuid}/children"
    def xmlString = executeGetRequest(url)
    return parseAtomFeed(xmlString)
}
```

Here, we again use the `executeGetRequest` method to make a call to a CMIS URL with the `/alfresco/service/cmis/s/workspace:SpacesStore/i/26cef395-d9c5-40f7-b1f4-6e4954462adb/children` format. When calling this CMIS URL, Alfresco will get all the content contained in the folder referenced in the URL and construct an **ATOMPub feed XML** document and send it back as a response. This response is then parsed by the `parseAtomFeed` method, which we will implement later. The method will return a list with map entries (property name → property value).

Searching the Alfresco repository via CMIS

To support the search feature, we will use the CMIS query URL `/alfresco/service/cmisis/queries`—it accepts `application/cmisisquery+xml` with the query statement and other search parameters. To query the repository, we POST data to this URL looking something like this:

```
<cmis:query xmlns:cmis='http://docs.oasis-open.org/ns/cmisis/core/200908/'>
  <cmis:statement>SELECT * FROM cmis:Document</cmis:statement>
  <cmis:searchAllVersions>false</cmis:searchAllVersions>
  <cmis:includeAllowableActions>false</cmis:includeAllowableActions>
  <cmis:includeRelationships>none</cmis:includeRelationships>
  <cmis:renditionFilter>*</cmis:renditionFilter>
  <cmis:maxItems>-1</cmis:maxItems>
  <cmis:pageSize>-1</cmis:pageSize>
  <cmis:skipCount>0</cmis:skipCount>
</cmis:query>
```

In our search scenario, users will be able to enter a keyword and we will search the content of documents and the name property of documents for matching the content. Here is how this is implemented:

```
def keywordSearch(String keyword) {
  def queryStatement = "SELECT cmis:ObjectTypeId, cmis:ObjectId,
    cmis:Name, cmis:ContentStreamLength, cmis:CreatedBy,
    cmis:LastModificationDate FROM cmis:Document WHERE
    CONTAINS('$keyword') OR cmis:name LIKE '%$keyword%'"
  def headers = ['Content-type': 'application/cmisisquery+xml',
    'Accept': 'application/atom+xml;type=feed']
  def queryWriter = new StringWriter()
  def queryXMLBuilder = new MarkupBuilder(queryWriter)
  queryXMLBuilder.'cmis:query'('xmlns:cmis':
    config.mobilex.cmisisNamespaces.cmisis) {
    'cmis:statement'(queryStatement)
    'cmis:searchAllVersions'(false)
    'cmis:includeAllowableActions'(false)
    'cmis:includeRelationships'('none')
    'cmis:renditionFilter'('*')
    'cmis:maxItems'(-1)
    'cmis:pageSize'(-1)
    'cmis:skipCount'(0)
  }
}
```

```
def xmlString = executePostRequest(config.mobilex.cmisQueryUrl,
    queryWriter.toString(), headers)
def list = []

if (!xmlString) {
    log.warn "No documents were found with keyword $keyword"
} else {
    list = parseAtomFeed(xmlString)
}

return list
}
```

Here again we use a very handy Groovy class called `MarkupBuilder`, which can be used to build the XML document that we are going to POST to the server. In our case, we use the following query statement:

```
SELECT cmis:ObjectTypeId,
    cmis:ObjectId,
    cmis:Name,
    cmis:ContentStreamLength,
    cmis:CreatedBy,
    cmis:LastModificationDate
FROM cmis:Document
WHERE CONTAINS('$keyword') OR cmis:name LIKE '%$keyword%'
```

We select only the properties that we will send back to the client. The query syntax is very much like a SQL query so it is easy to understand. The `CONTAINS` predicate is used to perform a full-text search in document content.



The `CONTAINS` predicate is an extension to the SQL 92 standard.

To execute the query we call the `executePostRequest` method, which will be implemented in a later section. It also takes a list of HTTP headers and we tell the server that we are going to send CMIS query XML (`Content-type: application/cmisquery+xml`) and that we accept ATOMPub feed (`Accept: application/atom+xml;type=feed`) back as a response.

The ATOMPub feed response is then parsed with the same `parseAtomFeed` method that was used in the `fetchChildrenForFolder` method.

Implementing the helper methods for the CMIS service

There are a couple of helper methods that we have used that need to be implemented. Let's start with the `executeGetRequest` method:

```
def executeGetRequest(url) {
  def method = new GetMethod(url)
  method.setDoAuthentication(true)
  def statusCode = httpClient.executeMethod(method)
  log.debug "Response status ${statusCode} from ${url}"
  def responseBody = method.getResponseBodyAsString()
  method.releaseConnection()
  return responseBody
}
```

This is pretty much the standard way of using the Apache HTTP client when making GET requests. Note that we call the `setDoAuthentication` method to do pre-emptive authentication, so that we do not get login dialogs when calling CMIS URLs. Implement the `executePostRequest` method in a similar way:

```
def executePostRequest(url, xmlData, headers) {
  def method = new PostMethod(url)
  method.setDoAuthentication(true)
  headers.each { k, v -> method.setRequestHeader(k, v) }
  method.setRequestBody(new StringBufferInputStream(xmlData))

  def responseBody
  try {
    def statusCode = httpClient.executeMethod(method)
    responseBody = method.getResponseBodyAsString()
  } catch (IOException ioe) {
    responseBody = ""
    throw new Exception("Error accessing CMIS", ioe)
  } finally {
    try {
      method.releaseConnection()
    } catch (Exception ex) {
      log.warn "Failed to release connection (possibly due to an
        earlier error): ${ex}"
    }
  }
  return responseBody
}
```

We start off by adding the extra HTTP headers to the `POST` method and then we set the CMIS query data. The response XML is returned without being processed as that will be done by the next method that we will implement, which is called `parseAtomFeed`. This method takes a parameter with the ATOMPub feed XML, which looks something like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:app="http://www.w3.org/2007/app"
      xmlns:cmisra="http://docs.oasis-open.org/ns/cmis/restatom/200908/"
      xmlns:cmis="http://docs.oasis-open.org/ns/cmis/core/200908/"
      xmlns:alf="http://www.alfresco.org"
      xmlns:opensearch="http://a9.com/-/spec/opensearch/1.1/">
  . . .
  <generator version="3.3.0 (2765)">Alfresco (Community)</generator>
  . . .
  <opensearch:totalResults>13</opensearch:totalResults>
  <opensearch:startIndex>0</opensearch:startIndex>
  <opensearch:itemsPerPage>-1</opensearch:itemsPerPage>
  <cmisra:numItems>13</cmisra:numItems>
  <entry>
    <author>
      <name>System</name>
    </author>
    <content>b2275521-005b-4580-ba5c-106ccdd389d9</content>
    <id>urn:uuid:b2275521-005b-4580-ba5c-106ccdd389d9</id>
    <link rel="self" . . .
    <link rel="edit" . . .
    . . .
    <link rel="up" . . .
    <published>2010-05-24T13:56:41.313+01:00</published>
    <summary>Site Collaboration Spaces</summary>
    <title>Sites</title>
    <updated>2010-05-24T13:56:41.332+01:00</updated>
    <app:edited>2010-05-24T13:56:41.332+01:00</app:edited>
    <alf:icon>http://localhost:8080/alfresco/images/icons/
      space-icon-default-16.gif</alf:icon>
    <cmisra:object>
      <cmis:properties>
        . . .
        <cmis:propertyString
          propertyDefinitionId="cmis:lastModifiedBy"
          displayName="Last Modified By"
          queryName="cmis:lastModifiedBy">
```

```

        <cmis:value>System</cmis:value>
      </cmis:propertyString>
      . . .
    <cmis:propertyString
      propertyDefinitionId="cmis:name"
      displayName="Name"   queryName="cmis:name">
      <cmis:value>Sites</cmis:value>
    </cmis:propertyString>
    . . .
  </cmis:properties>
</cmisra:object>
<cmisra:pathSegment>Sites</cmisra:pathSegment>
</entry>
. . .
</feed>

```

In the beginning, you have information about how many items were found (`opensearch:totalResults`) how many items were returned (`opensearch:totalResults`) and so on . Then there is one entry element per content item that was returned.

One of the first elements in the entry element is the content element:

```
<content>b2275521-005b-4580-ba5c-106ccdd389d9</content>
```

If it is not a folder, then this content entry contains information about the content-mime type and the URL to download the content, for example:

```

<content type="application/vnd.openxmlformats-
  officedocument.wordprocessingml.document"
  src="http://localhost:8080/alfresco/service/cmisa/s/
  workspace:SpacesStore/i/bbb487b2-3d70-4f8c-be67-e5d9300561fe/
  content.docx"/>

```

Then follow some links to be able to navigate from the folder node to, for example, the parent node (`link rel="up"`). Further down it contains an icon element (`alf:icon`), which we will use to get to the icon for the content item so that we can display it in the UI.

The `parseAtomFeed` method looks like this:

```

def parseAtomFeed(xmlString) {
  def feed
  try {
    feed = new XmlSlurper().parseText(xmlString).declareNamespace(
      config.mobilex.cmisaNamespaces)
  } catch (org.xml.sax.SAXParseException spe) {

```

```
// Workaround for 3.1.1 bug
def xmlString4 = xmlString.replaceAll("&(?!amp;)", "&amp;")
log.debug xmlString4
feed = new XmlSlurper().parseText(xmlString4).declareNamespace(
    config.mobilex.cmisNamespaces)
}

def list = feed.entry.collect {
    def objectId = getCmisProperty(it, 'cmis:objectId') as String
    def docName = getCmisProperty(it, 'cmis:name') as String
    def fileSize = getCmisProperty(it, 'cmis:contentStreamLength')
        as String
    if (fileSize) {
        fileSize = Integer.valueOf(fileSize)
    }
    def author = getCmisProperty(it, 'cmis:createdBy') as String
    def updatedAt = parseDate(getCmisProperty(it,
        'cmis:lastModificationDate') as String)
    def nsPrefixedType = getCmisProperty(it, 'cmis:objectTypeId')
        as String
    def type = ''
    if (nsPrefixedType) {
        // remove the xmlns prefix (e.g. cmis:)
        type = nsPrefixedType.substring(nsPrefixedType.
            lastIndexOf(':') + 1)
    }

    return [name: docName,
        type: type,
        contentId: objectId,
        iconUrl: it.'alf:icon' as String,
        documentUrl: it.content.@src,
        fileSize: fileSize,
        modifiedDate: updatedAt,
        author: author]
}

return list
}
```

This method starts off by parsing the passed-in ATOMPub feed XML to get a handle to the feed element. The feed variable is then used to loop through each entry element and extract the CMIS properties that we are after. We use a custom `getCmisProperty` method to get to a specific property, if we know that it is a date then we use the `parseDate` custom method.

At the end of the loop, we create a map with all the properties and add it to the list. This list is then returned to the caller.

The last couple of helper methods look like this:

```
def getCmisProperty(xmlRootObj, propName) {
    return xmlRootObj.'cmisra:object'.'cmis:properties'. '**'.grep
        { it.'cmis:propertyDefinitionId' == propName }. 'cmis:value'[0]
}

def parseDate(strDate) {
    if (strDate) {
        return new Date().parse("yyyy-MM-dd'T'HHmmss.SSSZ",
            strDate.replace(':', '').replace('Z', '+0000'))
    }
}

def extractNodeGuid(nodeRef) {
    return nodeRef?.substring((nodeRef?.lastIndexOf('/') ?: 0) + 1)
}
```

This completes the CMIS service, so we can start on the controllers.

Implementing UI controllers

Having built the CMIS service it is straightforward to implement the UI controllers. Let's start with the `AuthenticationController.groovy` class located in `alf_clients\mobilex\grails-app\controllers\com\bestmoney\mobilex\controllers`:

```
class AuthenticationController {
    def cmisService

    def showLoginPage = {
        render(view: 'authentication')
    }

    def authenticateUser = {
        flash.message = "Credentials set"

        try {
            def ticket = cmisService.login(params.j_username,
                params.j_password)
            session.alf_ticket = ticket
        } catch (ConnectException ce) {
```



```
        flash.message = "Cannot connect to CMIS repository."
    }

    redirect(uri: '/')
}

def logout = {
    cmisService.logout()
    session.invalidate()
    redirect(uri: '/')
}
}
```

The CMIS service is dependency injected via the `cmisService` variable declaration (this uses Spring autowiring by name). We then define the `showLoginPage` method that will display the login page by rendering the authentication view, which points to the `authentication.gsp` page. The `authenticateUser` method is called when the user has entered the username and password in the login page and clicks on the **Login** button. It uses the CMIS service to log in to Alfresco. If the login is successful then the Alfresco ticket (`alf_ticket`) is stored in the session.

The `logout` method will just log out via the CMIS service, invalidate the session, and redirect the user to the home page.

The next UI Controller is the `FolderNavigationController.groovy` class and the first part of it looks like this:

```
class FolderNavigationController {
    def cmisService
    def config = ConfigurationHolder.config

    def index = {
        redirect(action:list)
    }

    def list = {
        if (!params.parentFolderNodeRef) {
            def topLevelContentList = []
            def folderRootNode = cmisService.getRootFolderNodeRef()

            try {
                topLevelContentList =
                    cmisService.fetchChildrenForFolder(folderRootNode)
            } catch (Exception e) {
```

```

        log.error "Exception: ${e.message}"
        flash.message = "Could not access Alfresco via CMIS"
    }

    session.currentNode = folderRootNode
    session.folderNodeRefStack = new Stack()

    [contentList: topLevelContentList, parentFolderNodeRef: null]

```

The `list` method basically has two paths—one when the user is on the menu page and selects **Browse Folders** (which is the code part you see above) and a second one, when the user is already navigating around in the folder hierarchy and the current parent folder node reference is passed in as a parameter (`params.parentFolderNodeRef`) as shown:

```

    } else {
        def childContentList = []

        try {
            childContentList = cmisService.fetchChildrenForFolder(
                params.parentFolderNodeRef)
        } catch (Exception e) {
            log.warn "Exception accessing CMIS: ${e.message}"
            flash.message = "Could not access Alfresco via CMIS"
        }

        def parentFolderNodeRef = null
        def topFolderNodeRef = null

        if (!session.folderNodeRefStack.empty()) {
            topFolderNodeRef = session.folderNodeRefStack.peek()
        }

        if (topFolderNodeRef == params.parentFolderNodeRef) {
            session.folderNodeRefStack.pop()
            if (!session.folderNodeRefStack.empty()) {
                parentFolderNodeRef = session.folderNodeRefStack.peek()
            }
        } else {
            parentFolderNodeRef = session.currentNode
            session.folderNodeRefStack.push(parentFolderNodeRef)
        }

        session.currentNode = params.parentFolderNodeRef
    }

```

```
        [contentList: childContentList,
         parentFolderNodeRef: parentFolderNodeRef]
      }
    }
  }
```

If the parent node reference is not known, then we call the CMIS service with the root folder node reference to get the top folders:

```
cmisService.fetchChildrenForFolder(folderRootNode)
```

If a folder node reference is passed in, then we select the child content for that node reference:

```
cmisService.fetchChildrenForFolder(params.parentFolderNodeRef)
```

When we navigate around in the folder hierarchy we also create a folder node reference stack and store it in the session (`session.folderNodeRefStack = new Stack()`). We then use this stack to push a node reference on it when we navigate down into a folder and pop a node reference from it when we go up to the parent folder. In this way, we can manage when the user is using the **Back** button in the browser and not the buttons and link in the application UI.

The child content list (that is, `contentList`) is returned to the `list.gsp` page that will display it.

The last controller that we need to implement is the `SearchController.groovy` class:

```
class SearchController {
  def cmisService

  def index = {
    render(view: 'search')
  }
  def search = {
    if (!params.keyword?.trim()) {
      return [:]
    }

    def resultList
    try {
      resultList = cmisService.keywordSearch(params.keyword)
    } catch (Exception e) {
      log.error "Exception: ${e.message}"
      flash.message = "Could not access Alfresco via CMIS"
    }
  }
}
```

```

        return [searchResult: resultList]
    }
}

```

Once the user has entered a keyword and clicked on the **Search** button, we execute the search via the CMIS service. The search result is returned to the `search.gsp` page that displays it.

Implementing the Groovy Server Pages (GSP)

The only parts left to complete the **Model-View-Controller (MVC)** pattern are the views that are implemented as **Groovy Server Pages (GSP)**. The `authentication.gsp` page looks like this:

```

<html>
<head>
    <title>MobileX - Login</title>
    <meta name="layout" content="main"/>
</head>
<body>
<div class="body">
    <h1>Login</h1>
    <g:if test="{flash.message}">
        <div class="message">${flash.message}</div>
    </g:if>
    <g:form action='authenticateUser' method='post'>
        <p>
            <label for='j_username'>Username:</label>
            <input type='text' name='j_username' id='j_username' />
        </p>
        <p>
            <label for='j_password'>Password:</label>
            <input type='password' name='j_password' id='j_password' />
        </p>
        <p>
            <input type='submit' value='Login' />
        </p>
    </g:form>
</div>
</body>
</html>

```

It will display a username field, a password field, and a **Login** button. When the **Login** button is clicked, the form action `authenticateUser` is kicked off; this calls a method with the same name in the `AuthenticationController.groovy`.

After a successful login, the user is redirected to the main menu defined in the `index.gsp` page that looks like this:

```
<html>
<head>
  <title>Welcome to MobileX</title>
  <meta name="layout" content="main"/>
</head>
<body>
  <g:if test="${flash.message}">
    <div class="message">${flash.message}</div>
  </g:if>
  <div class="dialog" style="margin-left:20px;width:60%;">
    <ul>
      <li><g:link controller="folderNavigation">
        Browse Folders</g:link></li>
      <li><g:link controller="search">
        Search for Documents</g:link></li>
      <li><g:link controller="authentication"
        action="logout">Logout</g:link></li>
    </ul>
  </div>
</body>
</html>
```

The menu has links so the users can browse folders, search, or log out if they want to.

The folder navigation page is defined in the `list.gsp` page and the first part that creates the top menu/toolbar looks like this:

```
<html>
<head>
  <title>MobileX - Folder Browsing</title>
  <meta name="layout" content="main"/>
</head>
<body>

<div class="nav">
  <span class="menuButton">
    <a class="home" href="${resource(dir: '',
      file: 'index.gsp')}">Home</a>
  </span>
```

```

<span class="menuButton">
  <g:link class="list" controller="search">Search</g:link>
</span>
<g:if test="${parentFolderNodeRef}">
  <span class="menuButton">
    <g:link class="edit" action="list"
      params="[parentFolderNodeRef:parentFolderNodeRef]">Up
    </g:link>
  </span>
</g:if>
</div>

```

Clicking on the **Home** button takes you to `index.gsp` with the main menu and if we click on the **Search** button, the default action method in the `SearchController` will be invoked so that the `search.gsp` page is displayed. The **Up** link is displayed only if the user has navigated one level down in the folder hierarchy.

The body of this page will list the child folders and documents of the current parent folder node reference:

```

<div class="body">
  <div class="list">
    <table>
      <thead>
        <tr>
          <th>&nbsp;</th>
          <th>Name</th>
          <th>Size</th>
          <th>Owner</th>
          <th>Modified</th>
        </tr>
      </thead>
      <tbody>
        <g:each in="${contentList}" status="i" var="contentItem">
          <tr class="${(i % 2) == 0 ? 'odd' : 'even'}">
            <td style=
              "vertical-align:middle;horizontal-align:center;">
              
            </td>
            <td>
              <g:if test="${contentItem.type == 'document'}">
                <a href="${contentItem.documentUrl}?
                  alf_ticket=${session.alf_ticket}"
                  target="_blank">${contentItem.name}</a>
              </g:if>
            </td>
          </tr>
        </g:each>
      </tbody>
    </table>
  </div>
</div>

```

```
        </g:if>
        <g:else>
            <i><g:link action="list"
                params=" [parentFolderNodeRef:contentItem.contentId] ">
                ${contentItem.name}</g:link></i>
        </g:else>
    </td>
    <td>${contentItem.fileSize}</td>
    <td>${contentItem.author}</td>
    <td><g:formatDate date="${contentItem.modifiedDate}"
        type="datetime" style="SHORT"/>
    </td>
</tr>
</g:each>
</tbody>
</table>
</div>
</div>
</body>
</html>
```

Worth noting here is that we need to add the `alf_ticket` parameter stored in the session for every document download link, as for this implementation the URL will not be targeting the MobileX application but Alfresco directly. So there are no `httpClient` user credentials that can be automatically used for the request.

It is possible to build a CMIS-based application that is totally independent of the vendor; however, this would require proxying the content.

Finally, the `search.gsp` looks very much like the `list.gsp` when it displays the search result, so you can resort to the source code that comes with the book for the code for this page.

Implementing an authentication filter

Whenever the user is not authenticated he or she should be redirected to the login page no matter which URL is being called. This is best implemented with a "before" filter that can be used across a whole group of controllers. We already generated a filter in the beginning so we just need to implement it. Open up the `AuthenticationFilter.groovy` class located in the `grails-app/conf/mobilex` folder and change it to look like this:

```
class AuthenticationFilters {
    def filters = {
        loginCheck(controller: '*', action: '*') {
```

```

before = {
  if ((session == null || !session?.alf_ticket) &&
      !controllerName.equals('authentication')) {
    redirect(controller: 'authentication',
              action: 'showLoginPage')
    return false
  }
}
}
}
}
}

```

We basically just check the `alf_ticket` session variable and if it is not there then we send the user to the login page. This is mapped to all controllers and all actions though doesn't apply the logic to the authentication controller to prevent an infinite loop.

Running the mobile application

We are now done with the coding and we can try out the application by running the following Grails command:

```
3340_14_Code\bestmoney\alf_clients\mobilex> grails -Dserver.port=8081
run-app
```

Access the application from any browser with `http://<hostname>:8081/mobilex`.

Content creation with MobileX

This is beyond the scope of this chapter, but it would be quite easy to extend MobileX to add functionality to meet user stories such as "As a mobile user, I want to store attachments received by e-mail". This would involve operations such as creating folders, creating documents, and so on, which are all part of the CMIS specification.

Using the Apache chemistry API

There is also the Apache Chemistry OpenCMIS subproject that could be used instead of implementing the MobileX CMIS service from scratch. However, there are things like getting to the content icon that are not possible via that API and it is also useful to understand the inner workings of the CMIS interface.

It might be worth checking out Apache Chemistry (<http://incubator.apache.org/chemistry>) because it evolves all the time.

Summary

In this chapter, we have seen two solutions to how we can access content in the Alfresco repository via a mobile device such as an iPhone or a BlackBerry. The `mobile.war` web application that comes with the Alfresco package can be used to get to site content created via Alfresco Share. However, if we need to get to all content in the repository then we need to come up with a custom solution such as the MobileX application that we developed in this chapter.

In this chapter, we have learned how to use the mobile application that comes with Alfresco out of the box and how it can be used to access content in Alfresco Share-created sites from iPhones.

We saw how a web application framework like Grails that is built on top of Groovy, can be very productive when creating a custom web application like MobileX, targeting mobile device users with, for example, BlackBerry smartphones.

When developing the MobileX application, we explored CMIS and discovered that it is a good interface to use from Groovy as there are classes like `MarkupBuilder` and `XmlSlurper` that really make it easy to handle CMIS requests and responses.

CMIS requests for child content are just standard `GET` requests and the response is coded in ATOMPub Feed XML format. Searching via CMIS uses a `POST` where we can specify the search query in an SQL-like language. CMIS does not specify how authentication with the content server should be done and it is up to us to solve it via the transport layer.

Index

Symbols

<xs:sequence> element 288
_alfresco/config directory 110, 111
_alfresco/source directory 114
_share/config directory
 about 115, 116
 subfolders 116
_share directory 110
3G 527
4G 527

A

Acegi Security 24
ACP 110, 280
ACP file import
 about 357
 advantages 357
 disadvantages 357
ACP Generator tool 364, 365
Active Directory. *See* AD
AD
 about 159, 160
 authentication, configuring with 161, 162
 synchronization, configuring with 161, 162
Advanced Search
 properties, displaying in 328, 342, 343
advanced versioning management (AVM)
 system 264
Advanced Versioning Manager Store. *See*
 AVM Store
afterCreateVersion event 32
ALF_APPLIED_PATCH table 53
alf_client directory 110

alf_data directory
 about 46
 structure 46
alf_extensions directory 110
ALF_NODE_ASPECTS table 52
ALF_NODE_PROPERTIES table 52
ALF_NODE table 51
ALF_QNAME table 53
alf_ticket parameter 558
alf_ticket session variable 559
Alfresco
 about 8, 57
 accessing, via KDC TGS 145, 146
 application zones 152
 authentication subsystems 150, 151
 authentication zones 151
 bootstrap 41
 CMIS service, implementing 540
 components 25
 content rules 26, 27
 content transformers 36, 37
 custom mobile application solution, for
 smartphones 533
 directory structure 46
 event model 27
 extension modules 42
 folder hierarchy 225, 226
 Grails MobileX application, configuring
 539
 Grails MobileX application, creating 538
 groups 20, 21
 metadata extraction 35
 metadata extractors 35, 36
 Meta Model XML schema 287

- MT feature 23
- open source projects 24
- services 25
- store reference 15
- stores 12
- subsystems 40
- synchronization subsystems 150, 151
- third-party extensions 42
- UI controllers, implementing 551-554
- user interfaces clients 43, 45
- users, authenticating with 543, 544
- alfresco-global.properties file 153**
- alfresco.war AMP Extension 118**
- alfresco.war file 104, 105**
- alfresco/module**
 - about 112, 113
 - directories 113, 114
 - files 113, 114
- ALFRESCO_ADMINISTRATORS group 21**
- Alfresco boxes**
 - folder hierarchies, copying between 280, 281
- Alfresco Bulk Filesystem Import 42**
- Alfresco bulk import tool**
 - about 360
 - Import directory field 360
 - Target space field 360
 - Update existing files checkbox 361
- Alfresco CIFS**
 - about 45, 177
 - troubleshooting 215
 - troubleshooting strategies 217
- Alfresco CIFS server**
 - about 191
 - configurations 191
 - configuring 194
 - on Linux 193, 194
 - on Windows 192, 193
- Alfresco CIFS server configuration**
 - Alfresco File Server subsystem 195
 - Linux Server and Windows 7 client configuration 208-210
 - Windows 2003 Server and Windows 7 client configuration 200-202
 - Windows 2008 Server, Active Directory, and Windows 7 client configuration 202-207
 - Windows Vista Server, Windows 7, and XP clients configuration 195-200
- Alfresco Content Package (ACP) files 357**
- Alfresco Content Packages. *See* ACP**
- Alfresco database 50**
- Alfresco database schema 51**
- Alfresco data web script**
 - about 484
 - advantages 484
 - disadvantages 484
- Alfresco Explorer**
 - about 43, 105-107, 326
 - aspects, displaying in rules wizards 328
 - properties, displaying in Advanced Search 328
 - properties, displaying in content details pages 326, 327
 - property sheets, registering 329
 - resource file, registering 329
 - types, displaying in add content wizards 327
 - types, displaying in create content wizards 327
 - types, displaying in rules wizards 328
- Alfresco Extension projects**
 - about 105
 - Alfresco Explorer 105-107
 - Alfresco Share UI extensions 107
 - repository extensions 105-107
- Alfresco Forum Model**
 - type definition examples 301-304
- Alfresco JavaScript**
 - URL 390
- Alfresco Knowledgebase 529**
- Alfresco Management Beans (JMX) 39**
- Alfresco meta model**
 - and Alfresco content models, differences 286
 - and custom content models, differences 286
- Alfresco Mobile 45**
- Alfresco mobile web application**
 - accessing 528-532
 - installing 528
- Alfresco Module Management Tool 105**
- Alfresco Module Package (AMP) 59, 105**

- Alfresco namespaces**
 - bmc namespace 396
 - bmw namespace 395
 - bpm namespace 395
 - cm namespace 395
 - d namespace 395
- Alfresco Node Browser** 47
- alfrescoNtml subsystem** 151
- Alfresco platform**
 - about 7, 8, 23
 - layered approach 9, 10
 - overview 9, 10
- Alfresco portlet implementation approaches**
 - Alfresco data web script 484
 - Alfresco presentation web script 484
 - CMIS web script 484
- Alfresco presentation web script**
 - about 484
 - advantages 484
 - disadvantages 484
- Alfresco records management**
 - about 282, 283
 - File Plan, managing 283
- Alfresco repository**
 - Document Folder Template 229
 - overview 11, 12
 - searching, with CIMS 545, 546
- Alfresco Share**
 - about 43, 330, 530
 - aspects, displaying 341, 342
 - debugging 123
 - properties, displaying in Advanced Search 342, 343
 - properties, displaying in metadata pages 330-340
 - types, displaying 341, 342
 - URL 419
 - using 533
- Alfresco SharePoint** 43, 44
- Alfresco Share UI extensions**
 - about 107
 - META-INF directory 108
 - site-data directory 108
 - site-webscripts directory 108
- Alfresco source code**
 - downloading 50
- Alfresco Spring Surf JAR file**
 - directory structure 107
- Alfresco user interfaces**
 - folders, setting up 250, 251
- Alfresco WCM**
 - URL 22
- Alfresco WebDAV**
 - about 177, 210
 - accessing 210, 211
 - WebDAV clients 212
 - Windows built-in WebDAV clients 214
- Alfresco workflow feature**
 - diagrammatic representation 384
- AMP extensions**
 - debugging 123
 - deploying 119-122
- amps directories** 48
- Anovio Email Management solution**
 - about 510
 - URL 510
- Ant targets**
 - for alfresco.war AMP Extension 118, 119
 - for share.war JAR extension 120
- Apache Abdera** 24, 95
- Apache Ant build file** 118
- Apache Axis** 24
- Apache Chemistry** 24
- Apache chemistry API**
 - using 559
- Apache Commons** 24
- Apache CXF** 24
- Apache Directory Studio** 158
- Apache HTTP Commons** 490
- Apache PDFBox** 24
- Apache POI** 24
- Application Module Package (AMP)** 39
- Application Programming Interfaces (APIs)**
 - about 8, 39, 57, 59
 - Client-server APIs 94
 - In-process APIs 59
- Apply Best Money Document Type rule**
 - defining 260-262
- Apply Versioning rule**
 - defining 263, 264
- Archive Store** 12
- AS** 143

- aspect events**
 - about 30
 - beforeAddAspect 30
 - beforeRemoveAspect 30
 - onAddAspect 30
 - onRemoveAspect 30
- aspects**
 - about 16, 285
 - displaying 341, 342
 - displaying, in rules wizards 328
 - uses 309
- association events 31, 32**
- associations 285**
- associations definition**
 - about 298
 - association 298
 - child-association 298
 - description 299
 - name 299
 - source 299
 - target 299
 - title 299
- asynchronous action execution event**
 - about 29
 - onAsyncActionExecute 29
- ATOMPub feed XML document 544**
- audit.contentstore directory 48**
- authenticateUser method 552**
- authentication**
 - about 543
 - configuring, against Microsoft Active Directory 159
 - configuring, against OpenLDAP 153
 - configuring, with Active Directory 161, 162
 - making, more secure 172
 - setting up, with Remote Directory servers 152
- authentication.gsp page 555**
- authentication and security, CIFS 189**
- authentication concepts 138**
- AuthenticationController 543**
- authentication filter**
 - implementing 558
- AuthenticationFilter.groovy class 558**
- Authentication Server. *See* AS**
- authentication subsystem**
 - configuring 169, 170

- authentication zones 151**
- AVM Store**
 - about 14
 - versus Working Store 15

B

- backup-lucene-indexes directory 48**
- base64 encoded 138**
- BeanShell script**
 - about 390
 - URL 390
- beforeAddAspect event 30**
- beforeCopy event 29**
- beforeCreateNode event 30**
- beforeCreateReference event 32**
- beforeCreateStore event 29**
- beforeCreateVersion event 32**
- beforeDeleteNode event 30**
- beforeRemoveAspect event 30**
- beforeRemoveReference event 32**
- beforeRMActionExecution event 32**
- beforeUpdateNode event 30**
- BES 537**
- Best Money 8, 527**
- Best Money document management solution**
 - designing 236
 - implementing 246
- Best Money document management solution, designing**
 - about 236
 - meeting folder 242-245
 - meetings folder hierarchy 236-239
 - press folder hierarchy 236-242
 - space templates 242-245
- Best Money document management solution, implementing**
 - about 246
 - details list view for folder, configuring 271
 - document review periods, setting up 272
 - file display, configuring 271
 - folder hierarchy, setting up 249
 - Google Like search, configuring 271
 - groups, setting up 246-249
 - space templates, setting up 269, 270
 - users, setting up 246-249

- Best Money portal architecture** 484
- Binary Large Objects.** *See* BLOBs
- BitKinex**
 - about 212
 - download link 212
- BlackBerry Bold 9700 Simulator** 535
- BlackBerry Enterprise Server.** *See* BES
- BLOBs** 12
- bmc:job** aspect 399
- bmc namespace** 396
- bmw:assigneeCopywriter** association 436
- bmw:conceptWorkDueDate** property 436
- bmw:copywriteWorkDueDate** property 436
- bmw:signOffPhase** property 449
- bmw:baseAssignJobTask** job task 399
- bmw:baseWorkTask** task type 399
- bmw:workType** work type 399
- bmw_workTypes** collection 440
- bmw_workType** variable 404
- bmw namespace** 395
- bootstrap, Alfresco**
 - about 41
 - importer component 41, 42
 - patches 41
- Bootstrap APIs**
 - about 98
 - importer 99, 100
 - patches 98, 99
- bootstrapping** 41
- bpm:assignee** association 436
- bpm:dueDate** property 436
- bpm:assignee** property 399
- bpm:packageActionGroup** property 399
- bpm_assignee** variable 389
- bpm_package** 462
- bpm namespace** 395
- build server, downloading**
 - requisites 125
- business processes**
 - designing, with Swimlane diagram 370, 372
- business rules**
 - linking to 268
 - setting up, for folders 258, 259
- business rules, for folders**
 - Apply Best Money Document Type rule, defining 260-262

- Apply Versioning rule 263, 264
- Check Naming Convention rule 264, 265
- Extract Meeting Filename Metadata rule 266, 267

C

- calculateVersionLabel** event 32
- Check Naming Convention rule**
 - defining 264, 265
- Chiba** 24
- child-association definition**
 - about 300
 - child-name 300
 - duplicate 300
 - propagateTimestamps 300
- child content, for folder**
 - fetching, via CMIS 544
- CI environment**
 - about 124
 - setting up 125-129
- CIFS**
 - about 177, 178
 - connecting to 179
 - features 178
 - folders, setting up 250
 - working 179
- CIFS-based document migration**
 - diagrammatic representation 354, 355
- CIFS authentication**
 - about 139
 - custom authenticator, implementing for 165
- CIFS dialect negotiation** 187, 188
- CIFS features**
 - access 178
 - data integrity 178
 - optimization for slow links 178
 - security and granularity 178
 - unicode file names 178
- CIFS import**
 - about 353, 354
 - advantages 355
 - disadvantages 355
- CIFS interface**
 - used, for accessing 163, 164

Citrix XenApp 210

classes
 deploying, for custom authenticator 171

client-debug-autologging flag 124

client-debug flag 123

Client-server APIs
 about 94
 CMIS API 94, 95
 Custom APIs 95-97
 Repository API 95

CMIS
 about 534
 Alfresco repository, searching with 545, 546
 child content folder, fetching with 544

CMIS API
 about 58, 94
 using 95

CMIS service
 about 552
 helper methods, implementing for 547-551
 implementing 540

CMIS web script
 about 484
 advantages 484
 disadvantages 484

cm namespace 395

com.bestmoney.cms.MyCmsService interface 61

Command Servlet API 58

Common Internet File System. *See* CIFS

communication services, NBT
 Datagram service 185
 datagram service 180
 Session service 183
 session service 180

completionDate property 398

complex transformers 38

components, Alfresco 25

composite type, content model design pattern
 about 311
 example 312, 313
 issues 311, 312
 solution 312

conditions 430

configuration, Java Foundation Services API 59-62

configuration object, content model design pattern
 about 315
 code example 316-319
 definition example 316
 issues 315
 solution 316

constraint definition. *See* **model.constraints, Meta Model XML schema**

CONTAINS predicate 546

content-model.xml file 399

content details pages
 properties, displaying in 326, 327

content element 549

content files, storing in filesystem
 reasons 13

Content Management Interoperability Services API. *See* **CMIS API**

Content Management Systems (CMS) 7

content model
 about 285
 and Alfresco meta model, differences 286
 and custom content models, differences 286
 custom 320
 design patterns 310

content model, design patterns
 about 310
 composite type 311-313
 configuration object 315-319
 domain document root type 310, 311
 multiple types inheritance 314, 315

ContentModel objects 491

content rules, Alfresco
 about 26
 setting up 26, 27

content service 62

content service events
 about 28
 onContentPropertyUpdate 28
 onContentRead 28
 onContentUpdate 28

Content Store
 about 12
 policies 13, 14

- contentstore.deleted directory** 47
- contentstore directory**
 - about 47
 - file versioning 47
- Content Store Selectors** 14
- Content Transformation API** 58 86, 88
- content transformers, Alfresco**
 - about 36
 - Any text to plain text 37
 - benefits 37
 - E-mail (MIME) to text 37
 - E-mail (Outlook) to text 37
 - Excel to plain text 37
 - HTML to plain text 37
 - Image to image 37
 - MediaWiki markup to HTML 37
 - Open Office to image 37
 - Open Office to PDF 37
 - PDF to image 37
 - PDF to plain text 37
 - Plain Text to image 38
 - Plain Text to PDF 38
 - Word to plain text 37
- Continuous Integration.** *See* **CI environment**
- copy service events**
 - about 28
 - beforeCopy 29
 - getCopyCallback 28
 - onCopyComplete 29
- custom-slideshow-application-context.xml** 117
- Custom APIs** 58, 95
- custom authenticator**
 - implementing, for CIFS authentication 165
- custom content model**
 - about 320
 - and Alfresco content model, differences 286
 - and Alfresco meta model, differences 286
 - new model, defining 320-324
 - property sheets, configuring for UI display 326
 - registering, with repository 325
- customized task dashlets**
 - using 471
- CustomLinkGenerator** 477

- custom mobile application solution**
 - for smartphones 533
- custom NTLM authenticator**
 - building 166-169

D

- Dashlet components** 117
- data-types definition.** *See* **model.data-types, Meta Model XML schema**
- database schema, Alfresco** 51
- Data Dictionary top folder**
 - subfolders 226, 227
- Data Web Scripts** 96
- DEBUG logging** 76
- decide method** 430
- DecisionHandler** 430
- decision node** 428, 430
- DELETE permission group** 75
- description property** 398
- development environment**
 - directory servers, using in 174
 - setting up 105
- development environment, setting up**
 - about 105
 - Alfresco Extension projects 105
 - Apache Ant script, creating 118
 - extensions, deploying 118
 - project directory structure 109
- Dictionary Service** 75
- directory servers**
 - using, in development environment 174
- directory service, installing**
 - reasons 175
- directory structure, Alfresco**
 - about 46
 - alf_data directory 46
 - amps directories 48
 - tomcat directory 49
- dist directory** 121
- DM (Document Management)** 12
- DM repository** 529
- d namespace** 302, 395
- Document Folder Template, Alfresco repository**
 - about 229
 - document versioning 235

- folder name 229, 230
 - folder permissions 231, 232
 - folder title 231
 - metadata 233-235
 - processes 235
 - rules 232
 - document management solutions**
 - about 223
 - designing 228
 - features 223, 224
 - document management solutions, features**
 - 3rd party application integration 224
 - about 223
 - auditing 223
 - automatic transformations 224
 - business rules, applying 224
 - classification 223
 - extended search 223
 - fine grain permissions 224
 - process automation 224
 - versioning 223
 - document migration**
 - common steps 358
 - implementing 359
 - planning 358
 - document migration, implementing**
 - about 359
 - ACP Generator tool, using 364-367
 - Alfresco bulk filesystem import tool, using 359
 - Alfresco bulk import tool, running 360-362
 - extra metadata, applying 362-364
 - document migration staging area**
 - setting up 346
 - document migration strategies**
 - about 346
 - documents, importing via ACP file 357
 - documents, importing via CIFS 353
 - documents, importing via external tool 355
 - general migration strategies 346
 - document review periods, setting up**
 - about 272
 - reviewable aspect, adding 272, 274
 - review folder content script, running 278-280
 - for folder 274, 275
 - scripts, for folder review periods check 275, 277
 - document search feature**
 - for MobileX application 537, 538
 - document versioning 235**
 - domain**
 - new custom type, defining for 308
 - domain document root type, content model design pattern**
 - about 310
 - example 311
 - issues 310
 - solution 310
 - Do not Repeat Yourself. *See* DRY principle**
 - doView method 490**
 - DRY principle 373**
 - dueDate property 398**
 - DWG 36**
 - dynamic descriptions**
 - using, in Work process 404, 405
- ## E
- E-MAIL_CONTRIBUTORS group 21**
 - e-mail client talking to Alfresco, through custom built plugin and Web Scripts**
 - about 510
 - disadvantages 511
 - overview 510, 511
 - e-mail client talking to Alfresco, through custom module and Web Scripts**
 - about 512
 - advantages 513
 - disadvantages 513
 - overview 512
 - e-mail client talking to Alfresco, via IMAP protocol**
 - about 508
 - advantages 509
 - disadvantages 509, 510
 - working 508
 - e-mail integration solutions**
 - about 508
 - e-mail client talking to Alfresco through custom built plugin and Web Scripts 510, 511

- e-mail client talking to Alfresco through
 - custom module and Web Scripts 512, 513
 - e-mail client talking to Alfresco via IMAP
 - protocol 508, 509
 - e-mail management solutions**
 - implementing 514
 - implementing, with Alfresco IMAP 514
 - e-mail management solutions implementation**
 - about 514
 - document metadata, viewing, from e-mail client 520, 521
 - e-mail, drag-and-drop in Alfresco folder 517, 518
 - e-mail, viewing from Alfresco Explorer 518, 519
 - e-mail attachment extraction 520
 - e-mails, dragging-and-dropping in Alfresco Share site 522
 - IMAP account, setting up in Outlook 2007 515-517
 - IMAP server, enabling 514
 - Mount Points, using 523, 524
 - e-mail notification**
 - adding 470
 - e-mail template, sign-off process**
 - bootstrapping 449, 451
 - EAI solutions 479**
 - Eclipse 428**
 - Eclipse JBoss jPDL plug-in 390**
 - ECM solution 223**
 - EHCache 24**
 - Enterprise Application Integration. *See* EAI solutions**
 - Enterprise Reporting AMP 42**
 - entry element 549**
 - Event API 57**
 - event handler**
 - creating 78, 79
 - registering 79, 80
 - registering, requisites 33
 - working 34
 - Event management API**
 - about 77
 - event handler, creating 78, 79
 - event model, Alfresco 27**
 - EVERY_EVENT 33**
 - EVERYONE group 21**
 - Excel file**
 - folders, importing from 253-256
 - Excel spreadsheet**
 - task summary list, exporting in 475, 476
 - executeGetRequest method 542, 544, 547**
 - executionContext, script variables 394**
 - export command 100**
 - extension directory 111, 112**
 - extension modules, Alfresco 42**
 - extensions**
 - debugging 122
 - deploying, build file used 120
 - extensions, merging into alfresco.war**
 - reasons 105, 106
 - ExternalSignOffNotification action 464**
 - Extract Meeting Filename Metadata rule**
 - defining 266, 267
- ## F
- failover transformers 38**
 - fdk-config-custom.xml 117**
 - fetchChildrenForFolder method 546**
 - Fiber Channel (FC) 13**
 - file-mapping.properties file 107**
 - file access concepts**
 - about 178
 - CIFS protocol 178
 - CIFS Transport 179
 - File Folder service**
 - about 67
 - file, renaming with 68
 - nodes, copying 69
 - nodes, moving 69
 - using 67-69
 - FileInfo object 67**
 - File Plan**
 - managing 283
 - FIRST_EVENT 33**
 - folder and document browsing feature**
 - for MobileX application 536, 537
 - folder hierarchies**
 - copying, between Alfresco boxes 280, 281
 - folder hierarchy, Alfresco**
 - about 225

- Data Dictionary 225, 226
- Guest Home 225
- Sites 225
- User Homes 225
- Web Deployed 226
- Web Projects 226
- folder icons**
 - updating 251
- folder link**
 - adding, to property sheets 476, 477
- folder name 229, 230**
- FolderNavigationController class 544**
- folder permissions**
 - about 231, 232
 - setting up 257, 258
- folders**
 - business rules, setting up for 258, 259
 - cleaning up 256
 - importing, from Excel file 253-256
 - setting up, Alfresco user interfaces used 250, 251
 - setting up, CIFS used 250
 - setting up, scripts used 251
- folder title 231**
- foreach loop 429**
- Fork node 428**
- Foundation Services API 26, 57**
- FreeMarker 24**
- FreeMarker template 108, 488**
- full text search (FTS) 223**
- functions, MMT**
 - about 106
 - backup of changes 106
 - module listing 106
 - module updates 106
 - module versioning 106

G

- general migration strategies**
 - about 346
 - document migration staging area, setting up 346
 - Modified Date, preserving 346, 347
 - post migration processing scripts 348
- general steps, document migration 358**

- Generic Security Services Application Program Interface. *See* GSS-API**
- GenericWorkConcept node 431**
- getAlfrescoTicket function 498**
- getCmisProperty method 550**
- getCopyCallback event 28**
- Google Like search**
 - configuring 271
- Grails framework**
 - URL 533
- Grails MobileX application**
 - configuring 539
 - creating 538
- Greenmail 24**
- Groovy 533**
- Groovy Server Pages. *See* GSP**
- group**
 - folder hierarchies, copying between 281
 - synchronizing, with OpenLDAP 155, 156, 157, 158
- group imports**
 - customizing 163
- groups, Alfresco**
 - about 20
 - ALFRESCO_ADMINISTRATORS 21
 - E-MAIL_CONTRIBUTORS 21
 - EVERYONE 21
- GSP**
 - implementing 555-558
- GSSAPI 144**
- GWT/GXT-based 497-499, 503-506**
- GWT module class**
 - creating 499-501
- GXT 485**

H

- helper methods**
 - implementing, for CIMS service 547-551
- Hibernate 24**
- higher level permission groups 22**
- href attribute 542**
- HTML 36**
- HTTP Basic authentication**
 - about 138, 139
 - advantages 138

httpClient object 542
HTTP GET call 492
Hudson 124

I

iBATIS 24
IDC
 URL 527
identifier 15, 17
import definition. *See* **model.imports, Meta Model XML schema**
importer component 41, 99, 100
ImporterModuleComponent class 99
In-process APIs
 about 59
 Content Transformation API 86, 88
 Event management API 77-81
 Java Foundation Services API 59
 JavaScript API 88-92
 Metadata Extraction API 81-85
inbound variables 432, 433
index.recovery.mode property 48
index definition
 about 297
 atomic 297
 enabled 297
 stored 298
 tokenised 298
initiator swimlane 399
init method 481
installation, directory service
 reasons 175
iPhone 528
Is Material OK? (If Applicable) decision 372
Is Material OK? decision 372

J

Jackson JSON parser 490
Java-based 490-497
Java Community Process (JCP) 480
java directory
 about 115
 subfolders 115
Java Enterprise Edition (JEE) servers 23

Java Foundation Services API
 about 59
 configuring 59-62
 content services, using 62-66
 Dictionary Service 75
 File Folder service 67-69
 node services, using 62-66
 Permission Service 73, 74
 Search Service 70-73
 transaction management 59-62

Java Mail 24

JavaScript API
 about 57, 58, 88
 debug logging 94
 event handlers 93
 info site 93
 using 89-92

javascript directory 115

JavaScript event handlers 93

javax.portlet.Portlet interface 482

JBoss 8

JBOSS jBPM 24

jBPM Process Definition Language. *See* **jPDL**

jBPM Process Designer
 downloading 385
 URL 385

JBPM workflow definitions 114

jBPM workflow engine
 about 384
 implementing 385
 Work subprocess, implementing 385

JCR API 58

JGroups 24

JLAN technology
 about 191
 features 191

job data, Work process
 defining 405, 406

jobOwner swimlane 388, 399

Job process
 about 378, 379
 implementing 455
 property file 468
 property sheets 468
 testing 469

- workflow content model 467
- workflow definition (jPDL) 455-466
- jobStatus property** 406
- Join node** 428
- jPDL** 371, 372
- jPDL Schema** 389
- jPDL syntax** 389
- jPDL XML Schema version** 429
- JSR168** 480
- JSR286** 480
- JSR301** 480
- JSR329** 480

K

- KDC** 143
- KDC AS**
 - service login, performing 144, 145
 - user login, performing 144, 145
- KDC TGS**
 - Alfresco, accessing 145, 146
- Kerberos**
 - about 142
 - characteristics 143
- Kerberos authentication** 143, 144
- Key Distribution Center.** *See* KDC

L

- label property files, Work process**
 - bootstrapping 411, 412
- layered approach, Alfresco platform** 9, 10
- LDAP**
 - about 147
 - synchronizing 150
- ldap.authentication.allowGuestLogin prop-**
erty 153
- ldap.authentication.defaultAdministra-**
torUserNames property 154
- ldap.authentication.java.naming.provider.**
url property 154
- ldap.authentication.userNameFormat prop-**
erty 154
- LDAP authentication** 148
- ldapsearch command** 149
- LDAP structure**
 - sambaSamAccount, adding to 165, 166
- LDAP synchronization** 150

- LDAPv3** 149
- Liferay portal** 479
- Liferay portlet deployment descriptor**
 - creating 495, 496, 504
- list method** 553
- logging object**
 - setting up 76, 77
- login method** 543
- logout method** 544, 552
- low-level permissions** 22
- Lucene** 24
- lucene-indexes directory** 48
- LUCENE query language** 70

M

- management dashlets** 471-474
- mapper class** 168
- Marketing-Admins group** 472
- Marketing-All Users** 472
- marketing production workflow**
 - implementing 427
- MarkupBuilder class** 546
- mashups** 479
- material production process**
 - designing 378
- material production process, designing**
 - about 378
 - job process Swimlane diagram 378, 379
 - sign-off process Swimlane diagram 379, 380
 - studio process Swimlane diagram 380, 381
 - work process Swimlane diagram 381
- MD4**
 - hash 168
 - message digest 139
- MD4 passwords**
 - generating 166
- Meetings folder hierarchy** 236-239
- META-INF directory** 108, 114
- metadata** 233-235
- metadata extraction, Alfresco** 35
- Metadata Extraction API** 81-85
- Metadata Extractor API** 58
- metadata extractors, Alfresco**
 - about 35
 - DWG 36

- HTML 36
- MIME Email 36
- MSG Email 35
- MS Office 35
- OpenOffice 36
- PDF 35
- StarOffice 36
- metadata pages**
 - properties, displaying in 330-340
- Meta Model XML schema**
 - about 287
 - model 287, 288
 - model.aspects 304, 305
 - model.constraints 292-298
 - model.data-types 291
 - model.imports 289
 - model.namespaces 290
 - model.types 294
- Microsoft Active Directory**
 - about 150
 - authentication, configuring against 159
 - synchronization, configuring against 159
- Microsoft Visio 370**
- MIME 18**
- MIME Email 36**
- MMT**
 - about 106
 - functions 106
- mobile.war file 528**
- mobile web application, Alfresco**
 - accessing 528-532
 - for iPhone 528
 - installing 528
- MobileX application**
 - about 533
 - architecture 534, 539
 - document search feature 537, 538
 - folder and document browsing feature 536, 537
 - running 559
 - user authentication feature 535
- MobileX web application 533**
- model**
 - registering, with repository 325
- Model-View-Controller. *See* MVC**
- model.aspects 304, 305**
- model.constraints, Meta Model XML schema**
 - about 292, 298
 - constraint 292
 - list 293
 - name 292, 293
 - parameter 292
 - ref 292
 - type 292
 - value 293
- model.data-types, Meta Model XML schema**
 - about 291
 - analyser-class 291
 - data-type 291
 - description 291
 - java-class 291
 - name 291
 - title 291
- model.imports, Meta Model XML schema**
 - about 289
 - import 289
 - prefix 289
 - uri 289
- model.namespaces, Meta Model XML schema**
 - about 290
 - namespace 290
 - prefix 290
 - uri 290
- model.types. *See* types definition**
- model.types.type.associations. *See* associations definition**
- model.types.type.properties. *See* properties definition**
- model definition**
 - about 287
 - aspects 288
 - author 287
 - constraints 288
 - data-types 288
 - description 287
 - imports 288
 - name 287
 - namespaces 288
 - published 287
 - types 288
 - version 288

- modeling**
 - tips 306, 307
 - tricks 306, 307
- modelSchema.xsd file** 287
- Modified Date**
 - preserving 346, 347
- module.properties file** 107
- module listing** 106
- Module Management Tool.** *See* MMT
- module updates** 106
- module versioning** 106
- Mozilla Rhino** 24
- MS Active Directory** *See* Microsoft Active Directory
- MSG Email** 35
- MS Office** 35
- MT feature** 23
- Multi-Tenant feature.** *See* MT feature
- multiple LDAP authentication subsystems**
 - configuring 159, 160
- multiple types inheritance, content model**
 - design pattern**
 - about 314
 - example 315
 - issues 314
 - solution 315
- Multipurpose Internet Mail Extensions.** *See* MIME
- MVC** 555
- myCmsService custom beam** 60
- MyDocSite** 44
- mysql command-line utility** 51
- mysql directory** 48
- N**
 - name property** 545
 - namespace** 18
 - namespace definition.** *See* model.namespaces, Meta Model XML schema
 - Naming Service** 180
 - NBT**
 - about 179
 - communication services 180, 183
 - Naming Service 179, 180
 - NBT Datagram service** 185
 - NBT Session service**
 - about 183
 - Session close 184
 - Session establishment 184
 - steady state 184
 - NEGOTIATE parameter** 198
 - NetBIOS** 179
 - NetBIOS Node Type** 182
 - NetBIOS Node Types**
 - Broadcast 182
 - Hybrid 182
 - Mixed 182
 - Peer-to-Peer 182
 - NetBIOS over TCP/IP.** *See* NBT
 - NetbiosSmb device** 186
 - network.automatic-ntlm-auth.trusted-uris**
 - option 140
 - new custom type**
 - defining, for domain 308
 - node, script variables** 394
 - node-leave event handler** 457
 - node associations**
 - about 19
 - types 19
 - node events**
 - about 30
 - beforeCreateNode 30
 - beforeDeleteNode 30
 - beforeUpdateNode 30
 - onCreateNode 30
 - onDeleteNode 30
 - onMoveNode 30
 - onUpdateNode 30
 - onUpdateProperties 30
 - node properties** 18
 - node property sheets** 19
 - node references** 17, 18
 - nodes**
 - about 11, 16
 - associations 19
 - copying, File Folder service used 69
 - metadata 16
 - node reference 17, 18
 - properties 18
 - property sheets 19
 - root node 17
 - node service** 62

node service events

- about 29
- aspect 30
- association 31, 32
- node 30
- store 29

NodeServicePolicies class 81

NT LAN Manager. *See* NLTM

NLTM 139

NLTM authentication

- about 139-141
- troubleshooting 173

NTLMSSP 141

NTLMv1 141

NTLMv2 141

O

onAddAspect event 30

onAsyncActionExecute event 29

onContentPropertyUpdate event 28

onContentRead event 28

onContentUpdate event 28

onCopyComplete event 29

onCreateNode event 30

onCreateReference event 32

onCreateStore event 29

onCreateVersion event 32

onDeleteNode event 30

onModuleLoad method

- implementing 500

onMoveNode event 30

onRemoveAspect event 30

onRemoveReference event 32

On Request Review Output task 372

onRMActionExecution event 32

onUpdateNode event 30

onUpdateProperties event 30

oouser directory 48

OpenLDAP

- authentication, configuring against 153
- group, synchronizing with 155-158
- synchronization, configuring against 153
- user, configuring with 155-158
- user authentication, configuring 153-155

OpenLDAP subsystem configuration

- moving, to its own directory 160

OpenOffice 24, 36

open source projects, Alfresco

- about 24

Acegi Security 24

Apache Abdera 24

Apache Axis 24

Apache Chemistry 24

Apache Commons 24

Apache CXF 24

Apache PDFBox 24

Apache POI 24

Chiba 24

EHCache 24

FreeMarker 24

Greenmail 24

Hibernate 24

iBATIS 24

Java Mail 24

JBOSS jBPM 24

JGroups 24

Lucene 24

Mozilla Rhino 24

OpenOffice 24

OpenSymphony Quartz 24

Spring 24

OpenSymphony Quartz 24

OpsMailmanager AMP 42

org.alfresco.model.ContentModel interface 63

org.alfresco.repo.action.AsynchronousActionExecution QueuePolicies class 29

org.alfresco.repo.admin.patch.AbstractPatch class 98

org.alfresco.repo.admin.patch.Patch interface 98

org.alfresco.repo.content.ContentServicePolicies class 28

org.alfresco.repo.copy.CopyServicePolicies class 28

org.alfresco.repo.importer.ImporterBootstrap object 98

org.alfresco.repo.node.NodeServicePolicies class 29

org.alfresco.repo.version.VersionServicePolicies class 32

OTA 534

outbound variables 432

over-the-air. *See* OTA

owner authority 23

P

package-alfresco-jar 118

parseAtomFeed method 544-550

passthru authentication

about 141

implementing 142

Patch API 57

patches

about 98

creating 98, 99

patching 41

PATH keyword 71

PDF 35

permission groups 22

Permission Service 73, 74

portal architecture 483

portlet class

creating 491-499

portlet container 480

portlet descriptor 482

portlet implementation 482

portlets

about 480

Alfresco content, fetching 483

lifecycle 480, 481

modes 481

standards 480

window states 481

portlets, modes

about 481

edit 481

help 481

view 481

portlets, window states

about 481

maximized 481

minimized 481

normal 481

portlet standards

JSR168 480

JSR286 480

JSR301 480

JSR329 480

Portlet version 1.0. *See* JSR168

Portlet version 2.0. *See* JSR286

post migration processing scripts

about 348

legacy meeting metadata, extracting 350,
352, 353

problems, versioning 349

restrictions, searching 348

unwanted Modified Date updates 349

Presentation Web Scripts 96

Press folder hierarchy 240-242

principal 143

priorityAndWorkerAssigneeHeader 435

priority property 398

processAction method 481

process phase, Swimlane diagram 376

Process State node 429

procId variable 389, 429

Produce or Update Material Brief task 372

project directory structure

_alfresco/config directory 110, 111

_alfresco/source directory 114

_share/config directory 115, 116

about 109

properties

about 285

displaying, in Advanced Search 328, 342,
343

displaying, in content details pages 326,
327

displaying, in metadata pages 330-340

properties, for bmc:job aspect in content-
model.xml file

bmc:campaignId 405

bmc:jobType 405

bmc:product 405

properties, for bmc:job aspect in workflow-
model.xml file

bmw:briefSignOffCategory 406

bmw:conceptWorkDueDate 406

bmw:copywriteWorkDueDate 406

bmw:designWorkDueDate 406

bmw:jobStatus 406

bmw:productionSignOffCategory 406

bmw:workTypes 406

properties definition

- about 295
- constraints 296
- default 296
- description 295
- index 296
- mandatory 296
- multiple 296
- name 295
- property 295
- protected 296
- title 295
- type 295

property files for UI labels, Work process

- creating 401-403

property sheet definitions 19

property sheets

- configuring, for UI display 326
- folder link, adding to 476, 477
- registering 329

property sheets, sign-off process

- about 451-453
- bootstrapping 453

property sheets, studio process

- about 435-437
- bootstrapping 438
- configuring 438

property sheets configuration, work process

- bootstrapping 411, 412

protocol 15

Q

QName

- about 19
- examples 20

R

RBAC 231

READ permission group 74

real-time streaming 13

realm 143

RecentlyAddedDocumentsPortlet 495

record management events

- about 32
- beforeCreateReference 32

- beforeRemoveReference 32
- beforeRMActionExecution 32
- onCreateReference 32
- onRemoveReference 32
- onRMActionExecution 32

Records Management (RM) 135, 281

release notes template 133, 134

release process

- setting up 130-132

release process, setting up

- about 130-132
- change log, updating 135
- release notes template 133, 134
- training 135

Remote Directory servers

- authentication, setting up 152
- synchORIZATION, setting up 152

render method 481

repo 10

repository, Alfresco

- about 10
- overview 11, 12

Repository API 58 95

Repository extensions 104-107

Representational State Transfer (REST) 94

resource file

- registering 329

role-based access control. *See* RBAC

roles 21, 22

root folder node reference

- about 541
- fetching, from CMIS service document 541, 542

root node 17

rules wizards

- aspects, displaying in 328
- types, displaying in 328

runtime executable content transformer 39

S

samba-doc package 165

sambaSamAccount

- adding, to LDAP structure 165, 166

Samba server 149

SASL 149

- script element** 393
- scripts**
 - folders, setting up 251
- script tag** 390
- script variables**
 - executionContext 394
 - node 394
 - task 394
 - taskInstance 394
 - token 394
- SearchController.groovy class** 554
- Search Service** 70-73
- SendReworkEmailToApprover node** 447 449
- service login**
 - performing, via KDC AS 144, 145
- Service Principal Name.** *See* SPN
- services, Alfresco**
 - about 25
 - File Folder Service 26
 - Foundation Services 26
- Servlet Filter** 140
- session service primitives, NetBIOS**
 - about 184
 - call 184
 - hang up 184
 - listen 184
 - receive 184
 - send 184
 - send no ack 184
- setDoAuthentication method** 547
- setupBeforeLoadListener method** 501
- share-config-custom.xml** 117
- share.war file** 104
- Share JAR extension**
 - deploying 122
- showLoginPage method** 552
- Sign-off process**
 - about 379, 380
 - e-mail template 449, 451
 - property files, for UI labels 451
 - task property sheet, bootstrapping 453
 - task property sheets 451-453
 - testing 454
 - workflow content model 448
 - workflow definition (jPDL) 443-448
- Sign-off subprocess**
 - implementing 443
- signoff** 443
- Simple and Protected GSSAPI Negotiation Mechanism.** *See* SPNEGO
- Simple Authentication and Security Layer.** *See* SASL
- single sign-on.** *See* SSO
- site-data directory** 108
- site-webscripts directory** 108
- Slingshot** 108
- smartphones** 534
- SMB (Server Message Block) protocol** 177
- SMB2** 190
- SMB transport protocols**
 - identifying 186
- smoke test** 131
- Solid-State Drives (SSD)** 13
- source code, Alfresco**
 - downloading 50
- source definition**
 - about 299
 - mandatory 299
 - many 299
 - role 299
- SpacesStore** 15
- space templates**
 - setting up 269, 270
- SPN** 146
- SPNEGO** 141, 144
- Spring** 24
- Spring Dispatcher Servlet** 108
- Spring Surf** 107
- Squirrel SQL Client** 51
- SSO**
 - about 137, 139
 - using 172
- Standalone clients** 104
- standard development environment**
 - creating, reasons 103
- standard portlet deployment descriptor**
 - creating 494, 495, 503
- StarOffice** 36
- start-state node** 390 429
- StartConceptWork node** 431
- startDate property** 398

- status property** 398
- store events**
 - about 29
 - beforeCreateStore 29
 - onCreateStore 29
- store reference**
 - about 15
 - identifier 15
 - list 15, 16
 - protocol 15
- store root** 17
- stores, Alfresco**
 - about 12
 - Archive Store 12
 - AVM Store 14
 - Content Store 12
 - System Store 12
 - Version Store 12
 - Working Store 12
- studio** 428
- Studio process**
 - about 380, 381
 - property files, for UI labels 435
 - property sheets, configuring 438
 - task property sheets 435-437
 - task property sheets, bootstrapping 438
 - testing 439-443
 - workflow content model 434
 - workflow definition (jPDL) 428-433
- Studio subprocess**
 - implementing 428
- Studio Team Manager** 428
- studioTeamManagers swimlane** 388
- subfolders, Data Dictionary top folder**
 - about 226
 - Email Actions 226
 - Email Templates 226
 - Imap Configs 226
 - Messages 226
 - Models 226
 - Presentation Templates 226
 - Records Management 226
 - Rendering Actions Space 226
 - RSS Templates 226
 - Saved Searches 227
 - Scripts 227
 - Space Templates 227
 - Transfers 227
 - Web Client Extension 227
 - Web Forms 227
 - Web Scripts 227
 - Web Scripts Extensions 227
 - Workflow Definitions 227
- subprocesses, Swimlane diagram**
 - about 373
 - advantages 374
- subsystems, Alfresco**
 - about 40
 - characteristics 40
 - examples 40
- super states** 455
- Swimlane diagram**
 - about 370, 385, 428
 - better version 373
 - business processes, designing with 370, 372
 - features 370, 372
 - process phase 376
 - subprocesses 373, 375
 - task metadata 375, 376
 - task naming convention 377
- swimlanes**
 - about 388
 - jobOwner 388
 - studioTeamManagers 388
 - worker 388
- synchronization**
 - configuring, against Microsoft Active Directory 159
 - configuring, against OpenLDAP 153
 - configuring, with Active Directory 161, 162
 - setting up, with Remote Directory servers 152
- synchronization.autoCreatePeopleOnLogin property** 159
- synchronization.import.cron property** 158
- synchronization.synchronizeChangesOnly property** 158
- synchronization.syncOnStartup property** 159
- synchronization.syncWhenMissingPeopleLogin property** 158
- sys:aspect_root** 17
- System Store** 12

T

target definition

- about 300
- class 300
- mandatory 300
- many 300
- role 300

task, script variables 394

task-create event 444 404

task-node element 391

TaskAssignmentNotificationActionHandler 470

task due date, Work process

- setting 404, 405

taskInstance, script variables 394

taskInstance variable 405

taskkill command 128

task metadata, Swimlane diagram 375, 376

task naming convention, Swimlane diagram 377

Task nodes 428

task property sheets, Work process

- creating 406-410

task summary list

- exporting, in Excel spreadsheet 475, 476

TCP/IP 185

template

- dealing, with Best Money client 134

Tenant Administration Console 23

terms, Kerberos installation

- about 143
- GSS-API 144
- GSS-SPNEGO 144
- principal 143
- realm 143
- ticket 144

TGS 143

TGT 145

third-party extensions, Alfresco

- about 42
- Alfresco Bulk Filesystem Import 42
- Enterprise Reporting AMP 42
- OpsMailmanager AMP 42
- Thumbnails AMP 42

Thumbnails AMP 42

ticket 144

Ticket Granting Server. *See* TGS

Ticket Granting Ticket. *See* TGT

token 418

token, script variables 394

tomcat directory

- about 49
- structure 49

tool import

- about 355, 356
- advantages 356
- disadvantages 357

training

- about 135
- steps 135

TRANSACTION_COMMIT 33

transaction management, Java Foundation Services API 59-62

transient properties 334

troubleshooting, Alfresco CIFS

- about 215
- general tips 215, 216

troubleshooting strategies, Alfresco CIFS

- about 217
- CIFS server NetBIOS name, verifying 219
- Citrix XenApp virtualization environment, checking 220
- debug logging, enabling 217
- NetBIOS name registration, verifying 218
- ports, checking from client 218
- ports, checking from server 217

types

- about 285
- displaying 341, 342
- displaying, in add content wizards 327
- displaying, in create content wizards 327
- displaying, in rules wizards 328
- uses 309

types definition

- about 294
- archive 294
- associations 295
- description 294
- mandatory-aspects 295
- name 294
- overrides 295

- parent 294
- properties 295
- title 294
- type 294

U

- UI controllers**
 - implementing 551-554
- UI display**
 - property sheets, configuring for 326
- UI framework 485**
- UI Labels**
 - localizing 327
- unit testing 131**
- Universally Unique Identifier.** *See* **UUID**
- unused inbound variables 432**
- URL Rewriter Filter 108**
- user**
 - authenticating, with Alfresco 543, 544
 - configuring, with OpenLDAP 155-158
 - folder hierarchies, copying between 281
- user authentication**
 - configuring, with OpenLDAP 153-155
- user authentication feature**
 - for MobileX application 535
- User Interface customizations 104**
- user interfaces clients, Alfresco**
 - about 43
 - Alfresco CIFS 45
 - Alfresco Explorer 43
 - Alfresco Mobile 45
 - Alfresco Share 43
 - Alfresco SharePoint 43, 44
- user login**
 - performing, via KDC AS 144, 145
- UUID 18**

V

- ValidateWork task 386**
- variable element 393**
- version history 12**
- version service events**
 - about 32
 - afterCreateVersion 32

- beforeCreateVersion 32
- calculateVersionLabel 32
- onCreateVersion 32

- Version Store 12**

W

- W01_ProduceWork task 399, 404**
- W03_ValidateWork task node 392**
- W04_AmendWork task 399**
- WAP 534**
- WCM development receiver 40**
- Web-based Distributed Authoring and Versioning.** *See* **WebDAV**
- web-extension directory**
 - configuration files 117
- web-framework-config-custom.xml 117**
- web application archive (WAR) 59**
- webapps directory 50**
- WebDAV 177**
- WebDAV clients**
 - about 212
 - BitKinex 212
 - WebDrive 213
- WebDAV Mini Redirector**
 - about 214
 - limitations 214
- WebDrive 213**
- Web Folders**
 - about 214
 - using, for connecting to Alfresco 214
- webscript-framework-config-custom.xml 117**
- Web Scripts**
 - about 95
 - types 96
- Web Service API 58**
- WiFi Protected Setup.** *See* **WPS**
- Windows built-in WebDAV clients**
 - about 214
 - WebDAV Mini Redirector (XP, Vista, and Win7) 214
 - Web Folders (XP only) 214
- WINS 180**
- Wireless Application Protocol.** *See* **WAP**

- Wireless Markup Language. *See* WML
- WML 534
- workApprovedByJobOwner, Boolean variables 393
- workApprovedByTeamMgr, Boolean variables 393
- worker swimlane 388, 399
- Workflow Content Model 387
- workflow content model 394-401
- workflow content model, job process 467
- workflow content model, sign-off process 448
- workflow content model, studio process 434
- workflow definition 418
- workflow definition (jPDL), job process 455-466
- workflow definition (jPDL), sign-off process 443-448
- workflow definition (jPDL), studio process 428-433
- workflow definition, Work process
 - defining 386-394
- workflow instance 418
- workflow solution
 - extending 469
- workflow solution, extending
 - about 469
 - customized task dashlets 471
 - e-mail notification, adding 470
 - management dashlets 471-474
 - task summary list, exporting in Excel 475, 476

- Working Store
 - about 12
 - versus AVM Store 15
- work process 381, 429
- workspace element 541
- Work subprocess
 - about 385
 - dynamic descriptions, using 404, 405
 - job data, defining 405, 406
 - property files for UI labels 401-403
 - property sheets configuration, bootstrapping 411, 412
 - running, from Alfresco Share UI 418-425
 - task due date, setting 404, 405
 - task property sheets, creating 406-410
 - testing 412-418
 - UI property files, bootstrapping 411, 412
 - workflow content model 394-401
 - workflow definition 386-394
- WPS 535
- WRITE permission group 74

X

- XForms engine 330
- XmlSlurper 544
- XPATH query feature 70



Thank you for buying **Alfresco 3 Business Solutions**

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: www.packtpub.com.

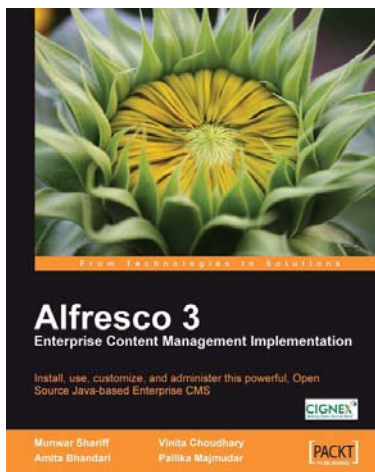
About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.



Alfresco 3 Enterprise Content Management Implementation

ISBN: 978-1-847197-36-8 Paperback: 600 pages

How to customize, use, and administer this powerful, Open Source Java-based Enterprise CMS

1. Manage your business documents with version control, library services, content organization, and advanced search
2. Create collaborative web sites using document libraries, wikis, blogs, forums, calendars, discussions, and social tagging
3. Integrate with external applications such as Liferay Portal, Adobe Flex, iPhone, iGoogle, and Facebook



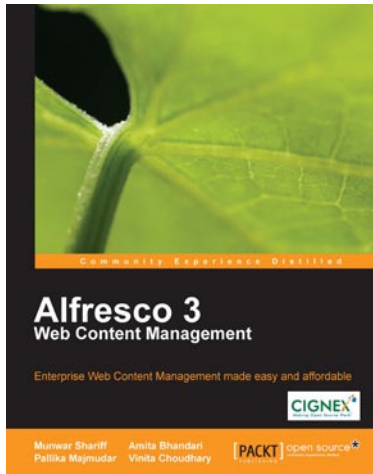
Alfresco 3 Web Services

ISBN: 978-1-849511-52-0 Paperback: 436 pages

Build Alfresco applications using Web Services, WebScripts and CMIS

1. Gain a comprehensive overview of the specifications of Web services
2. Implement the Alfresco specific Web Services
3. Get to grips with the Alfresco WebScripts and the Alfresco extensible RESTful API
4. Manipulate contents in Alfresco using different operations and APIs
5. Learn about the CMIS specification and its Alfresco implementation

Please check www.PacktPub.com for information on our titles



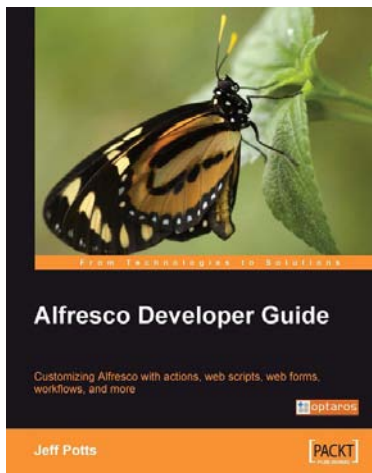
Alfresco 3 Web Content Management

ISBN: 978-1-847198-00-6

Paperback: 440 pages

Create an infrastructure to manage all your web content, and deploy it to various external production systems

1. A complete guide to Web Content Creation and Distribution
2. Understand the concepts and advantages of Publishing-style Web CMS
3. Leverage a single installation to manage multiple websites
4. Integrate Alfresco web applications with external systems



Alfresco Developer Guide

ISBN: 978-1-847193-11-7

Paperback: 556 pages

Customizing Alfresco with actions, web scripts, web forms, workflows, and more

1. Learn to customize the entire Alfresco platform, including both Document Management and Web Content Management
2. Jam-packed with real-world, step-by-step examples to jump start your development
3. Content modeling, custom actions, Java API, RESTful web scripts, advanced workflow
4. This book covers Alfresco Enterprise Edition version 2.2

Please check www.PacktPub.com for information on our titles



Alfresco 3 Records Management

ISBN: 978-1-849514-36-1 Paperback: 488 pages

Comply with regulations and secure your organization's records with Alfresco Records Management

1. Successfully implement your records program using Alfresco Records Management, fully certified for DoD-5015.2 compliance
2. The first and only book to focus exclusively on Alfresco Records Management
3. Step-by-step instructions describe how to identify records, organize records, and manage records to comply with regulatory requirements



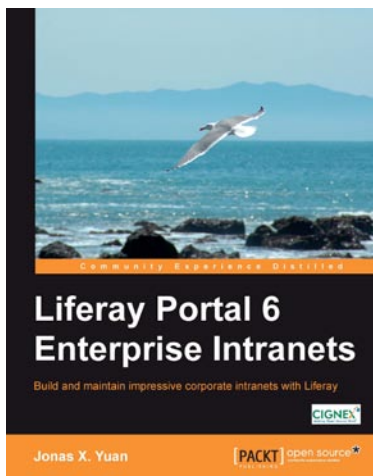
Alfresco Enterprise Content Management Implementation

ISBN: 978-1-904811-11-4 Paperback: 356 pages

How to Install, use, and customize this powerful, free, Open Source Java-based Enterprise CMS

1. Manage your business documents: version control, library services, content organization, and search
2. Workflows and business rules: move and manipulate content automatically when events occur
3. Maintain, extend, and customize Alfresco: backups and other admin tasks, customizing and extending the content model, creating your own look and feel

Please check www.PacktPub.com for information on our titles



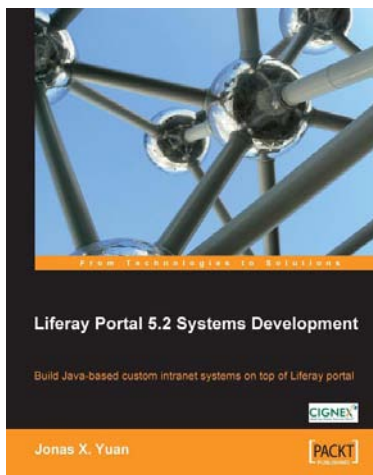
Liferay Portal 6 Enterprise Intranets

ISBN: 978-1-849510-38-7

Paperback: 692 pages

Build and maintain impressive corporate intranets with Liferay

1. Develop a professional Intranet using Liferay's practical functionality, usability, and technical innovation
2. Enhance your Intranet using your innovation and Liferay Portal's out-of-the-box portlets
3. Maximize your existing and future IT investments by optimizing your usage of Liferay Portal



Liferay Portal 5.2 Systems Development

ISBN: 978-1-847194-70-1

Paperback: 552 pages

Liferay Portal 5.2 Systems Development

1. Learn to use Liferay tools to create your own applications as a Java developer, with hands-on examples
2. Customize Liferay portal using the JSR-286 portlet, extension environment, and Struts framework
3. Build your own Social Office with portlets, hooks, and themes and manage your own community
4. The only Liferay book aimed at Java developers

Please check www.PacktPub.com for information on our titles