EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

The information in this publication is provided as is. EMC Corporation makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose. Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.

For the most up-to-date listing of EMC product names, see EMC Corporation Trademarks on EMC.com. All other trademarks used herein are the property of their respective owners.

© Copyright 2012 EMC Corporation. All rights reserved.
# Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Overview</th>
<th>Data Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Chapter 1</td>
<td>Overview</td>
<td>Data Package</td>
</tr>
<tr>
<td></td>
<td>What is DFS?</td>
<td>Example</td>
</tr>
<tr>
<td></td>
<td>Service orientation</td>
<td>DataObject</td>
</tr>
<tr>
<td></td>
<td>DFS SDK</td>
<td>DataObject related classes</td>
</tr>
<tr>
<td></td>
<td>Setting up dfc.properties</td>
<td>DataObject type</td>
</tr>
<tr>
<td></td>
<td>Setting up Java classpaths</td>
<td>DataObject construction</td>
</tr>
<tr>
<td></td>
<td>Public packages and namespaces</td>
<td>ObjectIdentity</td>
</tr>
<tr>
<td></td>
<td>DFS consumers</td>
<td>ObjectIdentitySet</td>
</tr>
<tr>
<td></td>
<td>WSDL-based consumer development</td>
<td>Example</td>
</tr>
<tr>
<td></td>
<td>Java client library consumers</td>
<td>Property</td>
</tr>
<tr>
<td></td>
<td>Location transparency</td>
<td>Property model</td>
</tr>
<tr>
<td></td>
<td>Configuring service addressing in Java</td>
<td>Transient properties</td>
</tr>
<tr>
<td></td>
<td>Running the Java consumer samples</td>
<td>Loading properties: convenience API</td>
</tr>
<tr>
<td></td>
<td>.NET client library consumers</td>
<td>ArrayProperty</td>
</tr>
<tr>
<td></td>
<td>.NET consumer project dependencies</td>
<td>ValueAction</td>
</tr>
<tr>
<td></td>
<td>.NET client configuration</td>
<td>Deleting a repeating property: use of empty value</td>
</tr>
<tr>
<td></td>
<td>Running the C# consumer samples</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enterprise Content Services</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Service development and generation tools</td>
<td></td>
</tr>
</tbody>
</table>

EMC Documentum Foundation Services Version 6 SP1 Development Guide 3
<table>
<thead>
<tr>
<th>PropertySet</th>
<th>38</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>38</td>
</tr>
<tr>
<td>PropertyProfile</td>
<td>39</td>
</tr>
<tr>
<td>Example</td>
<td>39</td>
</tr>
<tr>
<td>Content</td>
<td>40</td>
</tr>
<tr>
<td>ContentProfile</td>
<td>40</td>
</tr>
<tr>
<td>PostTransferAction</td>
<td>41</td>
</tr>
<tr>
<td>Example</td>
<td>41</td>
</tr>
<tr>
<td>Permissions</td>
<td>42</td>
</tr>
<tr>
<td>PermissionProfile</td>
<td>43</td>
</tr>
<tr>
<td>Compound (hierarchical) permissions</td>
<td>43</td>
</tr>
<tr>
<td>Example</td>
<td>43</td>
</tr>
<tr>
<td>Relationship</td>
<td>44</td>
</tr>
<tr>
<td>ReferenceRelationship and ObjectRelationship</td>
<td>44</td>
</tr>
<tr>
<td>Relationship model</td>
<td>44</td>
</tr>
<tr>
<td>Relationship fields</td>
<td>45</td>
</tr>
<tr>
<td>RelationshipIntentModifier</td>
<td>45</td>
</tr>
<tr>
<td>Relationship targetRole</td>
<td>46</td>
</tr>
<tr>
<td>DataObject as data graph</td>
<td>46</td>
</tr>
<tr>
<td>DataObject graph structural types</td>
<td>47</td>
</tr>
<tr>
<td>Standalone DataObject</td>
<td>47</td>
</tr>
<tr>
<td>DataObject with references</td>
<td>48</td>
</tr>
<tr>
<td>Compound DataObject instances</td>
<td>49</td>
</tr>
<tr>
<td>Compound DataObject with references</td>
<td>50</td>
</tr>
<tr>
<td>Removing object relationships</td>
<td>50</td>
</tr>
<tr>
<td>RelationshipProfile</td>
<td>51</td>
</tr>
<tr>
<td>ResultDataMode</td>
<td>51</td>
</tr>
<tr>
<td>Relationship filters</td>
<td>52</td>
</tr>
<tr>
<td>DepthFilter restrictions</td>
<td>52</td>
</tr>
<tr>
<td>Other classes related to DataObject</td>
<td>53</td>
</tr>
</tbody>
</table>

**Chapter 3**  
**Enterprise Content Services** ................................................................. 55  
Enterprise Content Services ........................................................................... 55  
Service commonalities .................................................................................. 56  
Service Context .............................................................................................. 56  
Setting up service context ............................................................................ 57  
Identities ...................................................................................................... 57  
Service context properties .......................................................................... 58  
Context registration .................................................................................... 58  
Getting service context for a chained service ............................................. 58  
Service instantiation .................................................................................... 59  
Service instantiation in Java ....................................................................... 59  
Service instantiation in C# ......................................................................... 59  
OperationOptions ......................................................................................... 60  

**Chapter 4**  
**Object Service** .......................................................................................... 61  
create operation ............................................................................................ 61  
Java syntax .................................................................................................... 62  
Parameters .................................................................................................... 62  
Profiles ......................................................................................................... 62  
Response ....................................................................................................... 62  
Examples ........................................................................................................ 63  
Simple object creation .................................................................................. 63  
Creating and linking ..................................................................................... 63  
Creating multiple related objects ................................................................. 65
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>createPath operation</td>
<td>66</td>
</tr>
<tr>
<td>Java syntax</td>
<td>66</td>
</tr>
<tr>
<td>Parameters</td>
<td>66</td>
</tr>
<tr>
<td>Response</td>
<td>66</td>
</tr>
<tr>
<td>Example</td>
<td>66</td>
</tr>
<tr>
<td>Get operation</td>
<td>67</td>
</tr>
<tr>
<td>Description</td>
<td>67</td>
</tr>
<tr>
<td>Java syntax</td>
<td>67</td>
</tr>
<tr>
<td>Parameters</td>
<td>67</td>
</tr>
<tr>
<td>Profiles</td>
<td>68</td>
</tr>
<tr>
<td>Response</td>
<td>68</td>
</tr>
<tr>
<td>Example</td>
<td>68</td>
</tr>
<tr>
<td>Controlling data returned by get operation</td>
<td>69</td>
</tr>
<tr>
<td>Filtering properties using PropertyProfile</td>
<td>69</td>
</tr>
<tr>
<td>Controlling Relationship instances</td>
<td>70</td>
</tr>
<tr>
<td>Filtering content</td>
<td>71</td>
</tr>
<tr>
<td>Getting content from external sources</td>
<td>72</td>
</tr>
<tr>
<td>update operation</td>
<td>73</td>
</tr>
<tr>
<td>Java syntax</td>
<td>73</td>
</tr>
<tr>
<td>Parameters</td>
<td>73</td>
</tr>
<tr>
<td>Profiles</td>
<td>73</td>
</tr>
<tr>
<td>Response</td>
<td>74</td>
</tr>
<tr>
<td>Examples</td>
<td>74</td>
</tr>
<tr>
<td>Updating properties</td>
<td>74</td>
</tr>
<tr>
<td>Modifying a repeating properties (attributes) list</td>
<td>75</td>
</tr>
<tr>
<td>Updating object relationships</td>
<td>77</td>
</tr>
<tr>
<td>delete operation</td>
<td>78</td>
</tr>
<tr>
<td>Description</td>
<td>78</td>
</tr>
<tr>
<td>Java Syntax</td>
<td>78</td>
</tr>
<tr>
<td>Parameters</td>
<td>78</td>
</tr>
<tr>
<td>DeleteProfile</td>
<td>79</td>
</tr>
<tr>
<td>Example</td>
<td>79</td>
</tr>
<tr>
<td>copy operation</td>
<td>80</td>
</tr>
<tr>
<td>Description</td>
<td>80</td>
</tr>
<tr>
<td>Java Syntax</td>
<td>80</td>
</tr>
<tr>
<td>Parameters</td>
<td>81</td>
</tr>
<tr>
<td>CopyProfile</td>
<td>81</td>
</tr>
<tr>
<td>Response</td>
<td>81</td>
</tr>
<tr>
<td>Examples</td>
<td>82</td>
</tr>
<tr>
<td>Copy across repositories</td>
<td>82</td>
</tr>
<tr>
<td>Copy with modifications</td>
<td>83</td>
</tr>
<tr>
<td>move operation</td>
<td>84</td>
</tr>
<tr>
<td>Description</td>
<td>84</td>
</tr>
<tr>
<td>Java syntax</td>
<td>84</td>
</tr>
<tr>
<td>Parameters</td>
<td>84</td>
</tr>
<tr>
<td>MoveProfile</td>
<td>85</td>
</tr>
<tr>
<td>Response</td>
<td>85</td>
</tr>
<tr>
<td>Example</td>
<td>85</td>
</tr>
<tr>
<td>validate operation</td>
<td>87</td>
</tr>
<tr>
<td>Java syntax</td>
<td>87</td>
</tr>
<tr>
<td>Parameters</td>
<td>87</td>
</tr>
<tr>
<td>Response</td>
<td>87</td>
</tr>
<tr>
<td>getObjectContentUrls operation</td>
<td>88</td>
</tr>
<tr>
<td>Description</td>
<td>88</td>
</tr>
<tr>
<td>Java syntax</td>
<td>88</td>
</tr>
<tr>
<td>Parameters</td>
<td>88</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>VersionControl Service</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>getCheckoutInfo operation</td>
<td>91</td>
</tr>
<tr>
<td>Description</td>
<td>91</td>
</tr>
<tr>
<td>Java syntax</td>
<td>91</td>
</tr>
<tr>
<td>Parameters</td>
<td>92</td>
</tr>
<tr>
<td>Response</td>
<td>92</td>
</tr>
<tr>
<td>Example</td>
<td>92</td>
</tr>
<tr>
<td>checkout operation</td>
<td>93</td>
</tr>
<tr>
<td>Description</td>
<td>93</td>
</tr>
<tr>
<td>Java syntax</td>
<td>93</td>
</tr>
<tr>
<td>Parameters</td>
<td>94</td>
</tr>
<tr>
<td>Response</td>
<td>94</td>
</tr>
<tr>
<td>Example</td>
<td>94</td>
</tr>
<tr>
<td>checkin operation</td>
<td>95</td>
</tr>
<tr>
<td>Description</td>
<td>95</td>
</tr>
<tr>
<td>Java syntax</td>
<td>96</td>
</tr>
<tr>
<td>Parameters</td>
<td>96</td>
</tr>
<tr>
<td>VersionStrategy values</td>
<td>96</td>
</tr>
<tr>
<td>CheckinProfile</td>
<td>97</td>
</tr>
<tr>
<td>Response</td>
<td>97</td>
</tr>
<tr>
<td>Example</td>
<td>97</td>
</tr>
<tr>
<td>cancelCheckout operation</td>
<td>99</td>
</tr>
<tr>
<td>Description</td>
<td>99</td>
</tr>
<tr>
<td>Java syntax</td>
<td>99</td>
</tr>
<tr>
<td>Parameters</td>
<td>99</td>
</tr>
<tr>
<td>Example</td>
<td>99</td>
</tr>
<tr>
<td>deleteVersion operation</td>
<td>100</td>
</tr>
<tr>
<td>Description</td>
<td>100</td>
</tr>
<tr>
<td>Java syntax</td>
<td>100</td>
</tr>
<tr>
<td>Parameters</td>
<td>100</td>
</tr>
<tr>
<td>Example</td>
<td>100</td>
</tr>
<tr>
<td>deleteAllVersions operation</td>
<td>101</td>
</tr>
<tr>
<td>Description</td>
<td>101</td>
</tr>
<tr>
<td>Java syntax</td>
<td>101</td>
</tr>
<tr>
<td>Parameters</td>
<td>101</td>
</tr>
<tr>
<td>Example</td>
<td>101</td>
</tr>
<tr>
<td>getCurrent operation</td>
<td>102</td>
</tr>
<tr>
<td>Description</td>
<td>102</td>
</tr>
<tr>
<td>Java syntax</td>
<td>102</td>
</tr>
<tr>
<td>Parameters</td>
<td>103</td>
</tr>
<tr>
<td>Response</td>
<td>103</td>
</tr>
<tr>
<td>Example</td>
<td>103</td>
</tr>
<tr>
<td>getVersionInfo operation</td>
<td>104</td>
</tr>
<tr>
<td>Description</td>
<td>104</td>
</tr>
<tr>
<td>Java syntax</td>
<td>104</td>
</tr>
<tr>
<td>Parameters</td>
<td>104</td>
</tr>
<tr>
<td>Response</td>
<td>104</td>
</tr>
<tr>
<td>Response</td>
<td>104</td>
</tr>
<tr>
<td>Example</td>
<td>104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 6</th>
<th>Schema Service</th>
<th>107</th>
</tr>
</thead>
<tbody>
<tr>
<td>Common schema classes</td>
<td>107</td>
<td></td>
</tr>
<tr>
<td>TypeInfo</td>
<td>107</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 8

| QueryResult | 137 |
| QueryStatus | 137 |
| RepositoryStatusInfo | 138 |
| RepositoryStatus | 138 |
| getRepositoryList operation | 139 |
| Description | 139 |
| Java syntax | 140 |
| Parameters | 140 |
| Returns | 140 |
| Repository | 140 |
| Example | 140 |

execute operation | 141 |
| Description | 141 |
| Java syntax | 141 |
| Parameters | 142 |
| Returns | 142 |
| Examples | 142 |
| Simple passthrough query | 142 |
| Structured query | 144 |

Chapter 9  Workflow Service 147

Workflow SBO dependency | 147 |

getProcessTemplates operation | 148 |
| Description | 148 |
| Java syntax | 148 |
| Parameters | 148 |
| Returns | 148 |
| Example | 149 |

getProcessInfo operation | 150 |
| Description | 150 |
| Java syntax | 150 |
| Parameters | 150 |
| Returns | 150 |
| Example | 150 |

startProcess operation | 151 |
| Description | 151 |
| Java syntax | 151 |
| Parameters | 152 |
| Returns | 152 |
| Example | 152 |

Chapter 10  Content and Content Transfer 157

Content transfer topologies and optimization | 157 |
| Content transfer modes | 158 |
| base64 | 158 |
| MTOM | 159 |
Building Custom Services Using DFS

Chapter 11

Memory limitations associated with MTOM content transfer mode .......... 159
Workarounds .................................................................................. 160
UCF ............................................................................................... 161
UCF is not supported in middle tier ................................................. 161
Content model ............................................................................... 161
ContentTransferProfile ................................................................... 162
UCF content transfer in DFS ............................................................. 163
UCF transfer to and from DFS consumers ........................................ 163
Client-orchestrated UCF transfer ...................................................... 164
Web applications with DFS services and consumer running in separate applications ........................................................................ 165
Enabling UCF transfer in a web application ....................................... 166
Set up the development environment ............................................... 167
Configure the Apache reverse proxy as a request forwarder ............ 168
Code an HTML user interface for serving the applet ......................... 168
Write the applet code for deploying and launching UCF .................... 171
Build and bundle the applet ............................................................ 172
Sign the applet ............................................................................... 172
Create a servlet for orchestrating the UCF content transfer ............... 172
Alternative methods of supplying ActivityInfo and their relative precedence ................................................................................... 175
Optimization: controlling UCF connection closure ........................... 175
Opening a transferred document in a viewer/editor .......................... 176

Chapter 11 Building Custom Services Using DFS .................................. 177
Service design considerations .......................................................... 177
SBO or POJO services ...................................................................... 177
DFS object model .......................................................................... 178
Creating inputs to DFS tools ............................................................ 178
Service packaging, namespace, and address ..................................... 178
Overriding default targetNamespace and artifact packaging ............. 180
The well-behaved service implementation ........................................ 181
Service annotation ......................................................................... 182
Data type and field annotation ......................................................... 183
Best practices for data type naming and annotation .......................... 184
Data type annotation ..................................................................... 184
Fields and accessors ...................................................................... 184
Things to avoid ............................................................................. 185
Other pitfalls .................................................................................. 186
Sample service ............................................................................... 186
Getting a DFC session manager ....................................................... 187
Invoking a DFS-provided service ..................................................... 187
Service test consumers .................................................................... 188
Tools for generating services ........................................................... 190
Apache Ant ..................................................................................... 190
generateModel task ...................................................................... 191
generateArtifacts task .................................................................... 191
buildService task .......................................................................... 192
packageService task ....................................................................... 193
Generating C# proxies .................................................................... 194
Creating shared assemblies for data objects shared by multiple services ................................................................................... 195
Exploring AcmeCustomService ....................................................... 196
Service test consumers .................................................................... 196
build.properties ............................................................................. 196
<table>
<thead>
<tr>
<th>File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>autodeploy.properties</td>
<td>197</td>
</tr>
<tr>
<td>dfs-client.xml</td>
<td>197</td>
</tr>
<tr>
<td>dfc.properties</td>
<td>198</td>
</tr>
<tr>
<td>build.xml</td>
<td>198</td>
</tr>
<tr>
<td>Running the tools</td>
<td>198</td>
</tr>
</tbody>
</table>

### Appendix A

**Guidelines for Migrating from Web Services Framework to DFS**

- WSF and DFS .................................................. 201
- Candidates for direct conversion .................................. 201
- DFS facade .......................................................... 202
- Building SBO services ................................................. 202
- Security model and service context ..................................... 202
- Content transfer .................................................... 203
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Property class hierarchy</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>ArrayProperty model</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>Relationship model</td>
<td>45</td>
</tr>
<tr>
<td>4</td>
<td>Relationship tree</td>
<td>47</td>
</tr>
<tr>
<td>5</td>
<td>Standalone DataObject</td>
<td>47</td>
</tr>
<tr>
<td>6</td>
<td>DataObject with references</td>
<td>48</td>
</tr>
<tr>
<td>7</td>
<td>DataObject with parent and child references</td>
<td>48</td>
</tr>
<tr>
<td>8</td>
<td>Compound DataObject</td>
<td>49</td>
</tr>
<tr>
<td>9</td>
<td>Compound object with references</td>
<td>50</td>
</tr>
<tr>
<td>10</td>
<td>Removing a relationship</td>
<td>51</td>
</tr>
<tr>
<td>11</td>
<td>Restriction when depth &gt; 1</td>
<td>53</td>
</tr>
<tr>
<td>12</td>
<td>ValidationInfoSet</td>
<td>88</td>
</tr>
<tr>
<td>13</td>
<td>Content class hierarchy</td>
<td>162</td>
</tr>
<tr>
<td>14</td>
<td>Establishing UCF connection and settings</td>
<td>164</td>
</tr>
<tr>
<td>15</td>
<td>UCF transfer to client</td>
<td>164</td>
</tr>
<tr>
<td>16</td>
<td>Topology with web application and service on separate hosts</td>
<td>165</td>
</tr>
<tr>
<td>17</td>
<td>Reverse proxy functioning as request forwarder</td>
<td>166</td>
</tr>
<tr>
<td>18</td>
<td>User interface for UCF test application</td>
<td>169</td>
</tr>
<tr>
<td>19</td>
<td>DfsProxyGen form</td>
<td>194</td>
</tr>
</tbody>
</table>
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>DFS technologies</td>
<td>15</td>
</tr>
<tr>
<td>Table 2</td>
<td>Public API Java packages</td>
<td>18</td>
</tr>
<tr>
<td>Table 3</td>
<td>Public .NET namespaces</td>
<td>19</td>
</tr>
<tr>
<td>Table 4</td>
<td>app.config settings</td>
<td>23</td>
</tr>
<tr>
<td>Table 5</td>
<td>DataObject related classes</td>
<td>29</td>
</tr>
<tr>
<td>Table 6</td>
<td>Enterprise Content Services provided with the DFS product</td>
<td>55</td>
</tr>
<tr>
<td>Table 7</td>
<td>Methods for instantiating services</td>
<td>59</td>
</tr>
<tr>
<td>Table 8</td>
<td>ExpressionValue subtypes</td>
<td>136</td>
</tr>
<tr>
<td>Table 9</td>
<td>PostTransferAction strings</td>
<td>176</td>
</tr>
<tr>
<td>Table 10</td>
<td>Sample service packages</td>
<td>179</td>
</tr>
<tr>
<td>Table 11</td>
<td>Java intrinsic type to XML mappings</td>
<td>181</td>
</tr>
<tr>
<td>Table 12</td>
<td>Java intrinsic type to XML mappings for arrays</td>
<td>181</td>
</tr>
<tr>
<td>Table 13</td>
<td>DfsBofService attributes</td>
<td>182</td>
</tr>
<tr>
<td>Table 14</td>
<td>DfsPojoService attributes</td>
<td>182</td>
</tr>
<tr>
<td>Table 15</td>
<td>Sample service build.xml Ant targets</td>
<td>198</td>
</tr>
</tbody>
</table>
This document is a guide to using EMC Documentum Foundation Services (DFS) for the development of DFS service consumers, and of custom DFS services. This document is not a comprehensive DFS reference. For additional information, refer to the Javadocs, to the sample code delivered with the DFS SDK, and to published white papers that addressed specialized topics on DFS development. For information on installation and deployment of DFS, refer to the Documentum Foundation Services Installation Guide.

Intended readership

This document is intended for developers and architects building consumers of DFS services, and for service developers seeking to extend DFS services with custom services. This document will also be of interest to managers and decision makers seeking to determine whether DFS would offer value to their organization.

Revision History

The following changes have been made to this document.

<table>
<thead>
<tr>
<th>Revision Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>December 2007</td>
<td>Initial publication</td>
</tr>
<tr>
<td>April 2012</td>
<td>Updated the Content and Content Transfer chapter with MTOM and UCF limitations</td>
</tr>
</tbody>
</table>

Conventions for referring to the APIs

This development guide covers the DFS Java and C# API, with occasional references to the web services SOAP API. All of the APIs use the same underlying data model, but have different approaches regarding naming conventions, data encapsulation, and exceptions. To avoid needless repetition, we have adopted the following conventions.
For public method names C# conventionally uses Pascal case (for example MyMethod), while Java uses "camel case" (myMethod). References to specific methods are generally avoided in data model descriptions, but where they are unavoidable, we have used the Java spelling convention. For service operation signatures, we have provided the Java signature, which includes a throws clause. The C# signature will be identical, except for the initial capitalization of the method (operation) name and the absence of the throws clause. (C# does not have a throws clause in method declarations, because it does not have checked exceptions.)

Java uses getter and setter methods for data encapsulation and C# uses properties. In this case we refer to the data as a "setting" or "field" in the form in which it is represented in the SOAP API (and also internally as a private class field in Java and C#). For example:

<table>
<thead>
<tr>
<th>SOAP element</th>
<th>C# Property</th>
<th>Java accessors</th>
</tr>
</thead>
<tbody>
<tr>
<td>sourceLocation</td>
<td>SourceLocation</td>
<td>getSourceLocation,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>setSourceLocation</td>
</tr>
</tbody>
</table>

As the correspondence between these forms of data access is consistent, it is our hope that readers will take this in their stride.
Overview

This chapter is intended to provide a brief overview of DFS products and technologies. This chapter covers the following topics:

- What is DFS?, page 15
- Service orientation, page 16
- DFS SDK, page 17
- DFS consumers, page 20
- Enterprise Content Services, page 24

What is DFS?

EMC Documentum Foundation Services (DFS) are a set of technologies that enable service-oriented programmatic access to the EMC Documentum Content Server platform and related products. It includes the following technologies.

<table>
<thead>
<tr>
<th>Table 1. DFS technologies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DFS technology</strong></td>
</tr>
<tr>
<td>Enterprise Content Services</td>
</tr>
<tr>
<td>Data model and API</td>
</tr>
<tr>
<td>Runtime support</td>
</tr>
</tbody>
</table>
Overview

<table>
<thead>
<tr>
<th>DFS technology</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tools for generating services and runtime support</td>
<td>Service-generation tools based on JAX-WS (Java API for XML-based Web Services) and Ant, which generate deployable DFS services from annotated source code. These tools also generate client-side runtime support for both Java clients. C# proxies are generated using the Dfs Proxy Generator utility.</td>
</tr>
<tr>
<td>SDK</td>
<td>A software development kit for development of DFS consumers, which includes the Java and .NET APIs, design-time build tools, and samples. The .NET APIs are CLS compliant, so they can be used to develop consumers using any .NET language (such as Visual Basic).</td>
</tr>
</tbody>
</table>

Service orientation

The design and technical implementation of DFS is grounded in the principles of Service-Oriented Architecture (SOA). Although an exploration of SOA concepts and principles is beyond the scope of this document, this section will summarize how DFS is designed to express well-accepted SOA principles.

One can define SOA in terms of its goals and function:

An architecture that provides for reuse of existing business services and rapid deployment of new business capabilities based on existing capital assets is often referred to as a service-oriented architecture (SOA). —Federal CIOs Council.

Or in terms of architectural principles:

The policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards-based form of interface. —CDBi Forum

DFS is designed around these goals and principles. The following is a brief list of some of the characteristics that express this design intent.

- DFS emphasizes service-orientation architecture, rather than web service technology. DFS remote service invocation is implemented using SOAP-based web services, but is designed to keep transport and messaging functionality orthogonal to other aspects of the DFS runtime, which provides for agility in regard to SOA implementation technology as DFS evolves. Web services standards, which are largely mature and well-accepted, provide a language- and platform-neutral layer for transport and messaging (SOAP), a well-accepted standard for expressing a service contract (WSDL), as well as a set of standards that are now widely accepted across a broad range of SOA functionality (such as WS-Security). DFS services are currently available not only as web services, but also as Java services that can be invoked locally.

- DFS enables preservation of capital assets by allowing services to be developed from existing SBOs (Service-based Business Objects), which belong to the Documentum Business Object Framework (BOF), as well as integration with standard frameworks using services developed from POJOS (Plain Old Java Objects). Service development from SBOs provides a migration path from the Documentum 5.3 Web Services Framework.

- Documentum Foundation Services are designed with the intent of interacting at an appropriate level of granularity with business processes implemented in service-oriented ECM consumers.
This is invariably a significantly coarser level of granularity than exhibited in a tightly bound API such as DFC. The level of granularity is achieved in part by consolidating functions that are implemented in numerous interdependent methods in the tightly bound API into a single service operation that addresses a conceptually singular business concern. For example, the update operation of the Object service concerns all aspects of updating a repository object, including modifying its properties, content, and relationships. In a tightly bound API, this business concern is addressed by a number of discrete, interdependent methods.

- The DFS data model, which is expressed primarily in the service XML schemas, as well as in the Java client library classes, provides a consistent, service-oriented approach to modeling data exchanged in ECM business processes. The DFS data model is designed with the intent of permitting arbitrarily sized, complex data packages to be passed in a payload to and from DFS services. This allows optimization of the payload size and minimization of costly service interactions with the consumer. The data model also supports loose-coupling by allowing a client to obtain complex data from a service, then cache and process the data independent of connection with the service. This is achieved by the Object service, for example, by returning objects as disconnected data graphs, which represent sets of objects and their relationships in the repository. The principle is also expressed in the Schema service, which enables downloading of repository metadata to the client, where it can be used for decoupled validation.

- DFS accomplishes a similar consolidation and standardization in the runtime specification of the behavior of operations through the mechanism of profiles. DFS profiles provide a uniform, coarse-grained approach to service- and operation-level specification of processing options. Profiles can be passed to individual service operations or stored in a stateful service context, which contains options that have the scope and lifetime of a set of services invoked by an application.

Generally speaking, the design of DFS services and data model simplifies the process of enterprise application development by reducing the overall complexity of the API and aligning the semantics of both services and data objects to the needs of ECM business logic. This supports rapid, agile application development using business process orchestration tools (such as BPM), and facilitates integration of enterprise content management into a service-oriented enterprise (SOE).

**DFS SDK**

The DFS Software Development Kit (SDK) includes Java class libraries, .NET assemblies, tools, documentation, and samples that you can use to build DFS services that extend the delivered DFS services, or to build DFS consumers using the optional C# or Java client library.

**Setting up dfc.properties**

For local execution of DFS services using the Java client library, DFS uses the DFC client bundled in the DFS SDK. This DFC client is configured in a dfc.properties file that must be located on the project classpath (it is provided in emc-dfs-sdk-6.0/etc). At minimum, to run the Java DFS samples in local mode, you will need to provide a setting for the machine name or IP address of a connection broker. To run workflow samples, or any other services that require an SBO, you will need to provide a global registry username and password.
For remote execution of DFS services, DFS uses the DFC client bundled in emc-dfs.ear, which is deployed with Content Server, or on a standalone application server. In these cases, the minimum dfc.properties settings for connection broker and global registry are set during installation.

Setting up Java classpaths

The Java classpaths that you set up in your development environment for DFS will vary, depending on the type of development you are doing. For example, you may be:

- Developing Java consumers of the services delivered with DFS.
- Developing services that wrap or extend DFS delivered services.
- Developing services that are peers to DFS services.

For detailed instructions on how to set up your project classpaths to support these different development scenarios, refer to the online document emc-dfs-sdk-6.0/docs/devsetup/index.html.

Note that the emc-dfs-sdk-6.0/etc directory contains a number of configuration files that are referenced by the Java client runtime. This directory or a copy of it, should be included in your project classpath.

Public packages and namespaces

Table 2, page 18 lists the public packages that are available for use in developing DFS services and Java client consumers. All other DFS Java packages contained within the SDK libraries are implementation packages that concern DFS internals, and should not be used in developing DFS applications. Note that as a general rule, any package containing .impl is an internal implementation package and should not be used directly in application code.

Table 2. Public API Java packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.emc.documentum.fs.datamodel.core</td>
<td>Principal data model classes, such as DataObject, DataPackage, and ObjectIdentity.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.content</td>
<td>Data model classes that pertain to content, such as Content, FileContent, and ActivityInfo.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.context</td>
<td>Data model classes that pertain to service context, for example ServiceContext and Identity.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.profiles</td>
<td>Classes that pertain to profiles, for example Profile and ContentProfile.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.properties</td>
<td>Classes that pertain to properties, such as Property and ArrayProperty.</td>
</tr>
<tr>
<td>Package</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.schema</td>
<td>Classes that represent repository metadata used by the Schema service, such as RepositoryInfo and SchemaInfo.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.datamodel.core.bpm</td>
<td>Classes used by the Workflow service, such as ProcessInfo.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.rt</td>
<td>Classes used by the DFS runtime, at the root level—principally exception classes such as ServiceException.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.rt.annotations</td>
<td>The DFS service annotation classes, specifically DfsBofService and DfsPojoService.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.rt.context</td>
<td>Classes related to instantiation and use of service and service context by DFS runtime, such as ContextFactory and ServiceContext.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.rt.services</td>
<td>Runtime services that support core services, such as ContextRegistryService.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.services.core.client</td>
<td>Public interfaces for services included with DFS, such as IObjectService.</td>
</tr>
<tr>
<td>com.emc.documentum.fs.tools</td>
<td>Includes classes used in DFS design-time tools.</td>
</tr>
</tbody>
</table>

Table 3, page 19 lists the public .NET client library namespaces. All other DFS .NET namespaces contained within the SDK libraries are implementation namespaces that concern DFS internals, and should not be used in developing DFS applications. Note that as a general rule, any namespace containing .Impl is for internal implementation and should not be used directly in consumer application code.

**Table 3. Public .NET namespaces**

<table>
<thead>
<tr>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emc.Documentum.FS.DataModel.Core</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Bpm</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Content</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Context</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Profiles</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Properties</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Query</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Schema</td>
</tr>
<tr>
<td>Emc.Documentum.FS.DataModel.Core.Utils</td>
</tr>
<tr>
<td>Emc.Documentum.FS.Runtime</td>
</tr>
<tr>
<td>Emc.Documentum.FS.Runtime.Context</td>
</tr>
<tr>
<td>Emc.Documentum.FS.Runtime.Resources</td>
</tr>
</tbody>
</table>
Overview

Package
Emc.Documentum_FS_Runtime_Services
Emc.Documentum_FS_Services_Bpm
Emc.Documentum_FS_Services_Core
Emc.Documentum_FS_Services_Search

DFS consumers

Consumers (or clients: the terms are used in this manual interchangeably) of all DFS services can be developed either using the WSDL interface alone, or using client runtime library support. Support for both Java and C# clients are in the SDK.

WSDL-based consumer development

The primary interface to DFS services is the WSDL. All DFS services are implemented using standard web services technology, and as such can be accessed by web services consumers using standard tools. No proprietary software is required on the client for a service consumer developed using the WSDL to connect to and use a DFS service. DFS makes all service functionality *available* to WSDL-based consumers; however, there may be additional work to do to *access* this functionality, as a WSDL-based consumer will not have the convenience functionality provided by the Java client library.

Java client library consumers

The DFS SDK provides development and runtime support for Java consumers of DFS services. It provides a number of conveniences for development of DFS service consumers, including:

- Convenience methods and constructors.
- Transparent handling of exceptions passed in SOAP messages returned by the service. The consumer will display the stack trace of the exception as it would an exception thrown by an application running in the local JVM.
- Transparent invocation of UCF and handling of UCF content transfer.
- Simplified security and registration of service context.
- Location transparency: the ability to execute the service either remotely via web services or within the local application.

Location transparency

All services provided by DFS, as well as custom services that you develop, can be executed locally with the optional Java client runtime support, or remotely via SOAP. This capability greatly decreases
the cost of testing and debugging: a custom service can be completely tested in a local environment before it is deployed remotely and retested using remote execution. Local deployment may also be a useful option in some production scenarios.

### Configuring service addressing in Java

A DFS client library consumer can invoke a service using either explicit or implicit addressing (for more information see Service instantiation, page 59).

The SDK consumer samples typically use implicit addressing, and are dependent on local configuration settings, provided in dfs-client.xml.

Note that DFS when installed with Content Server is addressed at port 9080.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DFSClientConfig defaultModuleName="core" registryProviderModuleName="core">
  <ModuleInfo name="core"
    protocol="http"
    host="contentServerHost"
    port="9080"
    contextRoot="services">
    </ModuleInfo>
</DFSClientConfig>
```

### Running the Java consumer samples

The Java client consumer samples are located in C:\emc-dfs-sdk-6.0\samples\DfsSamples. The samples, which are identical to the samples shown throughout this document, are driven by a set of JUnit tests, which are included under C:\emc-dfs-sdk-6.0\samples\DfsSamples\test. JUnit is not provided in the SDK, but it can be obtained from [http://www.junit.org/](http://www.junit.org/).

Repository names and user credentials are set up as instance variables in DFSTestCase.java, as shown here:

```java
//TODO: You must supply valid values for the following fields:
private String repository = "yourRepositoryName";
private String userName = "yourUserName";
private String password = "yourPassword";
```

The user under whose credentials the samples run should be privileged to create cabinets in the repository.

For service addressing, the samples are dependent on dfs-client.xml (see Configuring service addressing in Java, page 21) and instantiate services using implicit service addressing (see Service instantiation, page 59).

The JUnit tests set up and tear down sample content on your repository before and after running each test. The samples for the most part are created in a cabinet called DFSTestCabinet. The name of this cabinet, as well as other static variables related to the samples, are encapsulated by the SampleContentManager class. If you need to avoid conflicts, you may want to change the name of DFSTestCabinet to something more likely to remain unique.
The variable SampleContentManager.isDataCleanedUp determines whether the tests remove the sample data after each test. Setting this to false will enable you to examine data that is created in the repository by the samples. However, be aware that running multiple samples with this variable set to false will lead to exceptions caused by duplicate file names.

.NET client library consumers

The .NET (C#) client library offers support functionally identical to the Java client library, with the exception of local service invocation: the C# client library supports only remote service invocation. The .NET client provides the following conveniences for development of DFS service consumers:

- Convenience methods and constructors.
- Transparent handling of exceptions passed in SOAP messages returned by the service.
- Transparent invocation of UCF and handling of UCF content transfer.
- Simplified security and registration of service context.

.NET consumer project dependencies

The DFS .NET client library requires .NET 3.0, which includes the Windows Communication Foundation (WCF), Microsoft’s unified framework for creating service-oriented applications. For more information see http://msdn2.microsoft.com/en-us/library/ms735119.aspx.

DFS consumer projects will require references to the following assemblies from the DFS SDK:

- Emc.Documentum.FS.DataModel
- Emc.Documentum.FS.Runtime
- Emc.Documentum.FS.Services

.NET client configuration

.NET client configuration settings are specified in the consumer application’s app.config file, which is shown below. These settings are loaded at runtime if the app.config file is present in application’s working directory during startup.

The configuration settings include ContextRoot and Module settings used in implicit service addressing (see Service instantiation in C#, page 59), as well as other settings that are described in Table 4, page 23.

```xml
<configuration>
  <configSections>
    <sectionGroup name="Emc.Documentum">
      <sectionGroup name="FS">
        <section name="ConfigObject"
          type="Emc.Documentum.FS.Runtime.Impl.Configuration.XmlSerializerSectionHandler,
             Emc.Documentum.FS.Runtime"/>
      </sectionGroup>
    </sectionGroup>
  </configSections>
</configuration>
```
The following table describes the settings that are configurable using app.config.

**Table 4. app.config settings**

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bypassProxyOnLocal(optional)</td>
<td>A Boolean value that indicates whether to bypass the proxy server when resources are located at a local address. The default is false. For more information see <a href="http://msdn2.microsoft.com/en-us/library/ms731361.aspx">http://msdn2.microsoft.com/en-us/library/ms731361.aspx</a>.</td>
</tr>
<tr>
<td>closeTimeout (optional)</td>
<td>A TimeSpan value specifying the amount of time allowed for a close operation to complete. This value should be greater than or equal to 0. The default is 05:00:00 minutes.</td>
</tr>
<tr>
<td>openTimeout (optional)</td>
<td>A TimeSpan value specifying the amount of time allowed for an open operation to complete. This value should be greater than or equal to 0. The default is 05:00:00 minutes.</td>
</tr>
<tr>
<td>receiveTimeout (optional)</td>
<td>A TimeSpan value specifying the amount of time allowed for a receive operation to complete. This value should be greater than or equal to 0. The default is 05:00:00 minutes.</td>
</tr>
<tr>
<td>sendTimeout (optional)</td>
<td>A TimeSpan value specifying the amount of time allowed for a send operation to complete. This value should be greater than or equal to 0. The default is 05:00:00 minutes.</td>
</tr>
<tr>
<td>useDefaultWebProxy (optional)</td>
<td>A Boolean value that determines whether to use the auto-configured HTTP proxy, if one is available. The default is true.</td>
</tr>
<tr>
<td>proxyAddress (optional)</td>
<td>A URI that contains the address of the HTTP proxy. If useDefaultWebProxy is set to true, this setting must be null. The default is null.</td>
</tr>
</tbody>
</table>
Running the C# consumer samples

The C# documentation samples that you see in this manual are provided in the SDK as two projects, DotNetDocSamples, which contains the documentation samples proper, and DotNetDocSamplesTest, which includes a rudimentary NUnit test framework that can be used to run the samples. The test framework takes care of application configuration, as well as sample data creation and cleanup. NUnit is not provided in the SDK, but it can be obtained from http://www.nunit.org/.

Repository names and user credentials are set up as instance variables in DemoBase.cs, as shown here:

```csharp
// TODO: You must supply valid values for the following fields:
private string defaultDocbase = "yourRepositoryName";
private string secondaryDocbase = "yourSecondaryRepositoryName";
private string userName = "yourUserName";
private string password = "yourPassword";
```

The user under whose credentials the samples run should be privileged to create cabinets in the repository.

For service addressing, the samples are dependent on app.config (see .NET client configuration, page 22) and instantiate services using implicit service addressing (see Service instantiation in C#, page 59).

The NUnit tests set up and tear down sample content on your repository before and after running each test. The samples for the most part are created in a cabinet called DFSTestCabinet. The name of this cabinet, as well as other static variables related to the samples, are encapsulated by the SampleContentManager class. If you need to avoid conflicts, you may want to change the name of DFSTestCabinet to something more likely to remain unique.

The IsDataCleanedUp property of a SampleContentManager instance determines whether the tests remove the sample data after each test. Setting this to false will enable you to examine data that is created in the repository by the samples. However, be aware that running multiple samples with this property set to false will lead to exceptions caused by duplicate file names.

Enterprise Content Services

Enterprise Content Services (ECS), which includes all services that operate within the DFS framework, share a common set of technologies built around JAX-WS, including a service context, use of the
ContextRegistry and Agent DFS runtime services, and common runtime classes. DFS delivers a set of core Enterprise Content Services, which are deployed with Content Server, where they are hosted by the Java Method Server. (These services can also be deployed in a standalone or clustered configuration using a separate installation. For details, see the Documentum Foundation Services Installation Guide.) The services provided with DFS can be extended with additional Enterprise Content Services provided by EMC, partners, and customers.

### Service development and generation tools

The DFS SDK includes a set of tools for developing and deploying custom Enterprise Content Services. These services can be implemented as POJOs (Plain Old Java Objects), or as BOF (Business Object Framework) service-based business objects (SBOs). The tools rely on JAX-WS and JAXB, which require annotations of the original service source code, and on Ant tasks, which build service artifacts that are archived into a deployable EAR file for remote execution, and into JAR files to enable local execution using the optional client runtime. C# client-side proxies are generated using the DFS Proxy Generator utility. For further information, including a sample service, see Chapter 11, Building Custom Services Using DFS.
DFS Data Model

The DFS data model comprises the object model for data passed to and returned by Enterprise Content Services. This chapter covers the following topics:

- DataPackage, page 27
- DataObject, page 28
- ObjectIdentity, page 30
- Property, page 33
- Content, page 40
- Permissions, page 42
- Relationship, page 44
- Other classes related to DataObject, page 53

DataPackage

The DataPackage class defines the fundamental unit of information that contains data passed to and returned by services operating in the DFS framework. A DataPackage is a collection of DataObject instances, which is typically passed to, and returned by, Object service operations such as create, get, and update. Object service operations process all the DataObject instances in the DataPackage sequentially.

Example

The following sample instantiates, populates, and iterates through a data package:

Example 2-1. Java: DataPackage

```java
DataObject dataObject = new DataObject(new ObjectIdentity("myRepository"));
DataPackage dataPackage = new DataPackage(dataObject);

DataObject dataObject1 = new DataObject(new ObjectIdentity("myRepository"));
dataPackage.addDataObject(dataObject1);

ArrayList<DataObject> dataObjectList = new ArrayList<DataObject>();
```
dataObjectList.add(dataObject);
dataObjectList.add(dataObject1);
dataPackage.setDataObjects(dataObjectList);

for (DataObject dataObject2 : dataPackage.getDataObjects())
{
    System.out.println("Data Object: " + dataObject2);
}

Example 2-2. C#: DataPackage

DataObject dataObject = new DataObject(new ObjectIdentity("myRepository").
DataPackage dataPackage = new DataPackage(dataObject);

DataObject dataObject1 = new DataObject(new ObjectIdentity("myRepository").
dataPackage.addDataObject(dataObject1);

foreach (DataObject dataObject2 in dataPackage.DataObjects)
{
    Console.WriteLine("Data Object: " + dataObject2);
}

DataObject

A DataObject is a representation of an object in an ECM repository. In the context of EMC Documentum technology, the DataObject functions as a DFS representation of a persistent repository object, such as a dm_sysobject or dm_user. Enterprise Content Services (such as the Object service) consistently process DataObject instances as representations of persistent repository objects.

A DataObject instance is potentially large and complex, and much of the work in DFS service consumers will be dedicated to constructing the DataObject instances. A DataObject can potentially contain comprehensive information about the repository object that it represents, including its identity, properties, content, and its relationships to other repository objects. In addition, the DataObject instance may contain settings that instruct the services about how the client wishes parts of the DataObject to be processed. The complexity of the DataObject and related parts of the data model, such as Profile classes, are design features that enable and encourage simplicity of the service interface and the packaging of complex consumer requests into a minimal number of service interactions.

For the same reason DataObject instances are consistently passed to and returned by services in simple collections defined by the DataPackage class, permitting processing of multiple DataObject instances in a single service interaction.

DataObject related classes

Table 5, page 29 shows the object types that can be contained by a DataObject.
Table 5. DataObject related classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectIdentity</td>
<td>An ObjectIdentity uniquely identifies the repository object referenced by the DataObject. A DataObject can have 0 or 1 identities. For more details see ObjectIdentity, page 30.</td>
</tr>
<tr>
<td>PropertySet</td>
<td>A PropertySet is a collection of named properties, which correspond to the properties of a repository object represented by the DataObject. A DataObject can have 0 or 1 PropertySet instances. For more information see Property, page 33.</td>
</tr>
<tr>
<td>Content</td>
<td>Content objects contain data about file content associated with the data object. A DataObject can contain 0 or more Content instances. A DataObject without content is referred to as a &quot;contentless DataObject.&quot; For more information see Content, page 40.</td>
</tr>
<tr>
<td>Permission</td>
<td>A Permission object specifies a specific basic or extended permission, or a custom permission. A DataObject can contain 0 or more Permission objects. For more information see Permissions, page 42.</td>
</tr>
<tr>
<td>Relationship</td>
<td>A Relationship object defines a relationship between the repository object represented by the DataObject and another repository object. A DataObject can contain 0 or more Relationship instances. For more information, see Relationship, page 44.</td>
</tr>
</tbody>
</table>

**DataObject type**

A DataObject instance in normal DFS usage corresponds to a typed object defined in the repository. The type is specified in the type setting of the DataObject using the type name defined in the repository (for example dm_sysobject or dm_user). If the type is not specified, services will use an implied type, which is dm_document.

**DataObject construction**

The construction of DataObject instances will be a constant theme in examples of service usage throughout this document. The following typical example instantiates a DataObject, sets some of its properties, and assigns it some content. Note that because this is a new DataObject, only a repository name is specified in its ObjectIdentity.

*Example 2-3. Java: DataObject construction*

```java
ObjectIdentity objIdentity = new ObjectIdentity(repositoryName);
DataObject dataObject = new DataObject(objIdentity, "dm_document");

PropertySet properties = dataObject.getProperties();
properties.set("object_name", objName);
properties.set("title", objTitle);
properties.set("a_content_type", "gif");
```
`dataObject.getContents().add(new FileContent("c:/temp/MyImage.gif", "gif"));

DataPackage dataPackage = new DataPackage(dataObject);

**Example 2.4. C#: DataObject construction**

```csharp
ObjectIdentity objIdentity = new ObjectIdentity(repositoryName);
DataObject dataObject = new DataObject(objIdentity, "dm_document");

PropertySet properties = dataObject.Properties;
properties.Set("object_name", objName);
properties.Set("title", objTitle);
properties.Set("a_content_type", "gif");

dataObject.Contents.Add(new FileContent("c:/temp/MyImage.gif", "gif"));

DataPackage dataPackage = new DataPackage(dataObject);
```

**ObjectId**

The function of the ObjectIdentity class is to uniquely identify a repository object. An ObjectIdentity instance contains a repository name and an identifier that can take various forms, described in the following table listing the ValueType enum constants.

<table>
<thead>
<tr>
<th>ValueType</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OBJECT_ID</td>
<td>Identifier value is of type ObjectId, which is a container for the value of a repository r_object_id attribute, a value generated by Content Server to uniquely identify a specific version of a repository object.</td>
</tr>
<tr>
<td>OBJECT_PATH</td>
<td>Identifier value is of type ObjectPath, which contains a String expression specifying the path to the object, excluding the repository name. For example /MyCabinet/MyFolder/MyDocument.</td>
</tr>
<tr>
<td>QUALIFICATION</td>
<td>Identifier value is of type Qualification, which can take the form of a DQL expression fragment. The Qualification is intended to uniquely identify a Content Server object.</td>
</tr>
</tbody>
</table>

When constructing a DataObject to pass to the create operation, or in any case when the DataObject represents a repository object that does not yet exist, the ObjectIdentity need only be populated with a repository name. If the ObjectIdentity does contain a unique identifier, it must represent an existing repository object.

Note that the ObjectIdentity class is generic in the Java client library, but non-generic in the .NET client library.

**ObjectId**

An ObjectId is a container for the value of a repository r_object_id attribute, which is a value generated by Content Server to uniquely identify a specific version of a repository object. An ObjectId can therefore represent either a CURRENT or a non-CURRENT version of a repository object. DFS services exhibit service- and operation-specific behaviors for handling non-CURRENT versions, which are documented under individual services and operations.
ObjectPath

An ObjectPath contains a String expression specifying the path to a repository object, excluding the repository name. For example /MyCabinet/MyFolder/MyDocument. An ObjectPath can only represent the CURRENT version of a repository object. Using an ObjectPath does not guarantee the uniqueness of the repository object, because Content Server does permit objects with identical names to reside within the same folder. If the specified path is unique at request time, the path is recognized as a valid object identity; otherwise, the DFS runtime will throw an exception.

Qualification

A Qualification is an object that specifies criteria for selecting a set of repository objects. Qualifications used in ObjectIdentity instances are intended to specify a single repository object. The criteria set in the qualification is expressed as a fragment of a DQL SELECT statement, consisting of the expression string following "SELECT FROM", as shown in the following example.

```java
Qualification qualification =
    new Qualification("dm_document where object_name = 'dfs_sample_image'");
```

DFS services use normal DQL statement processing, which selects the CURRENT version of an object if the ALL keyword is not used in the DQL WHERE clause. The preceding example (which assumes for simplicity that the object_name is sufficient to ensure uniqueness) will select only the CURRENT version of the object named dfs_sample_image. To select a specific non-CURRENT version, the Qualification must use the ALL keyword, as well as specific criteria for identifying the version, such as a symbolic version label:

```java
String nonCurrentQual = "dm_document (ALL) " +
    "where object_name = 'DFS_sample_image' " +
    "and ANY r_version_label = 'test_version';
Qualification<String> qual = new Qualification<String>(nonCurrentQual);
```

Example

The following samples demonstrate the ObjectIdentity subtypes.

Example 2-5. Java: ObjectIdentity subtypes

```java
String repName = "MyRepositoryName";
ObjectIdentity[] objectIds = new ObjectIdentity[4];

// repository only is required to represent an object that has not been created
objectIds[0] = new ObjectIdentity(repName);

// show each form of unique identifier
ObjectId objId = new ObjectId("0900007d280075180");
objectIds[1] = new ObjectIdentity<ObjectId>(objId, repName);

Qualification qualification
    = new Qualification("dm_document where r_object_id = '0900007d280075180'");
objectIds[2] = new ObjectIdentity<ObjectId>(qualification, repName);

ObjectPath objPath = new ObjectPath("/testCabinet/testFolder/testDoc");
objectIds[3] = new ObjectIdentity<ObjectPath>(objPath, repName);
```
for (ObjectIdentity identity : objectIdenties)
{
    System.out.println(identity.getValueAsString());
}

**Example 2-6. C#: ObjectIdentity subtypes**

```csharp
String repName = "MyRepositoryName";
ObjectIdentity[] objectIdenties = new ObjectIdentity[4];

// repository only is required to represent an object that has not been created
objectIdenties[0] = new ObjectIdentity(repName);

// show each form of unique identifier
ObjectId objId = new ObjectId("090007d280075180");
objectIdenties[1] = new ObjectIdentity(objId, repName);
Qualification qualification =
    new Qualification("dm_document where r_object_id = '090007d280075180'");
objectIdenties[2] = new ObjectIdentity(qualification, repName);
ObjectPath objPath = new ObjectPath("/testCabinet/testFolder/testDoc");
objectIdenties[3] = new ObjectIdentity(objPath, repName);

foreach (ObjectIdentity identity in objectIdenties)
{
    Console.WriteLine(identity.GetValueAsString());
}
```

**ObjectIdentitySet**

An ObjectIdentitySet is a collection of ObjectIdentity instances, which can be passed to an Object service operation so that it can process multiple repository objects in a single service interaction. An ObjectIdentitySet is analogous to a DataPackage, but is passed to service operations such as move, copy, and delete that operate only against existing repository data, and which therefore do not require any data from the consumer about the repository objects other than their identity.

**Example**

The following code sample creates and populates an ObjectIdentitySet.

**Example 2-7. Java: ObjectIdentitySet**

```java
String repName = "MyRepositoryName";
ObjectIdentitySet objIdSet = new ObjectIdentitySet();
ObjectIdentity[] objectIdenties = new ObjectIdentity[4];

// add some ObjectIdentity instances
ObjectId objId = new ObjectId("090007d280075180");
objIdSet.addIdentity(new ObjectIdentity(objId, repName));
Qualification qualification =
    new Qualification("dm_document where object_name = 'bl_upwind.gif'";
objIdSet.addIdentity(new ObjectIdentity(qualification, repName));
ObjectPath objPath = new ObjectPath("/testCabinet/testFolder/testDoc");
```
objIdSet.addIdentity(new ObjectIdentity(objPath, repName));

// walk through and see what we have
Iterator iterator = objIdSet.getIdentities().iterator();
while (iterator.hasNext())
{
    System.out.println("Object Identity: " + iterator.next());
}

Example 2-8. C#: ObjectIdentitySet

String repName = "MyRepositoryName";
ObjectIdentitySet objIdSet = new ObjectIdentitySet();
ObjectIdentity[] objectIdentities = new ObjectIdentity[4];

// add some ObjectIdentity instances
ObjectId objId = new ObjectId("090007d280075180");
objIdSet.AddIdentity(new ObjectIdentity(objId, repName));

Qualification qualification
    = new Qualification("dm_document where object_name = 'bl_upwind.gif'");
objIdSet.AddIdentity(new ObjectIdentity(qualification, repName));

ObjectPath objPath = new ObjectPath("/testCabinet/testFolder/testDoc");
objIdSet.AddIdentity(new ObjectIdentity(objPath, repName));

// walk through and see what we have
IEnumerator<ObjectIdentity> identityEnumerator = objIdSet.Identities.GetEnumerator();
while (identityEnumerator.MoveNext())
{
    Console.WriteLine("Object Identity: " + identityEnumerator.Current);
}

Property

A DataObject optionally contains a PropertySet, which is a container for a set of Property objects. Each Property in normal usage corresponds to a property (also called attribute) of a repository object represented by the DataObject. A Property object can represent a single property, or an array of properties of the same data type. Property arrays are represented by subclasses of ArrayProperty, and correspond to repeating attributes of repository objects.

Property model

The Property class is subclassed by data type (for example StringProperty), and each subtype has a corresponding class containing an array of the same data type, extending the intermediate abstract class ArrayProperty (see Figure 1, page 34).
**Figure 1. Property class hierarchy**

```
Example

The following sample shows instantiation of the various Property subtypes.

Property[] properties =
{
    new StringProperty("subject", "dangers"),
    new StringProperty("title", "Dangers"),
    new NumberProperty("short", (short) 1),
    new DateProperty("my_date", new Date()),
    new BooleanProperty("a_full_text", true),
    new ObjectIdProperty("my_object_id", new ObjectId("090007d280075180")),
    new StringArrayProperty("keywords", new String[] {"lions", "tigers", "bears"}),
    new NumberArrayProperty("my_number_array", (short) 1, 10, 100L, 10.10),
    new BooleanArrayProperty("my_boolean_array", true, false, true, false),
    new DateArrayProperty("my_date_array", new Date(), new Date()),
    new ObjectIdArrayProperty("my_obj_id_array",
        new ObjectId("0c0007d280000107"), new ObjectId("090007d280075180")),
}
```

**Transient properties**

Transient properties are custom Property objects that are not interpreted by the services as representations of persistent properties of repository objects. You can therefore use transient properties to pass your own data fields to a service to be used for a purpose other than setting attributes on repository objects.

To indicate that a Property is transient, set the isTransient field of the Property object to true.

One intended application of transient properties implemented by the services is to provide the client the ability to uniquely identify DataObject instances passed in a validate operation, when the instances have not been assigned a unique ObjectIdentity. The validate operation returns a ValidationInfoSet.
field, which contains information about any DataObject instances that failed validation. If the service client has populated a transient property of each DataObject with a unique identifier, the client will be able to determine which DataObject failed validation by examining the ValidationInfoSet.

For more information see validate operation, page 87.

**Example**

The following sample would catch a ValidationException and print a custom id property for each failed DataObject to the console.

**Example 2-9. Java: Transient properties**

```java
public void showTransient(ValidationInfoSet infoSet) {
    List<ValidationInfo> failedItems = infoSet.getValidationInfos();
    for (ValidationInfo vInfo : failedItems) {
        System.out.println(vInfo.getDataObject()
            .getProperties()
            .get("my_unique_id"));
    }
}
```

**Example 2-10. C#: Transient properties**

```csharp
public void ShowTransient(ValidationInfoSet infoSet) {
    List<ValidationInfo> failedItems = infoSet.ValidationInfos;
    foreach (ValidationInfo vInfo in failedItems) {
    }
}
```

**Loading properties: convenience API**

As a convenience the Java client library will determine at runtime the correct property subclass to instantiate based on the data type passed to the Property constructor. For example, the following code adds instances of NumberProperty, DateProperty, BooleanProperty, and ObjectIdProperty to a PropertySet:

**Example 2-11. Java: Loading properties**

```java
PropertySet propertySet = new PropertySet();

//Create instances of NumberProperty
propertySet.set("TestShortName", (short) 10);
propertySet.set("TestIntegerName", 10);
propertySet.set("TestLongName", 10L);
propertySet.set("TestDoubleName", 10.10);

//Create instance of DateProperty
propertySet.set("TestPropertyName", new Date());
```
DFS Data Model

//Create instance of BooleanProperty
propertySet.set("TestBooleanName", false);

//Create instance of ObjectIdProperty
propertySet.set("TestObjectIdName", new ObjectId("10"));

Iterator items = propertySet.iterator();
while (items.hasNext())
{
    Property property = (Property) items.next();
    System.out.println(property.getClass().getName() + 
                        " = " + property.getValueAsString());
}

Example 2-12. C#: Loading properties

PropertySet propertySet = new PropertySet();

//Create instances of NumberProperty
propertySet.Set("TestShortName", (short)10);
propertySet.Set("TestIntegerName", 10);
propertySet.Set("TestLongName", 10L);
propertySet.Set("TestDoubleName", 10.10);

//Create instance of DateProperty
propertySet.Set("TestDateName", new DateTime());

//Create instance of BooleanProperty
propertySet.Set("TestBooleanName", false);

//Create instance of ObjectIdProperty
propertySet.Set("TestObjectIdName", new ObjectId("10"));

List<Property> properties = propertySet.Properties;
foreach (Property p in properties)
{
    Console.WriteLine(typeof(Property).ToString() + 
                        " = " + p.GetValueAsString());
}

The NumberProperty class stores its value as a java.lang.Number, which will be instantiated as a concrete numeric type such as Short or Long. Setting this value unambiguously, as demonstrated in the preceding sample code (for example 10L or (short)10), determines how the value will be serialized in the XML instance and received by a service. The following schema shows the numeric types that can be serialized as a NumberProperty:

<xs:complexType name="NumberProperty">
    <xs:complexContent>
        <xs:extension base="xscp:Property">
            <xs:sequence>
                <xs:choice minOccurs="0">
                    <xs:element name="Short" type="xs:short"/>
                    <xs:element name="Integer" type="xs:int"/>
                    <xs:element name="Long" type="xs:long"/>
                    <xs:element name="Double" type="xs:double"/>
                </xs:choice>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
ArrayProperty

The subclasses of ArrayProperty each contain an array of Property objects of a specific subclass corresponding to a data type. For example, the NumberArrayProperty class contains an array of NumberProperty. The array corresponds to a repeating attribute (also known as repeating property) of a repository object.

ValueAction

Each ArrayProperty optionally contains an array of ValueAction objects that contain an ActionType-index pair (see Figure 2, page 37). These pairs can be interpreted by the service as instructions for using the data stored in the ArrayProperty to modify the repeating attribute of the persistent repository object. The ValueAction array is synchronized to the ArrayProperty array, such that any position p of the ValueAction array corresponds to position p of the ArrayProperty. The index in each ActionType-index pair is zero-based and indicates a position in the repeating attribute of the persistent repository object. ValueActionType specifies how to modify the repeating attribute list using the data stored in the ArrayProperty.

Figure 2. ArrayProperty model

The following table describes how the ValueActionType values are interpreted by an update operation.
<table>
<thead>
<tr>
<th>Value type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>When processing ValueAction[p], the value at ArrayProperty[p] is appended to the end of repeating properties list of the persistent repository object. The index of the ValueAction item is ignored.</td>
</tr>
<tr>
<td>INSERT</td>
<td>When processing ValueAction[p], the value at ArrayProperty[p] is inserted into the repeating attribute list before position index. Note that all items in the list to the right of the insertion point are offset by 1, which must be accounted for in subsequent processing.</td>
</tr>
<tr>
<td>DELETE</td>
<td>The item at position index of the repeating attribute is deleted. When processing ValueAction[p] the value at ArrayProperty[p] must be set to a empty value (see Deleting a repeating property: use of empty value, page 38). Note that all items in the list to the right of the insertion point are offset by -1, which must be accounted for in subsequent processing.</td>
</tr>
<tr>
<td>SET</td>
<td>When processing ValueAction[p], the value at ArrayProperty[p] replaces the value in the repeating attribute list at position index.</td>
</tr>
</tbody>
</table>

Note in the preceding description of processing that the INSERT and DELETE actions will offset index positions to the right of the alteration, as the ValueAction array is processed from beginning to end. These effects must be accounted in the coding of the ValueAction object, such as by ensuring that the repeating properties list is processed from right to left.

For more information see Modifying a repeating properties (attributes) list, page 75.

Deleting a repeating property: use of empty value

When using a ValueAction to delete a repeating attribute value, the value stored at position ArrayProperty[p], corresponding to ValueAction[p] is not relevant to the operation. However, the two arrays must still line up. In this case, you should store an empty (dummy) value in ArrayProperty[p] (such as the empty string ""), rather than null.

PropertySet

A PropertySet is a container for named Property objects, which typically (but do not necessarily) correspond to persistent repository object properties.

You can restrict the size of a PropertySet returned by a service using the filtering mechanism of the PropertyProfile class (see PropertyProfile, page 39).

Example

Example 2-13. Java: PropertySet

Property[] properties =
{
    new StringProperty("subject", "dangers"),
    new StringProperty("title", "Dangers"),
    new StringArrayProperty("keywords",
PropertyProfile

A PropertyProfile defines property filters that limit the properties returned with an object by a service. This allows you to optimize the service by returning only those properties that your service consumer requires. PropertyProfile, like other profiles, is generally set in the OperationOptions passed to a service operation (or it can be set in the service context).

You specify how PropertyProfile filters returned properties by setting its PropertyFilterMode. The following table describes the PropertyProfile filter settings:

<table>
<thead>
<tr>
<th>PropertyFilterMode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NONE</td>
<td>No properties are returned in the PropertySet. Other settings are ignored.</td>
</tr>
<tr>
<td>SPECIFIED_BY_INCLUDE</td>
<td>No properties are returned unless specified in the includeProperties list.</td>
</tr>
<tr>
<td>SPECIFIED_BY_EXCLUDE</td>
<td>All properties are returned unless specified in the excludeProperties list.</td>
</tr>
<tr>
<td>ALL_NON_SYSTEM</td>
<td>Returns all properties except system properties.</td>
</tr>
<tr>
<td>ALL</td>
<td>All properties are returned.</td>
</tr>
</tbody>
</table>

Example

The following samples add a PropertyProfile to the operationOptions argument to be passed to an operation. The PropertyProfile will instruct the service to include only specified properties in the PropertySet of each returned DataObject.
Example 2-15. Java: PropertyProfile

```java
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.setFilterMode(PropertyFilterMode.SPECIFIED_BY_INCLUDE);
ArrayList<String> includeProperties = new ArrayList<String>();
includeProperties.add("title");
includeProperties.add("object_name");
includeProperties.add("r_object_type");
propertyProfile.setIncludeProperties(includeProperties);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setPropertyProfile(propertyProfile);
```

Example 2-16. C#: PropertyProfile

```csharp
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.FilterMode = PropertyFilterMode.SPECIFIED_BY_INCLUDE;
List<string> includeProperties = new List<string>();
includeProperties.Add("title");
includeProperties.Add("object_name");
includeProperties.Add("r_object_type");
propertyProfile.IncludeProperties = includeProperties;
OperationOptions operationOptions = new OperationOptions();
operationOptions.PropertyProfile = propertyProfile;
```

Content

File content in a DataObject is represented by an instance of a subtype of the Content class (such as FileContent). The Content subtypes support multiple types of input to services and multiple content transfer options, including use of UCF content transfer, Java DataHandler objects, and byte arrays. A Content object can be configured to represent a complete document, a page in a document, or a set of pages in a document identified by a characteristic represented by a pageModifier string.

A DataObject contains a list of zero or more Content instances, which are identified as either primary content or a rendition by examining their RenditionType. A repository object can have only one primary content object and zero or more renditions.

For information on content and content transfer, see Chapter 10, Content and Content Transfer.

ContentProfile

The ContentProfile class enables a client to set filters that control the content returned by a service. This has important ramifications for service performance, because it permits fine control over expensive content transfer operations.

ContentProfile includes three types of filters: FormatFilter, PageFilter, and PageModifierFilter. For each of these filters there is a corresponding variable that is used or ignored depending on the filter settings. For example, if the FormatFilter value is FormatFilter.SPECIFIED, the service will return content that has a format specified by the ContentProfile.format field. Each field corresponds to a setting in the dmr_content object that represents the content in the repository.

The following table describes the ContentProfile filter settings:
<table>
<thead>
<tr>
<th>Value type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FormatFilter</td>
<td>NONE</td>
<td>No content is included. All other filters are ignored.</td>
</tr>
<tr>
<td></td>
<td>SPECIFIED</td>
<td>Return only content specified by the format setting. The format field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>corresponds to the name of a dm_format object installed in the repository.</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>Return content in any format, ignoring format setting.</td>
</tr>
<tr>
<td>PageFilter</td>
<td>SPECIFIED</td>
<td>Return only page number specified by pageNumber setting. The pageNumber</td>
</tr>
<tr>
<td></td>
<td></td>
<td>field corresponds to the dmr_content.page property in the repository for</td>
</tr>
<tr>
<td></td>
<td></td>
<td>content objects that have multiple pages.</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>Ignore pageNumber setting.</td>
</tr>
<tr>
<td>PageModifierFilter</td>
<td>SPECIFIED</td>
<td>Return only page number with specified pageModifier. The pageModifier</td>
</tr>
<tr>
<td></td>
<td></td>
<td>field corresponds to the dmr_content.page_modifier property in the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>repository. This setting is used to distinguish different renditions of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>an object that have the same format (for example, different resolution</td>
</tr>
<tr>
<td></td>
<td></td>
<td>settings for images or sound recordings).</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>Ignore pageModifier setting.</td>
</tr>
</tbody>
</table>

Note that you can use the following DQL to get a list of all format names stored in a repository:

```
SELECT "name", "description" FROM "dm_format"
```

You can use this or a similar query to retrieve and cache format information using the Query service for lookup or validation. For more information see Chapter 7, Query Service.

**PostTransferAction**

You can set the PostTransferAction of a ContentProfile instance to open a transferred document in an application for viewing or editing. For information see Opening a transferred document in a viewer/editor, page 176.

**Example**

The following sample sets a ContentProfile in operationOptions. The ContentProfile will instruct the service to exclude all content from each returned DataObject.

```java
ContentProfile contentProfile = new ContentProfile();
contentProfile.setFormatFilter(FormatFilter.ANY);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setContentProfile(contentProfile);
```
Permissions

A DataObject contains a list of Permission objects, which together represent the permissions of the user who has logged into the repository on the repository object represented by the DataObject. The intent of the Permission list is to provide the client with read access to the current user’s permissions on a repository object. The client cannot set or update permissions on a repository object by modifying the Permission list and updating the DataObject. To actually change the permissions, the client would need to modify or replace the repository object’s permission set (also called an Access Control List, or ACL).

Each Permission has a permissionType field can be set to BASIC, EXTENDED, or CUSTOM. BASIC permissions are compound (sometimes called hierarchical), meaning that there are levels of permission, with each level including all lower-level permissions. For example, if a user has RELATE permissions on an object, the user is also granted READ and BROWSE permissions. This principle does not apply to extended permissions, which have to be granted individually.

The following table shows the PermissionType enum constants and Permission constants:

<table>
<thead>
<tr>
<th>Permission type</th>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BASIC</td>
<td>NONE</td>
<td>No access is permitted.</td>
</tr>
<tr>
<td></td>
<td>BROWSE</td>
<td>The user can view attribute values of content.</td>
</tr>
<tr>
<td></td>
<td>READ</td>
<td>The user can read content but not update.</td>
</tr>
<tr>
<td></td>
<td>RELATE</td>
<td>The user can attach an annotation to object.</td>
</tr>
<tr>
<td></td>
<td>VERSION</td>
<td>The user can version the object.</td>
</tr>
<tr>
<td></td>
<td>WRITE</td>
<td>The user can write and update the object.</td>
</tr>
<tr>
<td></td>
<td>DELETE</td>
<td>The user can delete the object.</td>
</tr>
<tr>
<td>EXTENDED</td>
<td>X_CHANGE_LOCATION</td>
<td>The user can change move an object from one folder to another. All users having at least Browse permission on an object are granted Change Location permission by default for that object.</td>
</tr>
<tr>
<td></td>
<td>X_CHANGE_OWNER</td>
<td>The user can change the owner of the object.</td>
</tr>
<tr>
<td></td>
<td>X_CHANGE_PERMIT</td>
<td>The user can change the basic permissions on the object.</td>
</tr>
<tr>
<td></td>
<td>X_CHANGE_STATE</td>
<td>The user can change the document lifecycle state of the object.</td>
</tr>
<tr>
<td></td>
<td>X_DELETE_OBJECT</td>
<td>The user can delete the object. The delete object extended permission is not equivalent to the base Delete permission. Delete Object extended permission does not grant Browse, Read, Relate, Version, or Write permission.</td>
</tr>
<tr>
<td></td>
<td>X_EXECUTE_PROC</td>
<td>The user can run the external procedure associated with the object. All users having at least Browse permission on an object are granted Execute Procedure permission by default for that object.</td>
</tr>
</tbody>
</table>
Note: The granted field of a Permission is reserved for future use to designate whether a Permission is explicitly not granted, that is to say, whether it is explicitly denied. In EMC Documentum 6, only granted permissions are returned by services.

PermissionProfile

The PermissionProfile class enables the client to set filters that control the contents of the Permission lists in DataObject instances returned by services. By default, services return an empty Permission list: the client must explicitly request in a PermissionProfile that permissions be returned.

The ContentProfile includes a single filter, PermissionTypeFilter, with a corresponding permissionType setting that is used or ignored depending on the PermissionTypeFilter value. The permissionType is specified with a Permission.PermissionType enum constant.

The following table describes the permission profile filter settings:

<table>
<thead>
<tr>
<th>Value type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PermissionType-Filter</td>
<td>NONE</td>
<td>No permissions are included</td>
</tr>
<tr>
<td></td>
<td>SPECIFIED</td>
<td>Include only permissions of the type specified by the PermissionType attribute</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>Include permissions of all types</td>
</tr>
</tbody>
</table>

Compound (hierarchical) permissions

Content Server BASIC permissions are compound (sometimes called hierarchical), meaning that there are conceptual levels of permission, with each level including all lower-level permissions. For example, if a user has RELATE permissions on an object, the user is also implicitly granted READ and BROWSE permissions on the object. This is a convenience for permission management, but it complicates the job of a service consumer that needs to determine what permissions a user has on an object.

The PermissionProfile class includes a useCompoundPermissions setting with a default value of false. This causes any permissions list returned by a service to include all BASIC permissions on an object. For example, if a user has RELATE permissions on the object, a Permissions list would be returned containing three BASIC permissions: RELATE, READ, and BROWSE. You can set useCompoundPermissions to true if you only need the highest-level BASIC permission.

Example

The following example sets a PermissionProfile in operationOptions, specifying that all permissions are to be returned by the service.

```java
PermissionProfile permissionProfile = new PermissionProfile();
permissionProfile.setPermissionTypeFilter(PermissionTypeFilter.ANY);
OperationOptions operationOptions = new OperationOptions();
```
operationOptions.setPermissionProfile(permissionProfile);

Relationship

Relationships allow the client to construct a single DataObject that specifies all of its relations to other objects, existing and new, and to get, update, or create the entire set of objects and their relationships in a single service interaction.

The Relationship class and its subclasses, ObjectRelationship and ReferenceRelationship, define the relationship that a repository object (represented by a DataObject instance) has, or is intended to have, to another object in the repository (represented within the Relationship instance). The repository defines object relationships using different constructs, including generic relationship types represented by hardcoded strings (folder and virtual_document); dm_relation objects, which contain references to dm_relation_type objects; and dmc_relationship_def objects, a representation provides more sophistication in Documentum 6. The DFS Relationship object provides an abstraction for dealing with various metadata representations in a uniform manner.

This document will use the term container DataObject when speaking of the DataObject that contains a Relationship. It will use the term target object to refer to the object specified within the Relationship. Each Relationship instance defines a relationship between a container DataObject and a target object. In the case of the ReferenceRelationship subclass, the target object is represented by an ObjectIdentity; in the case of an ObjectRelationship subclass, the target object is represented by a DataObject. Relationship instances can therefore be nested, allowing the construction of complex DataObject graphs.

ReferenceRelationship and ObjectRelationship

The create and update Object service operations use distinct rules when processing instances of ReferenceRelationship and ObjectRelationship.

A ReferenceRelationship represents a relationship to an existing repository object, and is specified using an ObjectIdentity. A ReferenceRelationship can be used to create a relationship between two objects, but it cannot be used to update or create target objects. A common use case would be linking a repository object (as it is created or updated) into an existing folder.

An ObjectRelationship represents a relationship to a new or existing repository object. An ObjectRelationship is used by the update operation to either update or create target objects. If an ObjectRelationship received by an update operation represents a new repository object, the object is created. If the ObjectRelationship represents an existing repository object, the object is updated. A possible use case would be the creation of a new folder and a set of new documents linked to the folder.

Relationship model

Figure 3, page 45 shows the model of Relationship and related classes.
Figure 3. Relationship model

Relationship fields

The following table describes the fields of the Relationship class:

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetRole</td>
<td>String</td>
<td>Specifies the role of the target object in the Relationship. For example, in relationships between a folder and an object linked into the folder the roles are parent and child.</td>
</tr>
<tr>
<td>intentModifier</td>
<td>RelationshipIntentModifier</td>
<td>Specifies how the client intends for the Relationship object to be handled by an update operation.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the relationship. The hard-coded names &quot;folder&quot; and &quot;virtual_document&quot; are mapped to the default implementation of the folder and VDM relationships. In relationships based on dm_relation objects, the dm_relation_type name as the relationship name.</td>
</tr>
<tr>
<td>properties</td>
<td>PropertySet</td>
<td>If the relationship supports custom properties, these properties can be provided in the PropertySet. The relationship implementation should support a separate persistent object in this case. For example: a subtype of dm_relation with custom attributes.</td>
</tr>
</tbody>
</table>

RelationshipIntentModifier

The following table describes the possible values for the RelationshipIntentModifier.
### IntentModifier value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD</td>
<td>Specifies that the relation should be added by an update operation if it does not exist, or updated if it does exist. This is the default value: the intentModifier of any Relationship is implicitly ADD if it is not explicitly set to REMOVE.</td>
</tr>
<tr>
<td>REMOVE</td>
<td>This setting specifies that a relationship should be removed by an update operation.</td>
</tr>
</tbody>
</table>

### Relationship targetRole

Relationships are directional, having a notion of source and target. The targetRole of a Relationship is a string representing the role of the target in a relationship. In the case of folders and VDMs, the role of a participant in the relationship can be parent or child. The following table describes the possible values for the Relationship targetRole.

<table>
<thead>
<tr>
<th>TargetRole value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship.ROLE_PARENT</td>
<td>Specifies that the target object has a parent relationship to the container DataObject. For example, if a DataObject represents a dm_document, and the target object represents a dm_folder, the targetRole of the Relationship should be &quot;parent&quot;. This value is valid for folder and virtual document relationships, as well as relationships based on a dm_relation object.</td>
</tr>
<tr>
<td>Relationship.ROLE_CHILD</td>
<td>Specifies that the target object has a child relationship to the container DataObject. For example, if a DataObject represents a dm_folder, and the target object represents a dm_document, the targetRole of the Relationship would be child. This value is valid for folder and virtual document relationships, as well as relationships based on a dm_relation object.</td>
</tr>
<tr>
<td>&lt;custom role&gt;</td>
<td>A custom value can be supplied for the role. This value has match the relationship schema defined using the Documentum version 6 relationship extensions.</td>
</tr>
</tbody>
</table>

### DataObject as data graph

A DataObject, through the mechanism of Relationship, comprises a data graph or tree of arbitrary depth and complexity. When the DataObject is parsed by a service, each DataObject directly contained in the DataPackage is interpreted as the root of the tree. A ReferenceRelationship, because it does not nest a DataObject, is always necessarily a leaf of the tree. An ObjectRelationship can be branch or leaf. Figure 4, page 47 shows a complex DataObject consisting of a set of related folders.
**Figure 4. Relationship tree**

The order of branching is determined not by hierarchy of parent-child relationships, but by the nesting of Relationship instances within DataObject instances. In some service processing it may be useful to reorder the graph into a tree based on parent-child hierarchy. Some services do this reordering and parse the tree from the root of the transformed structure.

**DataObject graph structural types**

A DataObject can have any of the following structures:

- A simple standalone DataObject, which contains no Relationship instances.
- A DataObject with references, containing only instances of ReferenceRelationship.
- A compound DataObject, containing only instances of ObjectRelationship.
- A compound DataObject with references, containing both ReferenceRelationship and ObjectRelationship instances.

**Standalone DataObject**

A standalone DataObject has no specified relationships to existing repository objects or to other DataObject instances. Standalone DataObject instances would typically be the result of a get operation or used to update an existing repository object. They could also be created in the repository independently of other objects, but normally a new object would have at least one ReferenceRelationship to specify a folder location. **Figure 5, page 47** represents an object of this type.
**DataObject with references**

A DataObject with references models a repository object (new or existing) with relationships to existing repository objects. References to the existing objects are specified using objects of class ObjectIdentity.

As an example, consider the case of a document linked into two folders. The DataObject representing the document would need two ReferenceRelationship instances representing dm_folder objects in the repository. The relationships to the references are directional: from parent to child. The folders must exist in the repository for the references to be valid. Figure 6, page 48 represents an object of this type.

**Figure 6. DataObject with references**

![Diagram showing a DataObject with references to folder1 and folder2.]

To create this object with references you could write code that does the following:

2. Add to doc1 a ReferenceRelationship to folder1 with a targetRole of "parent".
3. Add to doc1 a ReferenceRelationship to folder2 with a targetRole of "parent".

In most cases the client would know the ObjectId of each folder, but in some cases the ObjectIdentity can be provided using a Qualification, which would eliminate a remote query to look up the folder ID.

Let’s look at a slightly different example of an object with references (Figure 7, page 48). In this case we want to model a new folder within an existing folder and link an existing document into the new folder.

**Figure 7. DataObject with parent and child references**

![Diagram showing a DataObject with a parent folder1 and a child folder2.]
To create this DataObject with references you could write code that does the following:
2. Add to folder1 a ReferenceRelationship to folder2 with a targetRole of "parent".
3. Add to folder1 a ReferenceRelationship to doc1 with a targetRole of "child".

**Compound DataObject instances**

In many cases it is relatively efficient to create a complete hierarchy of objects and then create or update it in the repository in a single service interaction. This can be accomplished using a compound DataObject, which is a DataObject containing ObjectRelationship instances.

A typical case for using a compound DataObject would be to replicate a file system’s folder hierarchy in the repository. Figure 8, page 49 represents an object of this type.

**Figure 8. Compound DataObject**

To create this compound DataObject you could write code that does the following:
1. Create a new DataObject, folder 1.
2. Add to folder 1 an ObjectRelationship to a new DataObject, folder 1.1, with a targetRole of "child".
3. Add to folder 1.1 an ObjectRelationship to a new DataObject, folder 1.1.1, with a targetRole of "child".
4. Add to folder 1.1 an ObjectRelationship to a new DataObject, folder 1.1.2, with a targetRole of "child".
5. Add to folder 1 an ObjectRelationship to a new DataObject, folder 1.2, with a targetRole of "child".

In this logic there is a new DataObject created for every node and attached to a containing DataObject using a child ObjectRelationship.
Compound DataObject with references

In a normal case of object creation, the new object will be linked into one or more folders. This means that a compound object will also normally include at least one ReferenceRelationship. Figure 9, page 50 shows a compound data object representing a folder structure with a reference to an existing folder into which to link the new structure.

Figure 9. Compound object with references

To create this compound DataObject you could write code that does the following:
1. Create a new DataObject, folder 1.
2. Add to folder 1 an ObjectRelationship to a new DataObject, folder 1.1, with a targetRole of "child".
3. Add to folder 1.1 an ObjectRelationship to a new DataObject, folder 1.1.1, with a targetRole of "child".
4. Add to folder 1.1 an ObjectRelationship to a new DataObject, folder 1.1.2, with a targetRole of "child".
5. Add to folder 1 a ReferenceRelationship to an existing folder 1.2, with a targetRole of "parent".

Removing object relationships

The Relationship intentModifier setting allows you to explicitly specify how an update operation processes ReferenceRelationship objects. The default setting of intentModifier for all Relationship instances is ADD, which means that the update operation will handle the ReferenceRelation using default processing. Setting intentModifier to REMOVE requests that the update service remove an existing relation. Figure 10, page 51 illustrates this:
The preceding diagram shows that a new PARENT relation to folder 3 is added to folder 1, and an existing relation with folder 2 is removed. This has the effect of linking folder1 into folder3 and removing it from folder2. The folder2 object is not deleted.

To configure the data object you would:

1. Create a new DataObject, folder1.
2. Add to folder1 a ReferenceRelationship to folder2, with an intentModifier set to REMOVE.
3. Add to folder1 a ReferenceRelationship to folder3, with a targetRole of "parent".

**RelationshipProfile**

A RelationshipProfile is a client optimization mechanism that provides fine control over the size and complexity of DataObject instances returned by services. By default, the Object service get operation returns DataObject containing no Relationship instances. To alter this behavior, you must provide a RelationshipProfile that explicit sets the types of Relationship instances to return.

**ResultDataMode**

The RelationshipProfile.resultDataMode setting determine whether the Relationship instances contained in a DataObject returned by an Object service get operation are of type ObjectRelationship or ReferenceRelationship. If they are of type ObjectRelationship they will contain actual DataObject instances; if they are of type ReferenceRelationship, they will contain only an ObjectIdentity. The following table describe the possible values of resultDataMode:

<table>
<thead>
<tr>
<th>resultDataMode value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REFERENCE</td>
<td>Return all Relationship instances as ReferenceRelationship, which contain only the ObjectIdentity of the related object.</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Return all relations as ObjectRelationship objects, which contain actual DataObject instances.</td>
</tr>
</tbody>
</table>

Note that if resultDataMode is set to REFERENCE, the depth of relationships retrieved can be no greater than 1. This is because the related objects retrieved will be in the form of an ObjectIdentity, and so cannot nest any Relationship instances.
**Relationship filters**

RelationshipProfile includes a number of filters that can be used to specify which categories of Relationship instances are returned as part of a DataObject. For some of the filters you will need to specify the setting in a separate field and set the filter to SPECIFIED. For example, to filter by relationName, set nameFilter to SPECIFIED, and use the relationName field to specify the relationship name string.

The filters are ANDed together to specify the conditions for inclusion of a Relationship instance. For example, if targetRoleFilter is set to RelationshipProfile.ROLE_CHILD and depthFilter is set to SINGLE, only proximate child relationships will be returned by the service.

The following table describes the filters and their settings.

<table>
<thead>
<tr>
<th>Value type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RelationshipNameFilter</td>
<td>SPECIFIED</td>
<td>Only Relationship instances with the name specified in the relationName field will be included.</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>relationName field is ignored, and Relationship instances are not filtered by name.</td>
</tr>
<tr>
<td>TargetRoleFilter</td>
<td>SPECIFIED</td>
<td>Only relations with the target role specified in the targetRole attribute will be included.</td>
</tr>
<tr>
<td></td>
<td>ANY</td>
<td>Do not filter Relationship instances by targetRole setting (that is, ignore targetRole setting).</td>
</tr>
<tr>
<td>DepthFilter</td>
<td>SINGLE</td>
<td>Return only proximate Relationship instances (only Relationship instances that are immediate constituents of the root DataObject).</td>
</tr>
<tr>
<td></td>
<td>SPECIFIED</td>
<td>Return Relationship instances to the level specified in the depth field.</td>
</tr>
<tr>
<td></td>
<td>UNLIMITED</td>
<td>Return Relationship instances without regard to depth field, to indeterminate level.</td>
</tr>
</tbody>
</table>

**DepthFilter restrictions**

Relationships deeper than one level from the primary DataObject will be returned in a data graph only if they have the same relationship name and targetRole as the intervening relationship. For example, Figure 11, page 53 represents a repository object, doc 0, with relationships to a parent folder object (folder b) and with a virtual_document relationship to another document (doc 1).
Suppose a client were to use the get operation to retrieve doc 0 using the following RelationshipProfile settings:

```plaintext
nameFilter = ANY
targetRoleFilter = ANY
depthFilter = SPECIFIED
depth = 2
```

In this case, folder b, folder c, and doc 1 would all be included in the returned data graph. However, folder a would not be included, because the relationship between doc 1 and folder a does not have the same name and targetRole as the relationship between doc 0 and doc 1.

**Other classes related to DataObject**

Other classes related to DataObject are covered in this manual under the service with which they are most closely associated.
Chapter 3

Enterprise Content Services

This chapter covers some common features of Enterprise Content Services, which include the services delivered with DFS, as well as extended DFS services that can be delivered with EMC applications, or created by EMC customers and partners. The following topics are covered:

- Enterprise Content Services, page 55
- Service commonalities, page 56
- Service Context, page 56
- Service instantiation, page 59
- OperationOptions, page 60

Enterprise Content Services

Enterprise Content Services is an umbrella term for services provided by EMC that operate within the framework of DFS technologies. DFS includes a set of out-of-the-box Enterprise Content Services that provide a service-oriented programmatic interface to Content Server and associated repositories. These services include a set of core services, such as the Object service (which provides fundamental operations for creating, getting, updating, and deleting repository objects), and are extended with services that provide additional platform functionality. Table 6, page 55 shows the set of services provided at the time of writing.

Table 6. Enterprise Content Services provided with the DFS product

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Provides fundamental operations for creating, getting, updating, and deleting repository objects, as well as copy and move operations. For more information see Chapter 4, Object Service.</td>
</tr>
<tr>
<td>VersionControl</td>
<td>Provides operations that concern specific versions of repository objects, such as checkin and checkout. For more information see Chapter 5, VersionControl Service.</td>
</tr>
<tr>
<td>Query</td>
<td>Provides operations for obtaining data from repositories using ad-hoc queries. For more information see Chapter 7, Query Service.</td>
</tr>
</tbody>
</table>
### Service commonalities

All Enterprise Content Services delivered with DFS, as well as any services that are created using the DFS tools and executed using the DFS runtime, have certain features in common, some resulting from the technical framework in which the services are generated, and some established by convention.

- Services use the shared data model, unless it is absolutely necessary to create new models.
- Operation access security is handled in strict compliance with WS-Security. At the same time, repository access is provided using the Context Registry Service (which is part of the DFS runtime). This service allows registration of service context with multiple repository identities in exchange for a secure context token. This token can be used in subsequent service requests to access the context and the identities held within the context. Thus the same token is used for both service-level security and repository access.
- Classes of exceptions thrown by DFS services are serializable, and the DFS runtime has facilities that marshal and unmarshal the exception stack trace. If you are using the Java client library, you can view the exception stack trace as you would in a local application.

**Note:** While DFS version 6 complies with the WS-Security standard, it does not provide support out-of-the-box for related technologies such as SAML, Kerberos tickets, and X.509 tickets.

### Service Context

Services invocation in DFS takes place within a service context, which is a stateful object maintaining identity information for service authentication, profiles for setting options and filters, a locale, and properties. Service context can be shared among multiple services.
Setting up service context

To be used by a service that requires authentication, the service context should be populated with at least one identity. The following sample creates and returns a minimal service context.

Example 3-1. Java: Minimal service context

```java
public IServiceContext getSimpleServiceContext(String repositoryName,
                                                String userName,
                                                String password)
{
    ContextFactory contextFactory = ContextFactory.getInstance();
    IServiceContext context = contextFactory.newContext();
    RepositoryIdentity repoId = new RepositoryIdentity();
    repoId.setRepositoryName(repositoryName);
    repoId.setUserName(userName);
    repoId.setPassword(password);
    context.addIdentity(repoId);
    return context;
}
```

A service context will also typically contain a ContentTransferProfile, as shown in the following C# example:

Example 3-2. C#: Initializing service context

```csharp
private void initializeContext()
{
    ContextFactory contextFactory = ContextFactory.Instance;
    serviceContext = contextFactory.NewContext();

    RepositoryIdentity repoId = new RepositoryIdentity();
    RepositoryIdentity repositoryIdentity =
        new RepositoryIdentity(DefaultRepository, UserName, Password, "");
    serviceContext.AddIdentity(repositoryIdentity);

    ContentTransferProfile contentTransferProfile = new ContentTransferProfile();
    contentTransferProfile.TransferMode = ContentTransferMode.UCF;
    contentTransferProfile.GeoLocation = "Pleasanton";
    serviceContext.SetProfile(contentTransferProfile);
}
```

Identities

A service context contains a collection of identities, which are mappings of repository names onto sets of user credentials used in service authentication. A service context is expected to contain only one identity per repository name. Identities will be set in a service context using one of two concrete subclasses: BasicIdentity and RepositoryIdentity.

BasicIdentity directly extends the Identity parent class, and includes accessors for user name and password, but not for repository name. This class can be used in cases where the service is known to access only a single repository, or in cases where the user credentials in all repositories are known to be identical. BasicIdentity can also be used to supply fallback credentials in the case where the user has differing credentials on some repositories, for which RepositoryIdentity instances will be set, and identical credentials on all other repositories.
RepositoryIdentity extends BasicIdentity, and specifies a mapping of repository name to a set of user credentials, which include a user name, password, and optionally a domain name if required by your network environment.

**Service context properties**

A service context contains a RuntimeProperties collection, in which properties can be set for all services sharing the service context. These properties settings can be used to store configuration settings that are scoped to the service context, and which therefore are not suitable as operation parameters or inclusion in the OperationOptions PropertySet. Properties included in RuntimeProperties would generally be standalone properties. If your service configuration requires a set of interrelated properties, it may be more appropriate to extend the Profile class and manage the settings within the profile. DFS services generally use profiles in preference to RuntimeProperties.

**Context registration**

Registering a service returns a token string, which can be used to reference the service context in subsequent service instantiations. Registration requires a single interaction with the ContextRegistryService, subsequent to which the ServiceContext can be passed over the wire containing only the single token. This is an appropriate optimization in applications where multiple services will be created sharing the same ServiceContext, and the ServiceContext is of significant size. If the ServiceContext is small and profiles are largely contained within the OperationOptions argument, then registration of the service context will not result in a significant optimization.

Clients can also make delta modifications to a registered service context by setting profiles, properties, or identities to the service context before passing it to a ServiceFactory method.

To register a context, a WSDL client uses the ContextRegistry service provided with DFS services. From the Java and .NET client libraries, this service interaction is transparent and handled by the DFS runtime when the client calls one of the ContextFactory.register methods.

```csharp
serviceContext = contextFactory.register(serviceContext);
```

Note that the method of registering a service context in WSDL clients requires the consumer to invoke the ContextRegistry runtime service. The service register method returns a serviceToken string that is used subsequently to associate a service with its context. For an example using a C# WSDL client see .

**Getting service context for a chained service**

When a DFS service instantiates another DFS service, it will generally want the service it is invoking to share its service context. A service can obtain its own service context and pass it to the ServiceFactory method used to instantiate the chained service. For example, the following gets an instance of the SchemaService:

```csharp
IServiceContext context = ContextFactory.getInstance().getContext();
ISchemaService schemaService
    = ServiceFactory.getInstance()
    .getRemoteService(ISchemaService.class,
```
To see this example in context, refer to Sample service, page 186.

Note: DFS does not support registration of the service context in this case.

Service instantiation

The technique for instantiating a service varies somewhat, depending on whether you are using the Java client library or the .NET client library.

Service instantiation in Java

A Java client (or a service) can create an instance of a service using one of three methods of ServiceFactory, shown in Table 7, page 59.

Table 7. Methods for instantiating services

<table>
<thead>
<tr>
<th>Java ServiceFactory method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getRemoteService</td>
<td>Get service to be called using remote (SOAP) interaction.</td>
</tr>
<tr>
<td>getLocalService</td>
<td>Get service to be called and executed in local JVM.</td>
</tr>
<tr>
<td>getService</td>
<td>Attempts to instantiate the service locally; if this fails, then attempts to instantiate the service remotely.</td>
</tr>
</tbody>
</table>

All of these methods are overloaded on the Java client to allow either explicit or implicit service addressing. Implicit service addressing on the Java client uses settings provided by the Java runtime client configuration (in dfs-client.xml) as implicit service module name and context root. Explicit addressing requires passing of a service module name and context root in ServiceFactory method, for example:

Example 3-3. Java: Service instantiation with explicit addressing

```java
objectService = serviceFactory.getRemoteService(IObjectService.class,
                                             serviceContext,
                                             "core",
                                             "http://MyServerHost:9080/services");
```

Service instantiation in C#

The .NET client library does not support local service invocation; therefore it provides a single method, GetRemoteService, for service instantiation. Note that this is a generic method.

GetRemoteService is overloaded to allow either explicit or implicit service addressing. Implicit service addressing uses settings provided in the application configuration (in app.config, see .NET
client configuration, page 22) as implicit service module name and context root. Explicit addressing requires passing of a service module name and context root in ServiceFactory method, for example:

**Example 3-4. C#: Service instantiation with explicit addressing**

```csharp
objectService =
    serviceFactory.GetRemoteService<IObjectService>(DemoServiceContext
        "core",
        "http://MyServiceHost:9080/services");
```

**OperationOptions**

DFS services generally take an OperationOptions object as their final argument. OperationOptions contains profiles and properties that specify behaviors for the operation. The properties have no overlap with properties set in the service context RuntimeProperties. The profiles can potentially overlap with properties stored in the service context. In the case that they do overlap, the profiles in OperationOptions always take precedence over profiles stored in the service context. The profiles stored in the service context take effect when no matching profile is stored in the OperationOptions for a specific operation. The override of profiles in the service context takes place on a profile-by-profile basis: there is no merge of specific settings stored within the profiles.

As a recommended practice, a service client should avoid storing profiling information or properties in the service operation that are likely to be modified by specific service instances. This avoids possible side-effects caused by modifications to a service context shared by multiple services. It is likely that ContentTransferProfile will not change and so should be included in the service context. Other profiles are better passed within OperationOptions.

OperationOptions are discussed in more detail under the documentation for specific service operations. For more information on profiles, see PropertyProfile, page 39, ContentProfile, page 40, PermissionProfile, page 43, RelationshipProfile, page 51, and Controlling data returned by get operation, page 69.
The Object service provides a set of basic operations on repository objects, in cases where the client does not need to explicitly use the version control system. Each operation within the Object service uses default behaviors as relates to object versions that are appropriate for the specific operation. All of the Object service operations can operate on multiple objects (contained in either a DataPackage or an ObjectIdentitySet), enabling clients to optimize service usage by minimizing the number of service interactions.

This chapter covers the following topics:

- create operation, page 61
- createPath operation, page 66
- Get operation, page 67
- update operation, page 73
- delete operation, page 78
- copy operation, page 80
- move operation, page 84
- validate operation, page 87
- getObjectContentUrls operation, page 88

create operation

The Object service create operation creates a set of new repository objects based on the DataObject instances contained in a DataPackage passed to the operation. Because each DataObject represents a new repository object, its ObjectIdentity is populated with only a repository name. Content Server assigns a unique object identifier when the object is created in the repository.

To create an object in a specific location, or to create objects that have relationships to one another defined in the repository, the client can define Relationship instances in a DataObject passed to the operation. The most common example of this would be to create a Relationship between a newly created document and the folder in which it is to be created.
Java syntax

DataPackage create(DataPackage dataPackage,
                     OperationOptions operationOptions)
   throws CoreServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataPackage</td>
<td>DataPackage</td>
<td>A collection of DataObject instances that identify the repository objects to be created.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>An object containing profiles and properties that specify operation behaviors.</td>
</tr>
</tbody>
</table>

Profiles

Generally the expected behavior of the create operation can be logically determined by the objects contained within the DataPackage passed to the operation. For example, if DataObject instances contained in the DataPackage include content, the operation assumes that it should transfer the content and create contentful repository objects. Similarly, if DataObject instances contain Relationship objects, the relationships are created along with the primary object. The profile that does have direct bearing on the create operation is the ContentTransferProfile, which determines the mode used to transfer content from the remote client to the repository. The ContentTransferProfile will generally be set in the service context. For information on this profile, refer to ContentTransferProfile, page 162.

Other profiles, such as ContentProfile, PropertyProfile, and RelationshipProfile, will control the composition of the DataPackage returned by the create operation, which by default will contain an ObjectIdentity only. These profiles can allow the client to obtain detailed information about created objects if required without performing an additional query.

Response

Returns a DataPackage containing one DataObject for each repository object created by the create operation. By default, each DataObject contains only the ObjectIdentity of the created object and no other data. The client can modify this behavior by using Profile objects if it requires more data about the created objects.
Examples

The following examples demonstrate:

- Simple object creation, page 63
- Creating and linking, page 63
- Creating multiple related objects, page 65

Simple object creation

The following sample creates a folder in the repository in the default location.

Example 4-1. Java: Simple object creation

```java
public DataPackage createNewFolder() throws ServiceException {
    ObjectIdentity folderIdentity = new ObjectIdentity();
    folderIdentity.setRepositoryName(defaultRepositoryName);
    DataObject dataObject = new DataObject(folderIdentity, "dm_folder");
    PropertySet properties = new PropertySet();
    String folderName = "aTestFolder-" + System.currentTimeMillis();
    properties.set("object_name", folderName);
    dataObject.setProperties(properties);

    DataPackage dataPackage = new DataPackage(dataObject);

    OperationOptions operationOptions = null;
    return objectService.create(dataPackage, operationOptions);
}
```

Example 4-2. C#: Simple object creation

```csharp
public DataPackage CreateNewFolder() {
    ObjectIdentity folderIdentity = new ObjectIdentity();
    folderIdentity.RepositoryName = DefaultRepository;
    DataObject dataObject = new DataObject(folderIdentity, "dm_folder");
    PropertySet properties = new PropertySet();
    String folderName = "aTestFolder-" + System.DateTime.Now.Ticks;
    properties.Set("object_name", folderName);
    dataObject.Properties = properties;

    DataPackage dataPackage = new DataPackage(dataObject);

    OperationOptions operationOptions = null;
    return objectService.Create(dataPackage, operationOptions);
}
```

Creating and linking

The following sample creates an object and uses a ReferenceRelationship to link it into an existing folder.
Example 4-3. Java: Creating and linking

```java
public DataObject createAndLinkToFolder(String folderPath)
{
    // create a contentless document to link into folder
    String objectName = "linkedDocument" + System.currentTimeMillis();
    String repositoryName = defaultRepositoryName;
    ObjectIdentity sampleObjId = new ObjectIdentity(repositoryName);
    DataObject sampleDataObject = new DataObject(sampleObjId, "dm_document");
    sampleDataObject.getProperties().set("object_name", objectName);

    // add the folder to link to as a ReferenceRelationship
    ObjectPath objectPath = new ObjectPath(folderPath);
    ObjectIdentity/ObjectPath> sampleFolderIdentity
        = new ObjectIdentity/ObjectPath>(objectPath,
            defaultRepositoryName);
    ReferenceRelationship sampleFolderRelationship = new ReferenceRelationship();
    sampleFolderRelationship.setName(Relationship.RELATIONSHIP_FOLDER);
    sampleFolderRelationship.setTarget(sampleFolderIdentity);
    sampleFolderRelationship.setTargetRole(Relationship.ROLE_PARENT);
    sampleDataObject.getRelationships().add(sampleFolderRelationship);

    // create a new document linked into parent folder
    try
    {
        OperationOptions operationOptions = null;
        dataPackage dataPackage = new DataPackage(sampleDataObject);
        objectService.create(dataPackage, operationOptions);
    }
    catch (ServiceException e)
    {
        throw new RuntimeException(e);
    }

    return sampleDataObject;
}
```

Example 4-4. C#: Creating and linking

```csharp
public DataObject CreateAndLinkToFolder(String folderPath)
{
    // create a contentless document to link into folder
    String objectName = "linkedDocument" + System.DateTime.Now.Ticks;
    String repositoryName = DefaultRepository;
    ObjectIdentity sampleObjId = new ObjectIdentity(repositoryName);
    DataObject sampleDataObject = new DataObject(sampleObjId, "dm_document");
    sampleDataObject.Properties.Set("object_name", objectName);

    // add the folder to link to as a ReferenceRelationship
    ObjectPath objectPath = new ObjectPath(folderPath);
    ObjectIdentity/ObjectPath> sampleFolderIdentity = new ObjectIdentity/ObjectPath>(objectPath, DefaultRepository);
    ReferenceRelationship sampleFolderRelationship = new ReferenceRelationship();
    sampleFolderRelationship.setName(Relationship.RELATIONSHIP_FOLDER);
    sampleFolderRelationship.Target = sampleFolderIdentity;
    sampleFolderRelationship.TargetRole = Relationship.ROLE_PARENT;
    sampleDataObject.Relationships.Add(sampleFolderRelationship);

    // create a new document linked into parent folder
    OperationOptions operationOptions = null;
    DataPackage dataPackage = new DataPackage(sampleDataObject);
    objectService.Create(dataPackage, operationOptions);

    return sampleDataObject;
}
```
Creating multiple related objects

The following sample creates a folder with a Relationship to a new document. The create service will create both the document and the folder, and link the document into the folder.

Example 4-5. Java: Creating multiple related objects

```java
public DataPackage createFolderAndLinkedDoc() throws ServiceException {
    // create a folder data object
    String folderName = "0test-folder-" + System.currentTimeMillis();
    DataObject folderDataObj = new DataObject(new ObjectIdentity(defaultRepositoryName), "dm_folder");
    PropertySet folderDataObjProperties = new PropertySet();
    folderDataObjProperties.set("object_name", folderName);
    folderDataObj.setProperties(folderDataObjProperties);

    // create a contentless document DataObject
    String doc1Name = "aTestDoc-" + System.currentTimeMillis();
    DataObject docDataObj = new DataObject(new ObjectIdentity(defaultRepositoryName), "dm_document");
    PropertySet properties = new PropertySet();
    properties.set("object_name", doc1Name);
    docDataObj.setProperties(properties);

    // add the folder as a parent of the folder
    ObjectRelationship objRelationship = new ObjectRelationship();
    objRelationship.setTarget(folderDataObj);
    objRelationship.setName(Relationship.RELATIONSHIP_FOLDER);
    objRelationship.setTargetRole(Relationship.ROLE_PARENT);
    docDataObj.getRelationships().add(new ObjectRelationship(objRelationship));

    // create the folder and linked document
    DataPackage dataPackage = new DataPackage();
    dataPackage.addDataObject(docDataObj);
    OperationOptions operationOptions = null;
    return objectService.create(dataPackage, operationOptions);
}
```

Example 4-6. C#: Creating multiple related objects

```csharp
public DataPackage CreateFolderAndLinkedDoc()
{
    // create a folder data object
    string folderName = "0test-folder-" + System.DateTime.Now.Ticks;
    DataObject folderDataObj = new DataObject(new ObjectIdentity(DefaultRepository), "dm_folder");
    PropertySet folderDataObjProperties = new PropertySet();
    folderDataObjProperties.Set("object_name", folderName);
    folderDataObj.Properties = folderDataObjProperties;

    // create a contentless document DataObject
    string doc1Name = "aTestDoc-" + System.DateTime.Now.Ticks;
    DataObject docDataObj = new DataObject(new ObjectIdentity(DefaultRepository), "dm_document");
    PropertySet properties = new PropertySet();
    properties.Set("object_name", doc1Name);
    docDataObj.Properties = properties;

    // add the folder as a parent of the folder
    ObjectRelationship objRelationship = new ObjectRelationship();
    objRelationship.Target = folderDataObj;
    objRelationship.TargetRole = Relationship.ROLE_PARENT;
    objRelationship.Name = Relationship.RELATIONSHIP_FOLDER;
    docDataObj.AddRelationships(new ObjectRelationship(objRelationship));

    // create the folder and linked document
    DataPackage dataPackage = new DataPackage();
    dataPackage.AddDataObject(docDataObj);
    OperationOptions operationOptions = null;
    return objectService.Create(dataPackage, operationOptions);
}
```
ObjectService

```java
objRelationship.Name = Relationship.RELATIONSHIP_FOLDER;
objRelationship.TargetRole = Relationship.ROLE_PARENT;
docDataObj.Relationships.Add(new ObjectRelationship(objRelationship));

// create the folder and linked document
DataPackage dataPackage = new DataPackage();
dataPackage.AddDataObject(docDataObj);
OperationOptions operationOptions = null;
return objectService.Create(dataPackage, operationOptions);
```

createPath operation

The createPath operation creates a folder structure (from the cabinet down) in a repository. The path is passed to the service as an ObjectPath, which contains a path String in the format "/cabinetName/folderName...", which can be extended to any depth. If any of the folders specified in the path exist, no exception is thrown. This allows you to use the operation to create the complete path, or to add new folders to an existing path.

Java syntax

```java
ObjectIdentity createPath(ObjectPath objectPath, String repositoryName) throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectPath</td>
<td>ObjectPath</td>
<td>Contains a String in the form &quot;/cabinetName/folderName...&quot; that describes the complete path to create.</td>
</tr>
</tbody>
</table>

Response

Returns the ObjectIdentity of the final object in the path. For example, if the path is "/cabinetName/childFolder1/childFolder2", the operation will return the ObjectIdentity of childFolder2.

Example

The following sample creates a path consisting of a cabinet and a folder. If the cabinet exists, only the folder is created. If the cabinet and folder both exist, the operation does nothing.
Example 4-7. Java: Creating a folder in a cabinet

```java
public ObjectIdentity createFolderInCabinet()
    throws ServiceException
{
    ObjectPath objPath = new ObjectPath("/DFSTestCabinet/createPathTestFolder");
    return objectService.createPath(objPath, defaultRepositoryName);
}
```

Example 4-8. C#: Creating a folder in a cabinet

```csharp
public ObjectIdentity CreateFolderInCabinet()
{
    ObjectPath objPath = new ObjectPath("/DFSTestCabinet/createPathTestFolder");
    return objectService.CreatePath(objPath, DefaultRepository);
}
```

Get operation

Description

The get operation retrieves a set of objects from the repository based on the contents of an ObjectIdentitySet. The get operation always returns the version of the object specified by ObjectIdentity; if the ObjectIdentity identifies a non-CURRENT version, the get operation returns the non-CURRENT version. The operation will also return related objects if instructed to do so by RelationshipProfile settings.

The get operation supports retrieval of content from external sources available to the Search service (see Getting content from external sources, page 72).

Java syntax

```java
DataPackage get(ObjectIdentitySet forObjects,
    OperationOptions operationOptions)
    throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forObjects</td>
<td>ObjectIdentity-Set</td>
<td>Contains a list of ObjectIdentity instances specifying the repository objects to be retrieved.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>An object containing profiles and properties that specify operation behaviors. If this object is null, default operation behaviors will take effect.</td>
</tr>
</tbody>
</table>
Profiles

See Controlling data returned by get operation, page 69.

Response

Returns a DataPackage containing DataObject instances representing objects retrieved from the repository (see DataPackage, page 27 and DataObject, page 28). The client can control the complexity of the data in each DataObject using Profile settings passed in operationOptions or stored in the service context. The following table summarizes the data returned by the operation in the default case; that is, if no profiles are set.

<table>
<thead>
<tr>
<th>Data</th>
<th>Default behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Properties</td>
<td>Returns all non-system.</td>
</tr>
<tr>
<td>Content</td>
<td>Returns none.</td>
</tr>
<tr>
<td>Relations</td>
<td>Returns none.</td>
</tr>
<tr>
<td>Permission</td>
<td>Returns none.</td>
</tr>
</tbody>
</table>

For more information see Controlling data returned by get operation, page 69.

Example

The following example uses an ObjectId to reference a repository object and retrieves it from the repository.

**Example 4-9. Java: Basic object retrieval**

```java
public DataObject getObjectWithDefaults(ObjectIdentity objIdentity)
    throws ServiceException
{
    objIdentity.setRepositoryName(defaultRepositoryName);
    ObjectIdentitySet objectIdSet = new ObjectIdentitySet();
    List<ObjectIdentity> objIdList = objectIdSet.getIdentities();
    objIdList.add(objIdentity);

    OperationOptions operationOptions = null;
    DataPackage dataPackage = objectService.get(objectIdSet, operationOptions);

    return dataPackage.getDataObjects().get(0);
}
```

**Example 4-10. C#: Basic object retrieval**

```csharp
public DataObject GetObjectWithDefaults(ObjectIdentity objIdentity)
{
    objIdentity.RepositoryName = DefaultRepository;
    ObjectIdentitySet objectIdSet = new ObjectIdentitySet();
    List<ObjectIdentity> objIdList = objectIdSet.Identities;
    objIdList.Add(objIdentity);

    OperationOptions operationOptions = null;
    ```
Controlling data returned by get operation

You can control the type and quantity of data returned as part of a DataObject by the get operation using profiles, which are made accessible to the get operation via the operationOptions parameter or the service context. The following profiles can be used:

- PropertyProfile
- PermissionProfile
- RelationshipProfile
- ContentProfile

Filtering properties using PropertyProfile

By default, the get operation returns all non-system properties of an object (PropertyFilterMode.ALL_NON_SYSTEM). The following example shows how to configure the PropertyProfile so that the get operation returns no properties.

Example 4-11. Java: PropertyProfile

```java
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.setFilterMode(PropertyFilterMode.NONE);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setPropertyProfile(propertyProfile);
ObjectIdentitySet objectIdSet = new ObjectIdentitySet(objIdentity);
```

Example 4-12. C#: PropertyProfile

```csharp
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.FilterMode = PropertyFilterMode.NONE;
OperationOptions operationOptions = new OperationOptions();
operationOptions.PropertyProfile = propertyProfile;
ObjectIdentitySet objectIdSet = new ObjectIdentitySet(objIdentity);
```

Another useful option is to set the filter mode to SPECIFIED_BYINCLUDE, and provide a list of the specific properties that the client program requires.

Example 4-13. Java: Specifying included properties

```java
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.setFilterMode(PropertyFilterMode.SPECIFIED_BY_INCLUDE);
ArrayList<String> includeProperties = new ArrayList<String>();
includeProperties.add("title");
includeProperties.add("object_name");
includeProperties.add("r_object_type");
propertyProfile.setIncludeProperties(includeProperties);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setOperationProfile(propertyProfile);
```
Example 4-14. C#: Specifying included properties

```csharp
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.FilterMode = PropertyFilterMode.SPECIFIED_BYINCLUDE;
List<string> includeProperties = new List<string>();
includeProperties.Add("title");
includeProperties.Add("object_name");
includeProperties.Add("r_object_type");
propertyProfile.IncludeProperties = includeProperties;
OperationOptions operationOptions = new OperationOptions();
operationOptions.PropertyProfile = propertyProfile;
```

Conversely, you can set the filter mode to SPECIFIED_BY_EXCLUDE and provide a list of excluded properties.

Example 4-15. Java: Specifying excluded properties

```java
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.setFilterMode(PropertyFilterMode.SPECIFIED_BY_EXCLUDE);
ArrayList<String> excludeProperties = new ArrayList<String>();
excludeProperties.add("title");
excludeProperties.add("object_name");
excludeProperties.add("r_object_type");
propertyProfile.setExcludeProperties(excludeProperties);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setPropertyProfile(propertyProfile);
```

Example 4-16. C#: Specifying excluded properties

```csharp
PropertyProfile propertyProfile = new PropertyProfile();
propertyProfile.FilterMode = PropertyFilterMode.SPECIFIED_BY_EXCLUDE;
List<string> excludeProperties = new List<string>();
excludeProperties.Add("title");
excludProperties.Add("object_name");
excludProperties.Add("r_object_type");
propertyProfile.ExcludeProperties = excludeProperties;
OperationOptions operationOptions = new OperationOptions();
operationOptions.PropertyProfile = propertyProfile;
```

For more information on PropertyProfile see PropertyProfile, page 39.

**Controlling Relationship instances**

By default, the get operation returns no Relationship instances. You can use a RelationshipProfile to specify exactly what relation types and relation target roles the get operation will return, and to what depth to return Relationship instances. The get operation returns only ReferenceRelationship; it does not return object relations. For more information on RelationshipProfile, see RelationshipProfile, page 51.

The following example adds a RelationshipProfile to operationOptions to specify that all relations are returned as part of the data object, to any depth.

Example 4-17. Java: RelationshipProfile

```java
RelationshipProfile relationProfile = new RelationshipProfile();
relationProfile.setResultDataMode(ResultDataMode.REFERENCE);
relationProfile.setTargetRoleFilter(TargetRoleFilter.ANY);
```
relationProfile.setNameFilter(RelationshipNameFilter.ANY);  
relationProfile.setDepthFilter(DepthFilter.UNLIMITED);  
OperationOptions operationOptions = new OperationOptions();  
operationOptions.setRelationshipProfile(relationProfile);

Example 4-18. C#: RelationshipProfile

RelationshipProfile relationProfile = new RelationshipProfile();  
relationProfile.ResultDataMode = ResultDataMode.REFERENCE;  
relationProfile.TargetRoleFilter = TargetRoleFilter.SPECIFIED;  
relationProfile.NameFilter = RelationshipNameFilter.ANY;  
relationProfile.DepthFilter = DepthFilter.SINGLE;  
OperationOptions operationOptions = new OperationOptions();  
operationOptions.setRelationshipProfile(relationProfile);

The next example adds a RelationshipProfile to operationOptions to specify that only the proximate parent relations of the data object are returned.

Example 4-19. Java: Filtering relationships, parent only

RelationshipProfile relationProfile = new RelationshipProfile();  
relationProfile.ResultDataMode = ResultDataMode.REFERENCE;  
relationProfile.TargetRoleFilter = TargetRoleFilter.SPECIFIED;  
relationProfile.NameFilter = RelationshipNameFilter.ANY;  
relationProfile.DepthFilter = DepthFilter.SINGLE;  
OperationOptions operationOptions = new OperationOptions();  
operationOptions.setRelationshipProfile(relationProfile);

Example 4-20. C#: Filtering relationships, parent only

RelationshipProfile relationProfile = new RelationshipProfile();  
relationProfile.ResultDataMode = ResultDataMode.REFERENCE;  
relationProfile.TargetRoleFilter = TargetRoleFilter.SPECIFIED;  
relationProfile.NameFilter = RelationshipNameFilter.ANY;  
relationProfile.DepthFilter = DepthFilter.SINGLE;  
OperationOptions operationOptions = new OperationOptions();  
operationOptions.setRelationshipProfile(relationProfile);

Filtering content

By default, the get operation returns no content. The client can use a ContentProfile to specify that content be returned by the get operation, and to filter specific content types.

To specify that any content be returned, set the format filter to ANY, as shown in the following sample.

Example 4-21. Java: Returning any content format

ContentProfile contentProfile = new ContentProfile();  
contentProfile.setFormatFilter(FormatFilter.ANY);  
OperationOptions operationOptions = new OperationOptions();  
operationOptions.setContentProfile(contentProfile);  
operationOptions.setProfile(contentProfile);

Example 4-22. C#: Returning any content format

ContentProfile contentProfile = new ContentProfile();  
contentProfile.FormatFilter = FormatFilter.ANY;  
OperationOptions operationOptions = new OperationOptions();
operationOptions.ContentProfile = contentProfile;
operationOptions.SetProfile(contentProfile);

To specify that only content of a specified format be returned, set the format filter to SPECIFIED and set the format using the setFormat method, as shown in the following sample.

**Example 4-23. Java: Returning a specified content format**

```java
ContentProfile contentProfile = new ContentProfile();
contentProfile.setFormatFilter(FormatFilter.SPECIFIED);
contentProfile.setFormat("gif");
OperationOptions operationOptions = new OperationOptions();
operationOptions.setContentProfile(contentProfile);
```

**Example 4-24. C#: Returning a specified content format**

```csharp
ContentProfile contentProfile = new ContentProfile();
contentProfile.FormatFilter = FormatFilter.SPECIFIED;
contentProfile.Format = "gif";
OperationOptions operationOptions = new OperationOptions();
operationOptions.ContentProfile = contentProfile;
```

You can also filter content by page number or page modifier. For more information on content profiles, see [ContentProfile, page 40](#).

## Getting content from external sources

The Object service get operation can retrieve content from external, ECI-enabled repositories available to the Search service. The Search service getRepositoryList operation returns a list of available sources (both Documentum repositories and external services). The Search service execute operation returns the results of a query against a list of available sources. The get operation can return content based on the query result, as shown here:

**Example 4-25. Java: Retrieving content based on query results**

```java
List<DataObject> dataObjects = result.getDataObjects();
ObjectIdentitySet identities = new ObjectIdentitySet();
for(DataObject data : dataObjects)
{
    identities.addIdentity(data.getIdentity());
}
DataPackage package = objectService.get(identities, operationOptions);
```

For content retrieved from external sources, profiles, such as RelationshipProfile and PropertyProfile, are largely inapplicable. A ContentProfile is required to specify that content be retrieved; however filter settings within the ContentProfile are ignored.

For more information on the Search service, see [Chapter 8, Search Service](#).
update operation

The update operation updates a set of repository objects using data supplied in a set of DataObject instances passed in a DataPackage. The update operation will only update the CURRENT version of an object. If passed an ObjectIdentity that identifies a non-CURRENT object, the operation will throw an exception. The updated repository object will be saved as the CURRENT version.

The ObjectIdentity of each DataObject passed to the update operation must uniquely identify an existing repository object. The DataObject instances can contain updates to properties, content, and relationships, and only needs to include data that requires update.

If a DataObject contains ReferenceRelationship instances, the corresponding relationships are created or updated in the repository. The update operation can also remove existing relationships. It can therefore be used, for example, to unlink an object from a folder and link it into another folder. If the DataObject contains ObjectRelationship instances, then the related objects are either updated or created, depending on whether they already exist in the repository. If the object does not exist, it is created; if it does exist, it is updated.

Java syntax

```java
DataPackage update(DataPackage dataPackage, OperationOptions options) throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataPackage</td>
<td>DataPackage</td>
<td>A collection of DataObject instances that contain modifications to repository objects. The ObjectIdentity of each DataObject instance must uniquely identify the repository object to update. The DataObject instance need only contain data that is to be modified on the repository object; data that is to remain unchanged need not be supplied.</td>
</tr>
<tr>
<td>options</td>
<td>OperationOp-</td>
<td>An object containing profiles and properties that specify operation behaviors.</td>
</tr>
<tr>
<td></td>
<td>tions</td>
<td></td>
</tr>
</tbody>
</table>

Profiles

Generally the expected behavior of the update operation can be logically determined by the objects contained within the DataPackage passed to the operation. For example, if DataObject instances contained in the DataPackage include content, the operation assumes that it should transfer and update content. Similarly, if DataObject instances contain Relationship objects, the relationships are created or updated. The profile that does have direct bearing on the update operation is the ContentTransferProfile, which determines the mode used to transfer content from the remote client to
the repository. The ContentTransferProfile will generally be set in the service context. For information on this profile, refer to ContentTransferProfile, page 162.

Other profiles, such as ContentProfile, PropertyProfile, and RelationshipProfile, will control the contents of the DataPackage returned by the update operation, which by default will contain an ObjectIdentity only. These profiles can allow the client to obtain detailed information about updated objects if required without performing an additional query.

Response

The update operation returns a DataPackage, which by default is populated with DataObject instances that contain only an ObjectIdentity. This default behavior can be changed through the use of Profile objects set in the OperationOptions or service context.

Examples

The following examples demonstrate:

- Updating properties, page 74
- Modifying a repeating properties (attributes) list, page 75
- Updating object relationships, page 77

Updating properties

To update the properties of an existing repository object, the client can pass a DataObject that has an ObjectIdentity that identifies it as the existing object, and just those properties that require update. This keeps the data object passed to the update service as small as possible. If the client wants to test whether the updates have been applied by examining the DataPackage object returned by the update operation, it will need to use a PropertyProfile to instruct the service to return all properties. Otherwise the update operation will by default return DataObject instances with only an ObjectIdentity.

The following example updates the properties of an existing document. It passes a PropertyProfile object in operationOptions, causing the update operation to return all properties. It creates a new DataObject with an ObjectIdentity mapping it to an existing document in the repository, and passes this new DataObject to the update operation.

Example 4-26. Java: Updating properties

```java
public DataPackage updateObjectProperties(ObjectIdentity objectIdentity,
                                          String newTitle,
                                          String newSubject,
                                          String[] newKeywords)
    throws ServiceException
{
    PropertyProfile propertyProfile = new PropertyProfile();
    propertyProfile.setFilterMode(PropertyFilterMode.ALL);
    OperationOptions operationOptions = new OperationOptions();
    ```
operationOptions.setProfile(propertyProfile);

DataObject dataObject = new DataObject(objectIdentity);

PropertySet properties = new PropertySet();
properties.set("title", newTitle);
properties.set("subject", newSubject);
properties.set("keywords", newKeywords);
dataObject.setProperties(properties);

try {
    return objectService.update(new DataPackage(dataObject), operationOptions);
} catch (ServiceException sE) {
    sE.printStackTrace();
    return null;
}

Example 4-27. C#: Updating object properties

```csharp
public DataPackage UpdateObjectProperties(ObjectIdentity objectIdentity,
                                             String newTitle,
                                             String newSubject,
                                             String[] newKeywords)
{
    PropertyProfile propertyProfile = new PropertyProfile();
    propertyProfile.FilterMode = PropertyFilterMode.ALL;
    OperationOptions operationOptions = new OperationOptions();
    operationOptions.SetProfile(propertyProfile);

    DataObject dataObject = new DataObject(objectIdentity);

    PropertySet properties = new PropertySet();
    properties.set("title", newTitle);
    properties.set("subject", newSubject);
    properties.set("keywords", newKeywords);
    dataObject.Properties = properties;

    return objectService.Update(new DataPackage(dataObject), operationOptions);
}
```

Modifying a repeating properties (attributes) list

In some cases your client may need to make a specific change to a list of repeating properties (also called repeating attributes), such as appending values to the end of the list, inserting an item into the list, or removing an item from the list. To accomplish this you can add one or more ValueAction instances to the ArrayProperty.

A ValueAction list is synchronized with the ArrayProperty that contains it, such that an item in position p in the ValueAction list corresponds to a value stored at position p of the ArrayProperty. In this example the first item in the ValueAction list (INSERT, 0) corresponds to the first item in the ArrayProperty (snakes). The index value (0) specifies a position in the repeating property of the repository object.
Note that if you insert or delete items in a repeated properties list, the positions of items to the right of the alteration will be offset by 1 or -1. This will affect subsequent processing of the ValueAction list, which is processed from beginning to end. You must account for this effect when coding a ValueAction list, such as by ensuring that the repeating properties list is processed from right to left.

**Example 4-28. Java: Modifying repeating properties**

```java
public DataPackage updateRepeatProperty(ObjectIdentity objectIdentity)
    throws ServiceException
{
    PropertyProfile propertyProfile = new PropertyProfile();
    propertyProfile.setFilterMode(PropertyFilterMode.ALL);
    serviceContext.setProfile(propertyProfile);

    DataObject dataObject = new DataObject(objectIdentity);

    String[] moreDangers = {"snakes", "sharks"};
    ArrayProperty keywordProperty = new StringArrayProperty("keywords", moreDangers);

    ValueAction appendAction = new ValueAction(ValueActionType.INSERT, 0);
    ValueAction deleteAction = new ValueAction(ValueActionType.APPEND, 1);
    keywordProperty.setValueActions(appendAction, deleteAction);
    PropertySet properties = new PropertySet();
    properties.set(keywordProperty);
    dataObject.setProperties(properties);

    OperationOptions operationOptions = null;
    return objectService.update(new DataPackage(dataObject), operationOptions);
}
```

**Example 4-29. C#: Modifying repeating properties**

```csharp
public DataPackage UpdateRepeatProperty(ObjectIdentity objectIdentity)
{
    PropertyProfile propertyProfile = new PropertyProfile();
    propertyProfile.FilterMode = PropertyFilterMode.ALL;
    DemoServiceContext.SetProfile(propertyProfile);

    DataObject dataObject = new DataObject(objectIdentity);

    String[] moreDangers = {"snakes", "sharks"};
    ArrayProperty<
```
Updating object relationships

If the client adds a Relationship object to a DataObject passed to the update operation, the processing of the Relationship object depends on two factors:

- Whether the Relationship is an ObjectRelationship (which contains a DataObject) or a ReferenceRelationship (which contains only an ObjectIdentity).
- Whether the Relationship object represents an existing object in the repository.

If the Relationship object is an ObjectRelationship, the update operation will update an existing repository object represented by the ObjectRelationship, or create a new repository object if no such repository object exists. If the Relationship object is a ReferenceRelationship, the update operation will create a relationship (by modifying repository metadata) between the repository object represented by the DataObject and an existing repository object referenced by the ReferenceRelationship.

To remove a relationship, rather than add it, you can set the RelationshipIntentModifier to REMOVE (otherwise it is implicitly set to ADD).

To illustrate, the following example unlinks a document from one folder and links it into another folder.

**Example 4-30. Java: Updating and re-linking a folder**

```java
public DataPackage updateRelinkFolder(ObjectIdentity docId, 
                                      ObjectIdentity sourceFolderId, 
                                      ObjectIdentity targetFolderId)
  throws ServiceException
{
    DataObject docDataObj = new DataObject(docId, "dm_document");

    // add the source folder as a parent relationship of the document
    ReferenceRelationship removeRelationship = new ReferenceRelationship();
    removeRelationship.setTargetRole(Relationship.ROLE_PARENT);
    removeRelationship.setName(Relationship.RELATIONSHIP_FOLDER);
    removeRelationship.setTarget(sourceFolderId);
    docDataObj.getRelationships().add(removeRelationship);

    // specify that the folder is to be unlinked
    removeRelationship.setIntentModifier(RelationshipIntentModifier.REMOVE);

    // add the folder into which to link document
    ReferenceRelationship addRelationship = new ReferenceRelationship();
    addRelationship.setTargetRole(Relationship.ROLE_PARENT);
    addRelationship.setName(Relationship.RELATIONSHIP_FOLDER);
    addRelationship.setTarget(targetFolderId);
    docDataObj.getRelationships().add(addRelationship);

    OperationOptions operationOptions = null;
    return objectService.update(new DataPackage(docDataObj), operationOptions);
}
```

**Example 4-31. C#: Updating and re-linking a folder**

```csharp
public DataPackage UpdateRelinkFolder(ObjectIdentity docId, 
                                      ObjectIdentity sourceFolderId, 
                                      ObjectIdentity targetFolderId)
{
    DataObject docDataObj = new DataObject(docId, "dm_document");

    // add the source folder as a parent relationship of the document
```
ReferenceRelationship removeRelationship = new ReferenceRelationship();
removeRelationship.TargetRole = Relationship.ROLE_PARENT;
removeRelationship.Name = Relationship.RELATIONSHIP_FOLDER;
removeRelationship.Target = sourceFolderId;
docDataObj.Relationships.Add(removeRelationship);

// specify that the folder is to be unlinked
removeRelationship.IntentModifier = RelationshipIntentModifier.REMOVE;

// add the folder into which to link document
ReferenceRelationship addRelationship = new ReferenceRelationship();
addRelationship.TargetRole = Relationship.ROLE_PARENT;
addRelationship.Name = Relationship.RELATIONSHIP_FOLDER;
addRelationship.Target = targetFolderId;
docDataObj.Relationships.Add(addRelationship);

OperationOptions operationOptions = null;
return objectService.Update(new DataPackage(docDataObj), operationOptions);
}

For more information on the use of IntentModifier, see Removing object relationships, page 50.

delete operation

Description

The Object service delete operation deletes a set of objects from the repository. By default, for each object that it deletes, it deletes all versions. The specific behaviors of the delete operation are controlled by a DeleteProfile, which should be passed to the operation as part of OperationOptions.

Java Syntax

void delete(ObjectIdentitySet objectsToDelete,
operationOptions OperationOptions) throws ServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectsToDelete</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify repository objects to be deleted.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>An object containing profiles and properties that specify operation behaviors. If this object is null, default operation behaviors will take effect.</td>
</tr>
</tbody>
</table>
DeleteProfile

The DeleteProfile, normally passed within OperationOptions, controls specific behaviors of the delete operation. The following table describes the profile settings.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDeepDeleteFolders</td>
<td>If true, deletes all folders under a folder specified in objectsToDelete. This setting does not specify whether non-folder objects that are linked into other folders are deleted from the repository. Default value is false.</td>
</tr>
<tr>
<td>isDeepDeleteChildrenInFolders</td>
<td>If true, for each folder specified in objectsToDelete, removes all objects descended from the folder from the repository. However, this setting does not specify whether child objects of virtual documents that reside in other folders are removed from the repository. Default value is false.</td>
</tr>
<tr>
<td>isDeepDeleteVdmInFolders</td>
<td>If true, for each folder specified in objectsToDelete, removes all virtual document children descended from virtual documents residing in the folder tree, even if the child objects of the virtual document reside in folders outside the folder tree descended from the specified folder. Default value is false.</td>
</tr>
<tr>
<td>versionStrategy</td>
<td>Determines the behavior or the delete operation as pertains to versions, using a value of the DeleteVersionStrategy enum. Possible values are SELECTED_VERSIONS, UNUSED_VERSIONS, ALL_VERSIONS. Default value is ALL_VERSIONS.</td>
</tr>
<tr>
<td>isPopulateWithReferences</td>
<td>Specifies whether reference objects should be dereferenced during population, that is, when files/objects are added to the operation. True will indicate that the reference objects themselves will be added to the operation. False will indicate that reference objects will be dereferenced and the remote object will be added to the operation. The default is false.</td>
</tr>
</tbody>
</table>

Example

The following example deletes all versions of a document from the repository, as well as all descended folders and child objects residing within those folders. However, it does not delete children of virtual documents that reside in folders outside the tree descended from the specified folder.

Example 4-32. Java: Deep delete

```
public void objServiceDelete(String path) throws ServiceException
{
    ObjectPath docPath = new ObjectPath(path);
    ObjectIdentity<ObjectPath> objIdentity = new ObjectIdentity<ObjectPath>();
    objIdentity.setValue(docPath);
    objIdentity.setRepositoryName(defaultRepositoryName);
```
ObjectIdentitySet objectIdSet = new ObjectIdentitySet(objIdentity);

DeleteProfile deleteProfile = new DeleteProfile();
deleteProfile.setDeepDeleteFolders(true);
deleteProfile.setDeepDeleteChildrenInFolders(true);
OperationOptions operationOptions = new OperationOptions();
operationOptions.setDeleteProfile(deleteProfile);

objectService.delete(objectIdSet, operationOptions);

Example 4-33. C#: Deep delete

```csharp
public void ObjServiceDelete(String path)
{
    ObjectPath docPath = new ObjectPath(path);
    ObjectIdentity objIdentity = new ObjectIdentity();
    objIdentity.Value = docPath;
    objIdentity.RepositoryName = DefaultRepository;
    ObjectIdentitySet objectIdSet = new ObjectIdentitySet(objIdentity);

    DeleteProfile deleteProfile = new DeleteProfile();
deleteProfile.IsDeepDeleteFolders = true;
deleteProfile.IsDeepDeleteChildrenInFolders = true;
OperationOptions operationOptions = new OperationOptions();
operationOptions.DeleteProfile = deleteProfile;

    objectService.Delete(objectIdSet, operationOptions);
}
```

copy operation

Description

The copy operation copies a set of repository objects from one location to another, either within a single repository, or from one repository to another. During the copy operation, the service can optionally make modifications to the objects being copied.

Note: For the service to copy an object from one repository to another, the ServiceContext must be set up to provide the service with access to both repositories. This can be done by setting up a separate RepositoryIdentity for each repository, or by use of a BasicIdentity, which provides default user credentials for multiple repositories. For more information on RepositoryIdentity and BasicIdentity, see Identities, page 57.

Java Syntax

```java
DataPackage copy(ObjectIdentitySet fromObjects,
          ObjectLocation targetLocation,
          DataPackage modifyObjects,
          OperationOptions operationOptions)
throws CoreServiceException
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromObjects</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that identify the repository objects to be copied.</td>
</tr>
<tr>
<td>targetLocation</td>
<td>ObjectLocation</td>
<td>Contains an ObjectIdentity that identifies the location (a cabinet or folder) into which the repository objects are to be copied.</td>
</tr>
<tr>
<td>modifyObjects</td>
<td>DataPackage</td>
<td>Optionally contains a set of DataObject instances that contain modifications (such as changes to property values, content, or relationships) to all or some of the repository objects being copied. The ObjectIdentity of each DataObject must uniquely identify one of the copied objects. The modifications supplied in the DataObject are applied during the copy operation.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>An object containing profiles and properties that specify operation behaviors.</td>
</tr>
</tbody>
</table>

CopyProfile

The CopyProfile, normally passed within OperationOptions, controls specific behaviors of the copy operation. The following table describes the profile settings.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isDeepCopyFolders</td>
<td>If true, copies all folders and their contents descended from any folder specified in fromObjects. Default value is false.</td>
</tr>
<tr>
<td>isNonCurrentObjectsAllowed</td>
<td>If true, allows copy of non-CURRENT objects; otherwise throws an exception on attempt to copy non-CURRENT object. Default value is false.</td>
</tr>
</tbody>
</table>

Response

Returns a DataPackage containing one DataObject for each repository object created by the copy operation. By default, each DataObject contains only the ObjectIdentity of the created object and no other data. The client can modify this behavior by using Profile objects if it requires more data about the copied objects.
Examples

The following examples demonstrate:

- Copy across repositories, page 82
- Copy with modifications, page 83

Copy across repositories

The following example copies a single object to a secondary repository. Note that the service context must contain Identity instances that provide the service with access credentials to both repositories. For more information see Identities, page 57.

Example 4-34. Java: Copy across repositories

```java
public void objServiceCopyAcrossRepositories(String sourceObjectPathString, String targetLocPathString)
    throws ServiceException
{
    // identify the object to copy
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity<ObjectPath> docToCopy = new ObjectIdentity<ObjectPath>();
    docToCopy.setValue(objPath);
    docToCopy.setRepositoryName(defaultRepositoryName);

    // identify the folder to copy to
    ObjectPath folderPath = new ObjectPath();
    folderPath.setPath(targetLocPathString);
    ObjectIdentity<ObjectPath> toFolderIdentity = new ObjectIdentity<ObjectPath>();
    toFolderIdentity.setValue(folderPath);
    toFolderIdentity.setRepositoryName(secondaryRepositoryName);
    ObjectLocation toLocation = new ObjectLocation();
    toLocation.setObjectIdentity(toFolderIdentity);

    OperationOptions operationOptions = null;
    objectService.copy(new ObjectIdentitySet(docToCopy), toLocation, null, operationOptions);
}
```

Example 4-35. C#: Copy across repositories

```c#
public void ObjServiceCopyAcrossRepositories(String sourceObjectPathString, String targetLocPathString)
{
    // identify the object to copy
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity docToCopy = new ObjectIdentity();
    docToCopy.Value = objPath;
    docToCopy.RepositoryName = DefaultRepository;

    // identify the folder to copy to
    ObjectPath folderPath = new ObjectPath();
    folderPath.Path = targetLocPathString;
    ObjectIdentity toFolderIdentity = new ObjectIdentity();
    toFolderIdentity.Value = folderPath;
    toFolderIdentity.RepositoryName = SecondaryRepository;
    ObjectLocation toLocation = new ObjectLocation();
    toLocation.Identity = toFolderIdentity;
}
OperationOptions operationOptions = null;
objectService.Copy(new ObjectIdentitySet(docToCopy), toLocation, null, operationOptions);
}

Copy with modifications

The following sample copies a document to a new location, and at the same time changes its object_name property.

Example 4-36. Java: Copy with modifications

```java
public void objServiceCopyWithMods(String sourceObjectPathString,
                                      String targetLocPathString)
    throws ServiceException
{
    // identify the object to copy
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity/ObjectPath> docToCopy = new ObjectIdentity/ObjectPath();
    docToCopy.setValue(objPath);
    docToCopy.setRepositoryName(defaultRepositoryName);

    // identify the folder to copy to
    ObjectPath folderPath = new ObjectPath();
    folderPath.setPath(targetLocPathString);
    ObjectIdentity/ObjectPath> toFolderIdentity = new ObjectIdentity/ObjectPath();
    toFolderIdentity.setValue(folderPath);
    toFolderIdentity.setRepositoryName(defaultRepositoryName);
    ObjectLocation toLocation = new ObjectLocation();
    toLocation.setObjectIdentity(toFolderIdentity);

    // specify changes to make when copying
    DataObject modDataObject = new DataObject(docToCopy);
    modDataObject.setObject("dm_document");
    PropertySet modProperties = modDataObject.getProperties();
    modProperties.set("object_name", "copiedDocument-" + System.currentTimeMillis());
    DataPackage dataPackage = new DataPackage(modDataObject);

    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(docToCopy);
    OperationOptions operationOptions = null;
    objectService.copy(objIdSet, toLocation, dataPackage, operationOptions);
}
```

Example 4-37. C#: Copy with modifications

```csharp
public void ObjServiceCopyWithMods(String sourceObjectPathString,
                                      String targetLocPathString)
{
    // identify the object to copy
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity docToCopy = new ObjectIdentity();
    docToCopy.Value = objPath;
    docToCopy.RepositoryName = DefaultRepository;

    // identify the folder to copy to
    ObjectPath folderPath = new ObjectPath();
    folderPath.Path = targetLocPathString;
    ObjectIdentity toFolderIdentity = new ObjectIdentity();
    toFolderIdentity.Value = folderPath;
    toFolderIdentity.RepositoryName = DefaultRepository;
    ObjectLocation toLocation = new ObjectLocation();
```
toLocation.Identity = toFolderIdentity;

// specify changes to make when copying
DataObject modDataObject = new DataObject(docToCopy);
modDataObject.Type = "dm_document";
PropertySet modProperties = modDataObject.Properties;
modProperties.Set("object_name", "copiedDocument-" + System.DateTime.Now.Ticks);
DataPackage dataPackage = new DataPackage(modDataObject);

ObjectIdentitySet objIdSet = new ObjectIdentitySet();
objIdSet.Identities.Add(docToCopy);
OperationOptions operationOptions = null;
objectService.Copy(objIdSet, toLocation, dataPackage, operationOptions);
}

move operation

Description

The move operation moves a set of repository objects from one location to another within a repository, and provides the optional capability of updating the repository objects as they are moved. The move operation will only move the CURRENT version of an object, unless non-CURRENT objects are specifically permitted by a MoveProfile. By default, if passed an ObjectIdentity that identifies a non-CURRENT object, the operation will throw an exception.

Note: Moving an object across repositories is not supported in DFS version 6.

Java syntax

```java
DataPackage move(ObjectIdentitySet fromObjects,
                  ObjectLocation sourceLocation,
                  ObjectLocation targetLocation,
                  DataPackage modifyObjects
                  OperationOptions operationOptions) throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromObjects</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that identify the repository objects to be moved.</td>
</tr>
<tr>
<td>sourceLocation</td>
<td>ObjectLocation</td>
<td>Contains an ObjectIdentity that identifies the location (a cabinet or folder) from which the repository objects are to be moved.</td>
</tr>
<tr>
<td>targetLocation</td>
<td>ObjectLocation</td>
<td>Contains an ObjectIdentity that identifies the location (a cabinet or folder) into which the repository objects are to be moved.</td>
</tr>
</tbody>
</table>
Object Service

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>modifyObjects</td>
<td>DataPackage</td>
<td>Optionally contains a set of DataObject instances that contain modifications (such as changes to property values, content, or relationships) to all or some of the repository objects being moved. The ObjectIdentity of each DataObject must uniquely identify one of the moved objects. The modifications supplied in the DataObject are applied during the move operation.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. operationOptions can contain a MoveProfile, which provides options specific to this operation.</td>
</tr>
</tbody>
</table>

### MoveProfile

The MoveProfile, normally passed within OperationOptions, controls specific behaviors of the move operation. The following table describes the profile settings.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isNonCurrentObjectsAllowed</td>
<td>If true, allows move of non-CURRENT objects; otherwise throws an exception on attempt to move non-CURRENT object. Default value is false.</td>
</tr>
</tbody>
</table>

### Response

Returns a DataPackage containing one DataObject for each repository object created by the move operation. By default, each DataObject contains only the ObjectIdentity of the created object and no other data. The client can modify this behavior by using Profile objects if it requires more data about the moved objects.

### Example

**Example 4-38. Java: Moving an object**

```java
public void objServiceMove(String sourceObjectPathString, String targetLocPathString, String sourceLocPathString) throws ServiceException {
    // identify the object to move
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity<ObjectPath> docToCopy = new ObjectIdentity<ObjectPath>();
    docToCopy.setValue(objPath);
    docToCopy.setRepositoryName(defaultRepositoryName);
```

EMC Documentum Foundation Services Version 6 SP1 Development Guide 85
// identify the folder to move from
ObjectPath fromFolderPath = new ObjectPath();
fromFolderPath.setPath(sourceLocPathString);
ObjectIdentity<ObjectPath> fromFolderIdentity = new ObjectIdentity<ObjectPath>();
fromFolderIdentity.setValue(fromFolderPath);
fromFolderIdentity.setRepositoryName(defaultRepositoryName);
ObjectLocation fromLocation = new ObjectLocation();
fromLocation.setObjectIdentity(fromFolderIdentity);

// identify the folder to move to
ObjectPath folderPath = new ObjectPath();
folderPath.setPath(targetLocPathString);
ObjectIdentity<ObjectPath> toFolderIdentity = new ObjectIdentity<ObjectPath>();
toFolderIdentity.setValue(folderPath);
toFolderIdentity.setRepositoryName(defaultRepositoryName);
ObjectLocation toLocation = new ObjectLocation();
toLocation.setObjectIdentity(toFolderIdentity);

OperationOptions operationOptions = null;
objectService.move(new ObjectIdentitySet(docToCopy),
                   fromLocation,
                   toLocation,
                   new DataPackage(),
                   operationOptions);

Example 4-39. C#: Moving an object

public void ObjServiceMove(String sourceObjectPathString,
                            String targetLocPathString,
                            String sourceLocPathString)
{
    // identify the object to move
    ObjectPath objPath = new ObjectPath(sourceObjectPathString);
    ObjectIdentity docToCopy = new ObjectIdentity();
    docToCopy.Value = objPath;
    docToCopy.RepositoryName = DefaultRepository;

    // identify the folder to move from
    ObjectPath fromFolderPath = new ObjectPath();
    fromFolderPath.Path = sourceLocPathString;
    ObjectIdentity fromFolderIdentity = new ObjectIdentity();
    fromFolderIdentity.Value = fromFolderPath;
    fromFolderIdentity.RepositoryName = DefaultRepository;
    ObjectLocation fromLocation = new ObjectLocation();
    fromLocation.Identity = fromFolderIdentity;

    // identify the folder to move to
    ObjectPath folderPath = new ObjectPath(targetLocPathString);
    ObjectIdentity toFolderIdentity = new ObjectIdentity();
    toFolderIdentity.Value = folderPath;
    toFolderIdentity.RepositoryName = DefaultRepository;
    ObjectLocation toLocation = new ObjectLocation();
    toLocation.Identity = toFolderIdentity;

    OperationOptions operationOptions = null;
    objectService.Move(new ObjectIdentitySet(docToCopy),
                        fromLocation,
                        toLocation,
                        new DataPackage(),
                        operationOptions);
}
validate operation

The validate operation validates a set of DataObject instances against repository data dictionary (schema) rules, testing whether the DataObject instances represent valid repository objects, and whether the DataObject properties represent valid repository properties.

Java syntax

ValidationInfoSet validate(DataPackage dataPackage) throws CoreServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataPackage</td>
<td>DataPackage</td>
<td>A collection of DataObject instances to be validated by the operation.</td>
</tr>
</tbody>
</table>

Response

Returns a ValidationInfoSet, which contains a list of ValidationInfo objects. Each ValidationInfo contains a DataObject and a list of any ValidationIssue instances that were raised by the operation. A ValidationIssue can be of enum type ERROR, UNDEFINED, or WARNING. Figure 12, page 88 shows the ValidationInfoSet model.
**getObjectContentUrls operation**

**Description**

The `getObjectContentUrls` operation gets a set of `UrlContent` objects based on a set of `ObjectIdentity` instances.

**Java syntax**

```java
List<ObjectContentSet> getObjectContentUrls(ObjectIdentitySet forObjects)
throws CoreServiceException
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forObjects</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances for which to obtain UrlContent objects.</td>
</tr>
</tbody>
</table>
Response

Returns a list of ObjectContentSet objects, each of which contains a list of UrlContent objects. Note that more than one UrlContent can be returned for each ObjectIdentity. Additional Content instances represent renditions of the repository object.
Chapter 5

VersionControl Service

The VersionControl service provides operations that enable access and changes to specific object versions. This chapter covers the following topics:

- getCheckoutInfo operation, page 91
- checkout operation, page 93
- checkin operation, page 95
- cancelCheckout operation, page 99
- deleteVersion operation, page 100
- deleteAllVersions operation, page 101
- getCurrent operation, page 102
- getVersionInfo operation, page 104

getCheckoutInfo operation

Description

Provides checkout information about the specified objects, specifically whether the objects are checked out, and the user name of the user who has them checked out.

Java syntax

```java
List<CheckoutInfo> getCheckoutInfo(ObjectIdentitySet objectIdentitySet)
    throws CoreServiceException
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectIdentitySet</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects about which to obtain checkout information.</td>
</tr>
</tbody>
</table>

Response

Returns a List of CheckoutInfo instances. Checkout info encapsulates data about a specific checked out repository object. The following table shows the CheckoutInfo fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>ObjectIdentity</td>
<td>Uniquely identifies the checked out object.</td>
</tr>
<tr>
<td>userName</td>
<td>String</td>
<td>The name of the user who has the object checked out.</td>
</tr>
<tr>
<td>isCheckedOut</td>
<td>boolean</td>
<td>Indicates whether the repository object is checked out.</td>
</tr>
</tbody>
</table>

Example

The following example gets checkout information about an object and prints it to the console.

Example 5-1. Java: Getting checkout info

```java
public CheckoutInfo checkoutInfo(ObjectIdentity objIdentity)
    throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc
        = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);

    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);
    List<CheckoutInfo> objList =
        versionSvc.checkout(objIdSet, operationOptions);
    CheckoutInfo checkoutInfo = objList.get(0);

    if (checkoutInfo.isCheckedOut())
    {
        System.out.println("Object "+checkoutInfo.getIdentity()
            + " is checked out.");
        System.out.println("Lock owner is "+checkoutInfo.getUserName());
    }
    else
    {
        System.out.println("Object "+checkoutInfo.getIdentity()
            + " is not checked out.");
    }
}"
```
Example 5-2. C#: Getting checkout info

```csharp
public CheckoutInfo CheckoutInfo(ObjectIdentity objIdentity)
{
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);
    List<CheckoutInfo> objList;
    OperationOptions operationOptions = null;
    versionControlService.Checkout(objIdSet, operationOptions);
    objList = versionControlService.GetCheckoutInfo(objIdSet);
    CheckoutInfo checkoutInfo = objList[0];

    if (checkoutInfo.IsCheckedOut)
    {
        Console.WriteLine("Object " + checkoutInfo.Identity + " is checked out.");
        Console.WriteLine("Lock owner is " + checkoutInfo.UserName);
    }
    else
    {
        Console.WriteLine("Object " + checkoutInfo.Identity + " is not checked out.");
    }

    versionControlService.CancelCheckout(objIdSet);
    return checkoutInfo;
}
```

## checkout operation

### Description

The checkout operation checks out a set of repository objects. Any version of the object can be checked out.

The checkout operation by default returns no content and no properties. These defaults can be changed using ContentProfile and PropertyProfile instances passed in OperationOptions or set in the service context.

### Java syntax

```
DataPackage checkout(ObjectIdentitySet objectIdentitySet,
                      OperationOptions operationOptions) throws CoreServiceException
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectIdentitySet</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects to check out.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of the checkout operation, the profiles primarily provide filters that modify the contents of the returned DataPackage.</td>
</tr>
</tbody>
</table>

Response

Returns a DataPackage containing DataObject instances representing the checked out repository objects. The DataObject instances contain complete properties, and any object content is transferred. The client can change these defaults by setting Profile instances in OperationOptions.

Example

Example 5-3. Java: Checking an object out

```java
public DataPackage checkout(ObjectIdentity objIdentity) throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);

    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);

    OperationOptions operationOptions = null;
    DataPackage resultDp;
    try
    {
        resultDp = versionSvc.checkout(objIdSet, operationOptions);
    }
    catch (Exception e)
    {
        e.printStackTrace();
        throw new RuntimeException(e);
    }

    System.out.println("Checkout successful");

    List<VersionInfo> vInfo = versionSvc.getVersionInfo(objIdSet);
    VersionInfo versionInfo = vInfo.get(0);

    System.out.println("Printing version info for " + versionInfo.getIdentity());
    System.out.println("isCurrent is " + versionInfo.isCurrent());
    System.out.println("Version is " + versionInfo.getVersion());
    System.out.println("Symbolic labels are: ");
    for (String label : versionInfo.getSymbolicLabels())
    {
```
Example 5-4. C#: Checking an object out

```csharp
public DataPackage Checkout(ObjectIdentity objIdentity)
{
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);

    OperationOptions operationOptions = null;
    DataPackage resultDp;

    resultDp = versionControlService.Checkout(objIdSet, operationOptions);
    Console.WriteLine("Checkout successful");

    List<VersionInfo> vInfo = versionControlService.GetVersionInfo(objIdSet);
    VersionInfo versionInfo = vInfo[0];

    Console.WriteLine("Printing version info for " + versionInfo.Identity);
    Console.WriteLine("IsCurrent is " + versionInfo.IsCurrent);
    Console.WriteLine("Version is " + versionInfo.Version);
    Console.WriteLine("Symbolic labels are: ");
    foreach (String label in versionInfo.SymbolicLabels)
    {
        Console.WriteLine(label);
    }

    versionControlService.CancelCheckout(objIdSet);
    Console.WriteLine("Checkout cancelled");
    return resultDp;
}
```

checkin operation

Description

The checkin operation checks in a set of repository objects using data contained in a DataPackage. It provides control over how the checked in object is versioned and whether the object remains checked out and locked by the user after the changes are versioned, and provides a mechanism for applying symbolic version labels to the checked-in versions. The ObjectIdentity of each DataObject passed to the operation is expected to match the identity of a checked out repository object.
Java syntax

```java
DataPackage checkin(DataPackage dataPackage,
    VersionStrategy versionStrategy,
    boolean isRetainLock,
    List<String> symbolicLabels
    OperationOptions operationOptions) throws CoreServiceException
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dataPackage</td>
<td>DataPackage</td>
<td>Contains a set of DataObject instances that are to be checked in as new versions of checked out repository objects.</td>
</tr>
<tr>
<td>versionStrategy</td>
<td>VersionStrategy</td>
<td>Specifies option for incrementing the version number of the new version.</td>
</tr>
<tr>
<td>isRetainLock</td>
<td>boolean</td>
<td>Specifies whether the object is to remain checked out and locked by the user after the new version is saved.</td>
</tr>
<tr>
<td>symbolicLabels</td>
<td>List&lt;String&gt;</td>
<td>A list of symbolic version labels, which are applied to all repository objects represented in the DataPackage.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of the checkout operation, the profiles primarily provide filters that modify the contents of the returned DataPackage.</td>
</tr>
</tbody>
</table>

### VersionStrategy values

The VersionStrategy values represent the numbering strategy that is applied to a new repository object version when it is checked in.

<table>
<thead>
<tr>
<th>TargetRole value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IMPLIED</td>
<td>Use the default behavior configured on Content Server for versioning. This is typically to check the object in as a new version and to increment the version number as the next minor version.</td>
</tr>
<tr>
<td>NEXT_MINOR</td>
<td>Check the object in as a new version and to increment the version number as the next minor version. For example, if the version number is currently 1.1, give the new version the number 1.2.</td>
</tr>
<tr>
<td>NEXT_MAJOR</td>
<td>Check the object in as a new version and to increment the version number as the next major version. For example, if the version number is currently 1.1, give the new version the number 2.0.</td>
</tr>
<tr>
<td>SAME_VERSION</td>
<td>Save the new object as the same version as the current version, overwriting the current version. This requires that the user have WRITE permissions on the object.</td>
</tr>
</tbody>
</table>
CheckinProfile

The CheckinProfile, normally passed within OperationOptions, controls specific behaviors of the checkin operation. The following table describes the profile settings.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isKeepFileLocal</td>
<td>If true, does not remove the local file from the client when checking in to the repository. Default value is false.</td>
</tr>
<tr>
<td>isMakeCurrent</td>
<td>If true, makes the checked in version the CURRENT version. Default value is true.</td>
</tr>
<tr>
<td>isDeleteLocalFileHint</td>
<td>If using UCF content transfer, delete local file content after checkin to repository. Default value is false. This hint will not be honored if content transfer mode is not UCF. If ContentTransferMode is MTOM or base64, the local file is never deleted.</td>
</tr>
</tbody>
</table>

Response

Returns a DataPackage containing one DataObject for each repository object version created by the checkin operation. By default, each DataObject contains only the ObjectIdentity of the new version and no other data. The client can modify this behavior by using Profile objects if it requires more data about the new versions.

Example

The following example checks in a single DataObject as a new version. Note that it explicitly sets a ContentProfile for the that is applied on checkout and subsequent checkin. Note as well that new content is explicitly added to the object prior to checkin.

Example 5-5. Java: Checking an object in

```java
public DataPackage checkin(ObjectIdentity objIdentity, String newContentPath) throws Exception {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc = serviceFactory.getService(IVersionControlService.class, serviceContext);
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);

    OperationOptions operationOptions = new OperationOptions();
    ContentProfile contentProfile = new ContentProfile(FormatFilter.ANY, null,
                                                        PageFilter.ANY,
                                                        -1,
                                                        PageModifierFilter.ANY, null);

    operationOptions.setContentProfile(contentProfile);

    DataPackage checkinPackage = versionSvc.checkout(objIdSet, operationOptions);
```
DataObject checkinObj = checkinPackage.getDataObjects().get(0);
checkinObj.setContents(null);
FileContent newContent = new FileContent();
newContent.setLocalPath(newContentPath);
newContent.setRenditionType(RenditionType.PRIMARY);
newContent.setFormat("gif");
checkinObj.getContents().add(newContent);

boolean retainLock = false;
List<String> labels = new ArrayList<String>();
labels.add("test_version");
DataPackage resultDp;
try {
    resultDp = versionSvc.checkin(checkinPackage,
        VersionStrategy.NEXT_MINOR,
        retainLock,
        labels,
        operationOptions);
}
catch (ServiceException sE) {
    sE.printStackTrace();
    throw new RuntimeException(sE);
}
return resultDp;

Example 5-6. C#: Checking an object in

```csharp
public DataPackage Checkin(ObjectIdentity objIdentity, String newContentPath)
{
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);

    OperationOptions operationOptions = new OperationOptions();
    ContentProfile contentProfile = new ContentProfile(FormatFilter.ANY, null,
        PageFilter.ANY, -1,
        PageModifierFilter.ANY, null);
    operationOptions.ContentProfile = contentProfile;

    DataPackage checkinPackage = versionControlService.Checkout(objIdSet, operationOptions);

    DataObject checkinObj = checkinPackage.DataObjects[0];
    checkinObj.Contents = null;
    FileContent newContent = new FileContent();
    newContent.LocalPath = newContentPath;
    newContent.RenditionType = RenditionType.PRIMARY;
    newContent.Format = "gif";
    checkinObj.Contents.Add(newContent);

    bool retainLock = false;
    List<String> labels = new List<String>();
    labels.Add("test_version");
    DataPackage resultDp;
    try {
        resultDp = versionControlService.Checkin(checkinPackage,
            VersionStrategy.NEXT_MINOR,
            retainLock,
        
```
cancelCheckout operation

Description

The cancelCheckout operation cancels checkout of a set of repository objects.

Java syntax

```java
void cancelCheckout(ObjectIdentitySet objectIdentitySet) throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectIdentitySet</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects on which to cancel checkout.</td>
</tr>
</tbody>
</table>

Example

**Example 5-7. Java: Cancelling checkout**

```java
public void cancelCheckout(ObjectIdentity objIdentity) throws ServiceException {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);
    versionSvc.cancelCheckout(objIdSet);
}
```
Example 5-8. C#: Cancelling checkout

```csharp
public void CancelCheckout(ObjectIdentity objIdentity)
{
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);
    versionControlService.CancelCheckout(objIdSet);
}
```

deleteVersion operation

**Description**

The deleteVersion operation deletes a specific version of a repository object. If the deleted object is the CURRENT version, the previous version in the version tree is promoted to CURRENT.

**Java syntax**

```java
void deleteVersion(ObjectIdentitySet objectsToDelete) throws CoreServiceException
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectsToDelete</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository object versions to delete.</td>
</tr>
</tbody>
</table>

**Example**

The following sample deletes a specific version of a repository object. The ObjectIdentity representing the repository object can be an ObjectId or a Qualification that identifies a non-CURRENT version.

**Example 5-9. Java: Deleting a specific version**

```java
public void deleteVersionDemo(ObjectIdentity objIdentity)
    throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc
        = serviceFactory.getRemoteService(IVersionControlService.class,
        serviceContext);

    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);
```
Example 5-10. C#: Deleting a specific version

```csharp
public void DeleteVersionDemo(ObjectIdentity objIdentity)
{
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);
    versionControlService.DeleteVersion(objIdSet);
}
```

deleteAllVersions operation

**Description**

The deleteAllVersions operation deletes all versions of a repository object. An ObjectIdentity indicating the object to delete can reference any version in the version tree.

**Java syntax**

```java
void deleteAllVersions(ObjectIdentitySet objectIdentitySet) throws CoreServiceException
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objectIdentitySet</td>
<td>ObjectIdentity</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects of which to delete all versions.</td>
</tr>
</tbody>
</table>

**Example**

The following sample deletes all versions of an object. The qualification it uses can represent a CURRENT or a non-CURRENT version.

**Example 5-11. Java: Deleting all versions of an object**

```java
public void deleteAllVersionsDemoQual() throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);
```
String nonCurrentQual = "dm_document (ALL) " +
    "where object_name = 'DFS_sample_image' " +
    "and ANY r_version_label = 'test_version'";
Qualification<String> qual = new Qualification<String>(nonCurrentQual);
ObjectIdentity<Qualification> objIdentity = new ObjectIdentity<Qualification>();
objIdentity.setValue(qual);
objIdentity.setRepositoryName(defaultRepositoryName);
ObjectIdentitySet objectIdSet = new ObjectIdentitySet();
objectIdSet.getIdentities().add(objIdentity);

versionSvc.deleteAllVersions(objectIdSet);

Example 5-12. C#: Deleting all versions of an object
public void DeleteAllVersionsDemoQual()
{
    string nonCurrentQual = "dm_document (ALL) " +
    "where object_name = 'DFS_sample_image' " +
    "and ANY r_version_label = 'test_version'";
    Qualification qual = new Qualification(nonCurrentQual);
    ObjectIdentity objIdentity = new ObjectIdentity();
    objIdentity.Value = qual;
    objIdentity.RepositoryName = DefaultRepository;
    ObjectIdentitySet objectIdSet = new ObjectIdentitySet();
    objectIdSet.Identities.Add(objIdentity);

    versionControlService.DeleteAllVersions(objectIdSet);
}

getCurrent operation

Description

The getCurrent operation exports the CURRENT version of a repository object, transferring any
object content to the client. The getCurrent operation returns the CURRENT version of a repository
object even when passed an ObjectIdentity identifying a non-CURRENT version.

By default, the getCurrent operation returns no content, and only non-system properties.
These defaults can be changed using ContentProfile and PropertyProfile instances passed in
operationOptions or set in the service context.

Java syntax

DataPackage getCurrent(ObjectIdentitySet forObjects,
    OperationOptions operationOptions)
    throws CoreServiceException
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>forObjects</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects of which the CURRENT version will be exported.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of the getCurrent operation, the profiles primarily provide filters that modify the contents of the returned DataPackage.</td>
</tr>
</tbody>
</table>

## Response

Returns a DataPackage populated using the same defaults as the Object service get operation (see Response, page 68). These defaults can be modified by setting Profile instances in operationOptions or the service context (see Controlling data returned by get operation, page 69).

## Example

**Example 5-13. Java: Getting the current object**

```java
public DataObject getCurrentDemo(ObjectIdentity objIdentity) throws ServiceException {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);
    OperationOptions operationOptions = null;
    DataPackage resultDataPackage = versionSvc.getCurrent(objIdSet, operationOptions);
    return resultDataPackage.getDataObjects().get(0);
}
```

**Example 5-14. C#: Getting the current object**

```csharp
public DataObject GetCurrentDemo(ObjectIdentity objIdentity) {
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);
    OperationOptions operationOptions = null;
    DataPackage resultDataPackage = versionControlService.GetCurrent(objIdSet, operationOptions);
    return resultDataPackage.DataObjects[0];
}
```
getVersionInfo operation

Description

The getVersionInfo operation provides information about a version of a repository object.

Java syntax

```java
List<VersionInfo> getVersionInfo(ObjectIdentitySet objectIdentitySet)
throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectIdentitySet</td>
<td>ObjectIdentitySet</td>
<td>A collection of ObjectIdentity instances that uniquely identify the repository objects about which to provide version information.</td>
</tr>
</tbody>
</table>

Response

Returns a List of VersionInfo instances corresponding to the DataObject instances in the ObjectIdentitySet.

Response

Returns a List of VersionInfo instances corresponding to the DataObject instances in the ObjectIdentitySet. Each VersionInfo contains data about a specific version of a repository object. The following table shows the VersionInfo fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity</td>
<td>ObjectIdentity</td>
<td>Uniquely identifies the object version.</td>
</tr>
<tr>
<td>isCurrent</td>
<td>boolean</td>
<td>Specifies whether this is the CURRENT version of the repository object.</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td>The system version label, for example 1.1.</td>
</tr>
<tr>
<td>symbolicLabel</td>
<td>List</td>
<td>A List of String values representing all symbolic version labels applied to this version, including (if applicable) CURRENT.</td>
</tr>
</tbody>
</table>
Example

Example 5-15. Java: Getting version info

```java
public void versionInfoDemoQual(String nonCurrentQual)
    throws ServiceException {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IVersionControlService versionSvc
        = serviceFactory.getRemoteService(IVersionControlService.class, serviceContext);

    Qualification<String> qual = new Qualification<String>(nonCurrentQual);
    ObjectIdentity<Qualification> objIdentity = new ObjectIdentity<Qualification>(qual);
    objIdentity.setRepositoryName(defaultRepositoryName);
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.getIdentities().add(objIdentity);

    List<VersionInfo> vInfo = versionSvc.getVersionInfo(objIdSet);
    VersionInfo versionInfo = vInfo.get(0);

    System.out.println("Printing version info for ", versionInfo.getIdentity());
    System.out.println("isCurrent is ", versionInfo.isCurrent());
    System.out.println("Version is ", versionInfo.getVersion());

    System.out.println("Symbolic labels are: ");
    for (String label : versionInfo.getSymbolicLabels())
    {
        System.out.println(label);
    }
}
```

Example 5-16. C#: Getting version info

```csharp
public void VersionInfoDemoQual(String nonCurrentQual)
{
    Qualification qual = new Qualification(nonCurrentQual);
    ObjectIdentity objIdentity = new ObjectIdentity();
    objIdentity.Value = qual;
    objIdentity.RepositoryName = DefaultRepository;
    ObjectIdentitySet objIdSet = new ObjectIdentitySet();
    objIdSet.Identities.Add(objIdentity);

    List<VersionInfo> vInfo = versionControlService.GetVersionInfo(objIdSet);
    VersionInfo versionInfo = vInfo[0];

    Console.WriteLine("Printing version info for ", versionInfo.Identity);
    Console.WriteLine("isCurrent is ", versionInfo.IsCurrent);
    Console.WriteLine("Version is ", versionInfo.Version);

    Console.WriteLine("Symbolic labels are: ");
    foreach (string label in versionInfo.SymbolicLabels)
    {
        Console.WriteLine(label);
    }
}
}
Chapter 6

Schema Service

The Schema service provides a mechanism for retrieving information regarding repository schemas. A schema is a formal definition of repository metadata, including types, properties, and relationships. For the current release only the DEFAULT repository schema is supported, which provides metadata information concerning the data dictionary. In future releases a repository will potentially have an arbitrary number of named schemas. The Schema service can be used for creating a data structure against which a client can perform offline validation of objects against repository metadata.

This chapter covers the following topics:

- Common schema classes, page 107
- SchemaProfile, page 110
- getSchemaInfo operation, page 111
- getRepositoryInfo operation, page 113
- getTypeInfo operation, page 115
- getPropertyInfo operation, page 117
- getDynamicAssistValues operation, page 118

Common schema classes

The following sections describe common descriptor classes used by the Schema service.

TypeInfo

The TypeInfo class is a descriptor for repository object types. For detailed information on the types themselves, refer to the EMC Documentum Object Reference.

<table>
<thead>
<tr>
<th>Property</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getName</td>
<td>String</td>
<td>The name of the repository object type.</td>
</tr>
<tr>
<td>setName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getDescription</td>
<td>String</td>
<td>A description of the repository object type.</td>
</tr>
<tr>
<td>setDescription</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Property</td>
<td>Data type</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getLabel</td>
<td>String</td>
<td>The localized displayed string for the type name.</td>
</tr>
<tr>
<td>setLabel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getDisplayInfos</td>
<td>List&lt;DisplayInfo&gt;</td>
<td>Information to display a summary of the repository object type, which consists of its name and a List of its attribute names.</td>
</tr>
<tr>
<td>setDisplayInfos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getParentName</td>
<td>String</td>
<td>The name of the parent type of this repository object type.</td>
</tr>
<tr>
<td>setParentName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getPropertyName</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>setPropertyName</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getPropertyInfos</td>
<td>List&lt;PropertyInfo&gt;</td>
<td>A List of PropertyInfo objects describing the properties of this repository object type. See PropertyInfo, page 108.</td>
</tr>
<tr>
<td>setPropertyInfos</td>
<td></td>
<td></td>
</tr>
<tr>
<td>getRelationshipInfos</td>
<td>List&lt;RelationshipInfo&gt;</td>
<td>A list of RelationshipInfo objects indicating all the relationships in which this object type can participate. See RelationshipInfo, page 110.</td>
</tr>
<tr>
<td>setRelationshipInfos</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### PropertyInfo

The PropertyInfo class is a descriptor for a repository property (also called attribute).

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The property name.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Description of the property.</td>
</tr>
<tr>
<td>helpText</td>
<td>String</td>
<td>Help text to display in UI for this property.</td>
</tr>
<tr>
<td>searchOperations</td>
<td>List&lt;SearchOperation&gt;</td>
<td>A List of search operations allowed against this Property. Refer to the Javadoc for documentation of the PropertyInfo.SearchOperation enum constants.</td>
</tr>
<tr>
<td>dataType</td>
<td>DataType</td>
<td>The repository data type of this Property. Possible values are BOOLEAN, CUSTOM, DATE, DOUBLE, INTEGER, LONG, SHORT, OBJECT_ID, STRING.</td>
</tr>
<tr>
<td>defaultSearchOperation</td>
<td>SearchOperation</td>
<td>The default search operation to use against this Property. Refer to the Javadoc for documentation of the SearchOperation enum constants.</td>
</tr>
<tr>
<td>length</td>
<td>int</td>
<td>Maximum length for string properties. Undefined for all other data types.</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Localized display string for the property name.</td>
</tr>
<tr>
<td>defaultValues</td>
<td>ArrayProperty</td>
<td>Default values for this property. (These are the actual, raw values.)</td>
</tr>
<tr>
<td>dependencies</td>
<td>List&lt;String&gt;</td>
<td>List of property names that this property depends on (in terms of their values), if isDynamic is true.</td>
</tr>
<tr>
<td>Field</td>
<td>Field type</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>valueAssist</td>
<td>ValueAssist</td>
<td>Provides default value assistance, that is, values to display to enable user to select a value for this property in a dialog box control. These values are provide for both static value assist (a fixed list of values), or dynamic value assist (values derived from the values of other properties). If isDynamic = true, then the value assist is dynamic, and getValueAssist provides default values. For information on getting dynamic values, see getDynamicAssistValues operation, page 118.</td>
</tr>
<tr>
<td>valueMap</td>
<td>List&lt;ValueInfo&gt;</td>
<td>A map of possible values for this property onto localizable display strings. This data can be cached and used to look up display strings for values obtained from the getDynamicAssistValues operation.</td>
</tr>
<tr>
<td>isArray</td>
<td>boolean</td>
<td>True if multiple values are allowed. (In repository terms this is a repeating attribute.)</td>
</tr>
<tr>
<td>isDynamic</td>
<td>boolean</td>
<td>If true, value assistance for this property is obtained dynamically based on a query or on the value of other attributes. For information on getting dynamic values, see getDynamicAssistValues operation, page 118.</td>
</tr>
<tr>
<td>isHidden</td>
<td>boolean</td>
<td>If true, property is to be hidden in the user interface.</td>
</tr>
<tr>
<td>isNotNull</td>
<td>boolean</td>
<td>If true, property cannot have null values.</td>
</tr>
<tr>
<td>isReadOnly</td>
<td>boolean</td>
<td>True if property is read-only.</td>
</tr>
<tr>
<td>isRequired</td>
<td>boolean</td>
<td>If true, user must provide this value for this property in the user interface (dialog box).</td>
</tr>
<tr>
<td>isSearchable</td>
<td>boolean</td>
<td>True if searches allowed on this property.</td>
</tr>
</tbody>
</table>

**ValueInfo**

A PropertyInfo instance stores a List<ValueInfo>. This List can be used to lookup the localizable display label representing the value if value assistance is available for the property.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>Property</td>
<td>A Property instance that stores the raw value that functions as the key in the value map.</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Localizable display label for the value.</td>
</tr>
</tbody>
</table>
RelationshipInfo

The RelationshipInfo is a descriptor that provides access to information about a Relationship defined by the underlying metadata of the schema. Relationship instances can be based on metadata stored using one of the following strategies:

- The implicit relationships folder and virtual document. These are hard-coded values passed as strings.
- Metadata stored in dm_relation_type.
- Metadata stored in dmc_relationship_def.

The following table shows RelationshipInfo fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the relationship.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>A description of the relationship.</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Localizable display string for relationship name.</td>
</tr>
<tr>
<td>currentType</td>
<td>String</td>
<td>The name of the type that the relationship is resolved against.</td>
</tr>
<tr>
<td>currentTypeRole</td>
<td>String</td>
<td>Role of the current type.</td>
</tr>
<tr>
<td>targetType</td>
<td>String</td>
<td>The repository object type of the source object in the relationship. Any object that participates in the relationship must be of this type or a subtype of this type.</td>
</tr>
<tr>
<td>targetTypeRole</td>
<td>String</td>
<td>Role that target type can play in this relationship.</td>
</tr>
<tr>
<td>degree</td>
<td>RelationshipDegree</td>
<td>An enum constant indicating the kind of mapping between the two terms of the relationship, with possible values of ONE_TO_ONE, ONE_TO_MANY, and MANY_TO_ONE. This data is available for relationships based on dmc_relationship_def only; otherwise it is null (in which case the Relationship is not completely defined).</td>
</tr>
<tr>
<td>propertyInfos</td>
<td>List&lt;PropertyInfo&gt;</td>
<td>Possible properties that can be set on the actual relationship object.</td>
</tr>
</tbody>
</table>

SchemaProfile

A SchemaProfile specifies categories of data returned by the Schema service. The following table describes the SchemaProfile fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isIncludeProperties</td>
<td>If true, return information regarding properties.</td>
</tr>
<tr>
<td>isIncludeValues</td>
<td>If true, return information regarding value assistance for properties.</td>
</tr>
</tbody>
</table>
## Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isIncludeRelationships</td>
<td>If true, return information regarding relationships for a specified type.</td>
</tr>
<tr>
<td>isIncludeTypes</td>
<td>If true, return information regarding repository object types.</td>
</tr>
<tr>
<td>scope</td>
<td>A String value that specifies a scope setting that confines attributes returned to a subset delimited to a specific scope. Typically scope is a value designating an application, such as webtop.</td>
</tr>
</tbody>
</table>

## getSchemaInfo operation

### Description

Retrieves schema information for the default schema of the specified repository. (Named schemas will be supported in a future release.)

### Java syntax

```java
SchemaInfo getSchemaInfo(String repositoryName,
                          String schemaName,
                          OperationOptions operationOptions)
  throws CoreServiceException
```

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>The name of the repository about which to obtain schema information.</td>
</tr>
<tr>
<td>schemaName</td>
<td>String</td>
<td>The name of the repository schema. If null or an empty string, examine the default repository schema.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of this operation, a SchemaProfile can be passed to control the information returned.</td>
</tr>
</tbody>
</table>

### Response

Returns a SchemaInfo instance containing the following information about a repository schema.
### Field Service

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the schema. Null if this is the default schema.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>The description of the schema. Null if this is the default schema.</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Default label for this schema. Null if this is the default schema.</td>
</tr>
<tr>
<td>typeInfos</td>
<td>List&lt;TypeInfo&gt;</td>
<td>A list of TypeInfo instances showing the types defined in the schema/repository.</td>
</tr>
</tbody>
</table>

### Example

**Example 6-1. Java: Getting schema info**

```java
public SchemaInfo getSchemaInfo() throws ServiceException {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    ISchemaService schemaSvc = serviceFactory.getRemoteService(ISchemaService.class, serviceContext);

    SchemaProfile schemaProfile = new SchemaProfile();
    schemaProfile.setIncludeTypes(true);
    serviceContext.setProfile(schemaProfile);

    SchemaInfo schemaInfo = schemaSvc.getSchemaInfo(defaultRepositoryName, "DEFAULT", null);
    System.out.println("Schema name is: " + schemaInfo.getName());
    System.out.println("Schema description is: " + schemaInfo.getDescription());
    System.out.println("Schema label is: " + schemaInfo.getLabel());
    List<TypeInfo> typeInfoList = schemaInfo.getTypeInfos();
    System.out.println("Printing schema type info:");
    for (TypeInfo typeInfo : typeInfoList) {
        System.out.println(typeInfo.getName());
    }
    return schemaInfo;
}
```

**Example 6-2. C#: Getting schema info**

```csharp
public void SchemaInfoDemo() {
    SchemaProfile schemaProfile = new SchemaProfile();
    schemaProfile.IncludeTypes = true;
    DemoServiceContext.SetProfile(schemaProfile);

    SchemaInfo schemaInfo = schemaService.GetSchemaInfo(DefaultRepository, "DEFAULT", null);
    Console.WriteLine("Schema name is: " + schemaInfo.Name);
    Console.WriteLine("Schema description is: " + schemaInfo.Description);
    Console.WriteLine("Schema label is: " + schemaInfo.Label);
    List<TypeInfo> typeInfoList = schemaInfo.TypeInfos;
    Console.WriteLine("Printing schema type info:");
    foreach (TypeInfo typeInfo in typeInfoList) {
        Console.WriteLine(typeInfo.Name);
    }
```
RepositoryInfo getRepositoryInfo(String repositoryName,
OperationOptions operationOptions)
throws CoreServiceException

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>Name of the repository to examine.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of this operation, a SchemaProfile can be passed to control the information returned.</td>
</tr>
</tbody>
</table>

**Response**

Returns a RepositoryInfo descriptor object containing the following data.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>The name of the repository.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>Description of the repository.</td>
</tr>
<tr>
<td>label</td>
<td>String</td>
<td>Localizable display string for the repository name.</td>
</tr>
<tr>
<td>schemaNameList</td>
<td>List&lt;String&gt;</td>
<td>A list of the repository schemas.</td>
</tr>
<tr>
<td>defaultSchemaName</td>
<td>List&lt;String&gt;</td>
<td>The name of the default schema. Typically the value is &quot;DEFAULT&quot;.</td>
</tr>
</tbody>
</table>
Example

Example 6.3. Java: Getting repository info

```java
public RepositoryInfo getRepositoryInfo() throws ServiceException {
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    ISchemaService schemaSvc = serviceFactory.getRemoteService(ISchemaService.class, serviceContext);

    SchemaProfile schemaProfile = new SchemaProfile();
    schemaProfile.setIncludeTypes(true);
    serviceContext.setProfile(schemaProfile);

    OperationOptions operationOptions = null;
    RepositoryInfo repositoryInfo = schemaSvc.getRepositoryInfo(defaultRepositoryName, operationOptions);

    System.out.println("Name: " + repositoryInfo.getName());
    System.out.println("Default schema name: " + repositoryInfo.getDefaultSchemaName());
    System.out.println("Label: " + repositoryInfo.getLabel());
    System.out.println("Description: " + repositoryInfo.getDescription());
    System.out.println("Schema names: ");
    List<String> schemaList = repositoryInfo.getSchemaNames();
    for (String schemaName : schemaList) {
        System.out.println(schemaName);
    }
    return repositoryInfo;
}
```

Example 6.4. C#: Getting repository info

```csharp
public RepositoryInfo RepositoryInfoDemo() {
    OperationOptions operationOptions = new OperationOptions();
    RepositoryInfo repositoryInfo = schemaService.GetRepositoryInfo(DefaultRepository, operationOptions);

    Console.WriteLine(repositoryInfo.Name);
    Console.WriteLine("Default schema name: " + repositoryInfo.DefaultSchemaName);
    Console.WriteLine("Label: " + repositoryInfo.Label);
    Console.WriteLine("Description: " + repositoryInfo.Description);
    Console.WriteLine("Schema names: ");
    List<String> schemaList = repositoryInfo.SchemaNames;
    foreach (String schemaName in schemaList) {
        Console.WriteLine(schemaName);
    }
    return repositoryInfo;
}
```
getTypeInfo operation

Description

The getTipoInfo operation returns information about a repository type specified by name.

Java syntax

```java
TypeInfo getTypeInfo(String repositoryName,
                     String schemaName,
                     String typeName,
                     OperationOptions operationOptions)
```

throws CoreServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>The name of the repository to examine.</td>
</tr>
<tr>
<td>schemaName</td>
<td>String</td>
<td>The name of the repository schema. For the current release set this value to &quot;DEFAULT&quot; or null.</td>
</tr>
<tr>
<td>typeName</td>
<td>String</td>
<td>The name of the type about which information is to be retrieved.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of this operation, a SchemaProfile can be passed to control the information returned.</td>
</tr>
</tbody>
</table>

Response

Returns a TypeInfo instance with populated with information about the specified type. For details, see TypeInfo, page 107. For information on the repository types, refer to the EMC Documentum Object Reference.

Example

Example 6-5. Java: Getting type info

```java
public TypeInfo getTypeInfo() throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    ISchemaService schemaSvc
```
Schema Service

= serviceFactory.getRemoteService(ISchemaService.class, serviceContext);

SchemaProfile schemaProfile = new SchemaProfile();
schemaProfile.setIncludeProperties(true);
schemaProfile.setIncludeValues(true);
serviceContext.setProfile(schemaProfile);

OperationOptions operationOptions = null;
TypeInfo typeInfo = schemaSvc.getTypeInfo(defaultRepositoryName, 
null, 
"dm_document", 
operationOptions);

System.out.println("Name: " + typeInfo.getName());
System.out.println("Label: " + typeInfo.getLabel());
System.out.println("Description: " + typeInfo.getDescription());
List<PropertyInfo> propertyInfoList;
propertyInfoList = typeInfo.getPropertyInfos();
System.out.println("Properties: ");
for (PropertyInfo propertyInfo in propertyInfoList)
{
    System.out.print(" " + propertyInfo.getName());
    System.out.println(" " + propertyInfo.getDataType().toString());
}
return typeInfo;

Example 6-6. C#: Getting type info

public TypeInfo TypeInfoDemo()
{
    SchemaProfile schemaProfile = new SchemaProfile();
schemaProfile.IncludeProperties = true;
schemaProfile.IncludeValues = true;
DemoServiceContext.SetProfile(schemaProfile);

    OperationOptions operationOptions = null;
    TypeInfo typeInfo = schemaService.GetTypeInfo(DefaultRepository, 
null, 
"dm_document", 
operationOptions);

    Console.WriteLine("Name: " + typeInfo.Name);
    Console.WriteLine("Label: " + typeInfo.Label);
    Console.WriteLine("Description: " + typeInfo.Description);
    List<PropertyInfo> propertyInfoList;
    propertyInfoList = typeInfo.PropertyInfos;
    Console.WriteLine("Properties: ");
    foreach (PropertyInfo propertyInfo in propertyInfoList)
    {
        Console.WriteLine(" " + propertyInfo.Name);
        Console.WriteLine(" " + propertyInfo.DataType.ToString());
    }
    return typeInfo;
}
getPropertyInfo operation

Description

The getPropertyInfo operation returns data about a repository property specified by repository, schema, type, and name.

Java syntax

```java
PropertyInfo getPropertyInfo(String repositoryName,
                                 String schemaName,
                                 String typeName,
                                 String propertyName
                                 OperationOptions operationOptions)
```

throws CoreServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>The name of the repository to examine.</td>
</tr>
<tr>
<td>schemaName</td>
<td>String</td>
<td>The name of the repository schema.</td>
</tr>
<tr>
<td>typeName</td>
<td>String</td>
<td>The name of the repository type in which information about this property is to be retrieved.</td>
</tr>
<tr>
<td>propertyName</td>
<td>String</td>
<td>The name of the repository property about which to retrieve information.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of this operation, a SchemaProfile can be passed to control the information returned.</td>
</tr>
</tbody>
</table>

Response

Returns a PropertyInfo instance with populated with information about the specified property. The following table describes the fields of the PropertyInfo class. For details, see PropertyInfo, page 108.
getDynamicAssistValues operation

Description

The getDynamicAssistValues operation retrieves information about dynamic value assistance for a specified repository property. Value assistance provides a list of valid values for a property, which are used to populate a pick list associated with a field on a dialog box. Dynamic value assistance uses a query or a routine to list possible values for an attribute, generally based on the values of other attributes, rather than a literal list. A value assist list (whether literal or dynamic) can be complete—meaning that no values for the property are valid other than those in the list, or incomplete—meaning that the user is allowed to provide values in addition to those in the list.
Java syntax

```java
ValueAssist getDynamicAssistValues(String repositoryName,
                                          String schemaName,
                                          String typeName,
                                          String propertyName,
                                          PropertySet propertySet,
                                          OperationOptions operationOptions)
     throws CoreServiceException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>The name of the repository to examine.</td>
</tr>
<tr>
<td>schemaName</td>
<td>String</td>
<td>The name of the repository schema. For the current release set this value to &quot;DEFAULT&quot; or null.</td>
</tr>
<tr>
<td>typeName</td>
<td>String</td>
<td>The name of the repository type in which information about the property is to be retrieved.</td>
</tr>
<tr>
<td>propertyName</td>
<td>String</td>
<td>The name of the repository property about which to retrieve information.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of this operation, a SchemaProfile can be passed to control the information returned.</td>
</tr>
</tbody>
</table>

Response

Returns a ValueAssist object containing data about any value assistance configured in the repository for the property in question.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>values</td>
<td>List&lt;String&gt;</td>
<td>A List of the raw values to be used as value assistance.</td>
</tr>
<tr>
<td>isAllowUserValues</td>
<td>boolean</td>
<td>If true, this property allows users to add their own values, in addition to those provided by value assistance. If false, the user can choose only values that are provided by value assistance.</td>
</tr>
</tbody>
</table>

Notice that only the raw values for value assistance are returned. This is an optimization to minimize payload size for this operation. To look up the labels for the values, you can use the getPropertyValueInfo operation to retrieve and cache values locally for properties, then use the getValueMap method of the PropertyInfo object to look up the label on the relevant property, using the raw value returned in ValueAssist as a key.
Example

The following example shows basic usage of the getDynamicAssistValues operation.

Example 6-9. Java: Getting dynamic assist values

public void demoGetValueInfo() throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    ISchemaService schemaSvc = serviceFactory.getRemoteService(ISchemaService.class, serviceContext);

    SchemaProfile schemaProfile = new SchemaProfile();
    schemaProfile.setIncludeValues(true);
    OperationOptions operationOptions = new OperationOptions();
    operationOptions.setSchemaProfile(schemaProfile);

    System.out.println("Printing value info:");
    ValueAssist valueAssist = schemaSvc.getDynamicAssistValues(defaultRepositoryName, null, "dm_document", "subject", null, operationOptions);

    if (valueAssist == null)
    {
        System.out.println("valueAssist is null.");
        return;
    }
    for (String value : valueAssist.getValues())
    {
        System.out.println("  " + value);
    }
}

Example 6-10. C#: Getting dynamic assist values

public void DemoGetValueInfo()
{
    SchemaProfile schemaProfile = new SchemaProfile();
    schemaProfile.IncludeValues = true;
    OperationOptions operationOptions = new OperationOptions();
    operationOptions.SchemaProfile = schemaProfile;

    Console.WriteLine("Printing value info:");
    ValueAssist valueAssist = schemaService.GetDynamicAssistValues(DefaultRepository, null, "dm_document", "subject", null, operationOptions);

    if (valueAssist == null)
    {
        Console.WriteLine("valueAssist is null.");
        return;
    }
    foreach (String value in valueAssist.Values)
    {
        Console.WriteLine("  " + value);
    }
}
)})
The Query service is a primary mechanism for retrieving information from a repository. The Query service is general purpose and uses execution semantics similar to the use of queries in an RDBMS. The service returns a data set resulting from the query to the user either directly or through asynchronous caching.

This chapter covers the following topics:

- Query model, page 123
- QueryExecution, page 123
- PassthroughQuery, page 124
- execute operation, page 125

### Query model

The Query class has two subclasses: StructuredQuery and PassthroughQuery. For Version 6, the Query service only accepts objects of class PassthroughQuery. Execution of a StructuredQuery is not supported.

### QueryExecution

The QueryExecution class defines an object that is passed as an argument to the Query service, and which encapsulates settings that specify Query service behaviors. The following table summarizes the QueryExecution fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryId</td>
<td>String</td>
<td>Id of the query. This should be set to null for a new query or should be set to the queryId returned by the operation in the QueryResult for sequential processing of cached query results.</td>
</tr>
<tr>
<td>Field</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>startingIndex</td>
<td>long</td>
<td>Specifies the position in the query results beginning at which to return data to the service client. Default value is 0. Normally used only in sequential processing of cached query results.</td>
</tr>
<tr>
<td>maxResultCount</td>
<td>int</td>
<td>Specifies the maximum number of DataObject instances returned in the QueryResult. If set to the default (-1) there is no defined limit.</td>
</tr>
<tr>
<td>maxResultPerSource</td>
<td>int</td>
<td>For Search service: the number of maximum number of results that can be returned to the client by any one of the managed or external repositories that are in the search scope. Not used by the Query service: should be set to -1. If set to the default (-1) there is no defined limit.</td>
</tr>
<tr>
<td>cacheStrategyType</td>
<td>CacheStrategyType</td>
<td>Specifies a service behavior for caching query results that can be sequentially processed in multiple service interactions. See CacheStrategyType values, page 124. Supported by Query service. Not supported by Search service in DFS version 6.</td>
</tr>
</tbody>
</table>

### CacheStrategyType values

The following table describes the CacheStrategyType values.

<table>
<thead>
<tr>
<th>CacheStrategyType value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFAULT_CACHE_STRATEGY</td>
<td>The system default for caching query results, which is equal to NO_CACHE_STRATEGY.</td>
</tr>
<tr>
<td>BASIC_FILE_CACHE_STRATEGY</td>
<td>Cache query results on the remote file system. If the cached result does not exist the query is re-run.</td>
</tr>
<tr>
<td>BASIC_MEMORY_CACHE_STRATEGY</td>
<td>Cache query results in memory on the remote system. If the cached result does not exist the query is re-run.</td>
</tr>
<tr>
<td>NO_CACHE_STRATEGY</td>
<td>Do not cache query results, and void any previous query cache stored for this user and query.</td>
</tr>
</tbody>
</table>

### PassthroughQuery

The PassthroughQuery type extends Query, and contains a queryString field that holds a DQL statement.
Example

Example 7-1. Java: PassthroughQuery

```java
PassthroughQuery query = new PassthroughQuery();
query.setQueryString("select r_object_id, "
+ "object_name from dm_cabinet";
query.setRepository(defaultRepositoryName);
```

Example 7-2. C#: PassthroughQuery

```csharp
PassthroughQuery query = new PassthroughQuery();
query.QueryString = "select r_object_id, "
+ "object_name from dm_cabinet";
query.AddRepository(DefaultRepository);
```

execute operation

Description

The execute operation runs a query against data in a repository and returns the results to the client as a QueryResult containing a DataPackage.

Java syntax

```java
QueryResult execute(PassthroughQuery query, 
queryExecution QueryExecution 
OperationOptions operationOptions) 
throws CoreServiceException, 
QueryValidationException, 
CacheException
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>PassthroughQuery</td>
<td>Contains a DQL statement that expresses the query.</td>
</tr>
<tr>
<td>queryExecution</td>
<td>QueryExecution</td>
<td>Object describing execution parameters.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of the execute operation, the profiles primarily provide filters that modify the contents of DataPackage returned in the QueryResult.</td>
</tr>
</tbody>
</table>
Response

The execute operation returns a QueryResult, which contains:

• A queryId string matching the id in the query passed to the service. This aids the client in matching the query result to the query in batch operations.

• A DataPackage containing a DataObject for each repository object selected by the query. By default, each DataObject is contains a PropertySet and ObjectIdentity populated with the query results. This result can be modified by filter settings in profiles passed in OperationOptions.

The QueryResult object contains substantial additional information within a QueryStatus object, some of which is more relevant to the use of QueryResult in the Search service. For more information see QueryResult, page 137.

Examples

The following examples demonstrate:

• Basic PassthroughQuery, page 126
• Cached query processing, page 127

Basic PassthroughQuery

The following examples shows basic use of a PassthroughQuery. In this example the query result is not cached, and the entire result is returned to the client.

Example 7-3. Java: Executing a PassthroughQuery

```java
public void basicPassthroughQuery()
    throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IQueryService querySvc = serviceFactory.getRemoteService(IQueryService.class,
            serviceContext);
    PassthroughQuery query = new PassthroughQuery();
    query.setQueryString("select r_object_id, "
            + "object_name from dm_cabinet");
    query.addRepository(defaultRepositoryName);
    QueryExecution queryEx = new QueryExecution();
    queryEx.setCacheStrategyType(CacheStrategyType.DEFAULT_CACHE_STRATEGY);
    OperationOptions operationOptions = null;
    QueryResult queryResult = querySvc.execute(query, queryEx, operationOptions);
    System.out.println("QueryId = " + query.getQueryString());
    System.out.println("CacheStrategyType == " + queryEx.getCacheStrategyType());
    DataPackage resultDp = queryResult.getDataPackage();
    List<DataObject> dataObjects = resultDp.getDataObjects();
    int numberOfObjects = dataObjects.size();
    System.out.println("Total objects returned is: " + numberOfObjects);
    for (DataObject dObj : dataObjects)
    {
        PropertySet docProperties = dObj.getProperties();
        String objectId = dObj.getIdentity().getValueAsString();
```
Example 7-4. C#: Executing a PassthroughQuery

```csharp
public void BasicPassthroughQuery()
{
    PassthroughQuery query = new PassthroughQuery();
    query.QueryString = "select r_object_id, " + "object_name from dm_cabinet";
    query.AddRepository(DefaultRepository);
    QueryExecution queryEx = new QueryExecution();
    queryEx.CacheStrategyType = CacheStrategyType.DEFAULT_CACHE_STRATEGY;
    OperationOptions operationOptions = null;
    QueryResult queryResult = queryService.Execute(query, queryEx, operationOptions);
    Console.WriteLine("QueryId = = = " + query.QueryString);
    Console.WriteLine("CacheStrategyType = = = " + queryEx.CacheStrategyType);
    DataPackage resultDp = queryResult.DataPackage;
    List<DataObject> dataObjects = resultDp.DataObjects;
    int numberOfObjects = dataObjects.Count;
    foreach (DataObject dObj in dataObjects)
    {
        PropertySet docProperties = dObj.Properties;
        String objectId = dObj.Identity.GetValueAsString();
        String docName = docProperties.Get("object_name").GetValueAsString();
        Console.WriteLine("Document " + objectId + " name is " + docName);
    }
}
```

Cached query processing

To process large result sets, the client can specify that they be cached on the remote system and process the query result sequentially in a loop. Each pass can examine a range of the query results determined by startingIndex position and maxQueryResultCount. When the startingIndex position is out of range, the execute operation will return a QueryResult containing zero DataObject instances.

Example 7-5. Java: Cached query

```java
public void cachedPassthroughQuery() throws ServiceException
{
    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IQueryService querySvc = serviceFactory.getRemoteService(IQueryService.class,
                                                            serviceContext);
    PassthroughQuery query = new PassthroughQuery();
    query.setQueryString("select r_object_id, " + "object_name from dm_cabinet");
    query.addRepository(defaultRepositoryName);
    QueryExecution queryEx = new QueryExecution();
    OperationOptions operationOptions = null;
    queryEx.setCacheStrategyType(CacheStrategyType.BASIC_FILE_CACHE_STRATEGY);
    queryEx.setMaxResultCount(10);
    while (true)
    {
        QueryResult queryResult = querySvc.execute(query,
                                                    queryEx,
                                                    operationOptions,
                                                    startingIndex,
                                                    maxQueryResultCount);
        List<DataObject> dataObjects = queryResult.DataPackage.DataObjects;
        int numberOfObjects = dataObjects.Count;
        if (numberOfObjects > 0)
        {
            List<DataObject> dataObjects = queryResult.DataPackage.DataObjects;
            int numberOfObjects = dataObjects.Count;
            if (numberOfObjects == 0)
            {
                break;
            }
        }
    }
}
```
operationOptions);
DataPackage resultDp = queryResult.getDataPackage();
List<DataObject> dataObjects = resultDp.getDataObjects();
int numberOfObjects = dataObjects.size();
if (numberOfObjects == 0)
    { break; }
System.out.println("Total objects returned is: "+ numberOfObjects);
for (DataObject dObj : dataObjects)
    {
    PropertySet docProperties = dObj.getProperties();
    String objectId = dObj.getIdentity().getValueAsString();
    String cabinetName = docProperties.get("object_name").getValueAsString();
    System.out.println("Cabinet " + objectId + " name is " + cabinetName);
    }
queryEx.setStartingIndex(queryEx.getStartingIndex() + 10);
}

Example 7-6. C#: Cached query

public void CachedPassthroughQuery()
{
    PassthroughQuery query = new PassthroughQuery();
    query.QueryString = "select r_object_id, "
        + "object_name from dm_cabinet";
    query.AddRepository(DefaultRepository);
    QueryExecution queryEx = new QueryExecution();
    OperationOptions operationOptions = null;
    queryEx.CacheStrategy = CacheStrategyType.BASIC_FILE_CACHE_STRATEGY;
    queryEx.MaxResultCount = 10;
    while (true)
    {
    QueryResult queryResult = queryService.Execute(query,
        queryEx, operationOptions);
    DataPackage resultDp = queryResult.DataPackage;
    List<DataObject> dataObjects = resultDp.DataObjects;
    int numberOfObjects = dataObjects.Count;
    if (numberOfObjects == 0)
        { break; }
    Console.WriteLine("Total objects returned is: "+ numberOfObjects);
    foreach (DataObject dObj in dataObjects)
        {
    PropertySet docProperties = dObj.Properties;
    String objectId = dObj.Identity.GetValueAsString();
    String cabinetName = docProperties.Get("object_name").GetValueAsString();
    Console.WriteLine("Cabinet " + objectId + " name is "
        + cabinetName);
    }
    queryEx.StartingIndex += 10;
}
})
Search Service

The Search service provides full-text and structured search capabilities against multiple EMC Documentum repositories (termed managed repositories in DFS), as well as against external sources (termed external repositories).

Successful use of the Search service is dependent on deployment and configuration of full-text indexing on Documentum repositories, and installation of ECI adapters on external repositories (registered with an ECIS server). For information on these topics, refer to the following documents:

- EMC Documentum Content Server Full-Text Indexing System Installation and Administration Guide
- EMC Enterprise Content Integration Services Adapter Development Guide

To use the Search service it is also helpful to understand FTDQL queries, dfc.properties settings, and DQL hint file settings. For information on these topics, refer to the EMC Documentum Search Development Guide. For full information on FTDQL syntax, refer to the Content Server DQL Reference Manual.

Note: The Object service get operation can return contents from both managed and external repositories based on the search results. For more information see Getting content from external sources, page 72.

This chapter covers the following topics:

- Full-text and database searches, page 132
- PassthroughQuery, page 132
- StructuredQuery, page 132
- ExpressionSet, page 134
- RepositoryScope, page 134
- Expression model, page 134
- QueryResult, page 137
- getRepositoryList operation, page 139
- execute operation, page 141
Full-text and database searches

Search service queries can be run as either full-text queries against a full-text index, or as database queries against object attributes on a managed or external repository.

Whether the search query is run as a full-text or database search depends on a number of different factors.

- The availability to the service of full-text indexed repositories.
- Settings in the DQL hints file, if present.
- The presence or absence of full-text expressions (a SEARCH DOCUMENT CONTAINS clause) in a DQL query.
- Explicit setting of setDatabaseSearch in a StructuredQuery.

Searches against a full-text index are case-insensitive. Database searches are by default case-sensitive.

If a query includes a full-text expression, either as a SEARCH DOCUMENT CONTAINS clause in PassthroughQuery, or as a FullTextExpression object in a StructuredQuery (see FullTextExpression, page 135), but is executed as a database query, the full-text expression is evaluated against the title, subject, and object_name properties of repository objects of type dm_sysobject.

PassthroughQuery

The PassthroughQuery object is a container for a DQL or FTDQL query string. It can be executed as either a full-text or database query, depending on factors specified in Full-text and database searches, page 132.

A PassthroughQuery will search multiple managed repositories, but does not run against external repositories. To search an external repository a client must use a StructuredQuery.

StructuredQuery

A structured query defines a query using an object-oriented model. The query is constrained by a set of criteria contained in an ExpressionSet object, and the scope of the query or search (the sources against which it is run), is defined by an ordered list of RepositoryScope objects. The following table describes the StructuredQuery fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dateFormat</td>
<td>String</td>
<td>String representing the date format. For details, see Date and time, page 133.</td>
</tr>
<tr>
<td>objectType</td>
<td>String</td>
<td>String representing the repository object type to query on. If not specified, the default value is dm_sysobject. Searches are always executed against a single object type.</td>
</tr>
</tbody>
</table>
### Field | Data Type | Description
--- | --- | ---
rootExpressionSet | ExpressionSet | The expression set defining the search criteria. Because ExpressionSet instances can be nested, the top of the expression hierarchy is called the root expression set. For information on expression sets see ExpressionSet, page 134.
scopes | List<RepositoryScope> | A collection of RepositoryScope objects that determines the total scope of the search. Each RepositoryScope object represents a search source. For more information on RepositoryScope, see RepositoryScope, page 134.
isDatabaseSearch | boolean | If true, force the search to run against the database. If false, the search will run in full-text mode unless some other factor prevents executing the search against a full-text index.
isIncludeAllVersions | boolean | If true, the search includes all versions of objects, if versioning is supported by the search source. If false, only the CURRENT version of objects are included in the search.
isIncludeHidden | boolean | If true, hidden objects are included. Whether an object is hidden is determined by the a_is_hidden property of the dm_sysobject.

### Date and time

The getDataFormat and setDateFormat methods use a date format String to determine the format date of date values included in the query’s expression set. The format strings understood by DFS are the same as those specified in the IDFTime DFC interface. Refer to the DFC Javadocs for a complete list of these datetime format strings. For each IDFTime format string, the corresponding java.text.SimpleDateFormat string is also supported; for example "MM/dd/yyyy hh:mm:ss" is equivalent to "mm/dd/yyyy hh:mm:ss".  

The time zone of the datetime value used in an expression should match the time zone where the DFS service (and therefore the DFC client) is located. When the query is sent to a Content Server, Content Server translates the date into a universal time zone (GMT+00) based on the assumption that the date provided is expressed in the time zone of the DFC client. This can lead to unexpected results if the date is not provided at an appropriate level of precision. As a matter of best practice, do not use a date format that is precise only to the day (for example "mm/dd/yyyy"). A date provided at this level of precision can in some cases cause Content Server to calculate GMT 1 day before or after the date that the query expects. For example if the date passed to Content Server is using the
DFC time zone GMT+1 (for example 01/26/1973), Content Server will transform it as '01/26/1973 00:00:00' minus 1 hour (with a result of 01/25/1973).

**ExpressionSet**

An ExpressionSet is collection of Expression objects, each of which defines either a full-text expression, or a search constraint on a single property. The Expression instances comprising the ExpressionSet are related to one another by a single logical operator (either AND or OR). The ExpressionSet as a whole defines the complete set of search criteria that will be applied during a search.

An ExpressionSet contains Expression instances, and it also extends the Expression class. This enables an ExpressionSet to nest ExpressionSet instances, permitting construction of arbitrarily complex expression trees. The top-level Expression passed contained in a StructuredQuery is referred to as the root expression of the expression tree.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>expressions</td>
<td>List&lt;Expression&gt;</td>
<td>A list of Expression instances, which can be of concrete type FullTextExpression or PropertyExpression, or a nested ExpressionSet.</td>
</tr>
<tr>
<td>operator</td>
<td>ExpressionSetOperator</td>
<td>Logical operator connecting members of the ExpressionSet. Possible values are AND and OR.</td>
</tr>
</tbody>
</table>

**RepositoryScope**

RepositoryScope enables a search to be constrained to a specific folder of a repository.

<table>
<thead>
<tr>
<th>Field</th>
<th>Field type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>String</td>
<td>The name of the managed or external repository.</td>
</tr>
<tr>
<td>locationPath</td>
<td>String</td>
<td>The path to the folder to which the search is to be constrained. This value is only valid for managed repositories. For example '/mycabinet/myfolder'.</td>
</tr>
<tr>
<td>isDescend</td>
<td>boolean</td>
<td>Whether the descendants of the search folder are to be included in the search. Valid only for managed repositories.</td>
</tr>
</tbody>
</table>

**Expression model**

The Expression class is extended by three concrete classes: FullTextExpression, PropertyExpression, and ExpressionSet.

Because ExpressionSet extends Expression and contains a set of Expression instances, an ExpressionSet can nest ExpressionSet instances. This allows construction of arbitrarily complex
expression trees. The top-level Expression passed contained in a StructuredQuery is referred to as the *root expression* of the expression tree.

**FullTextExpression**

FullTextExpression encapsulates a search string accessed using the getValue and setValue methods. This string supports use of "AND" and "OR", as well as parentheses. The following are examples of full-text expressions:

"foo bar"
foo bar
foo AND bar
foo OR bar
foo AND bar OR cake
foo AND (bar OR cake)

The Search service interprets the string using the following rules:

- A quoted string is searched for as a complete phrase.
- Words separated by space without an operator use an implicit ACCRUE operator (essentially an OR operator with a result ranking that gives higher scores to results that contain more of the words) for full-text queries. For database queries the operator is a simple OR.
- AND has precedence over OR.
- Search is case-insensitive by default for full-text queries, and case-sensitive by default for database queries.

**PropertyExpression**

PropertyExpression provides a search constraint based on a single property.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property</td>
<td>String</td>
<td>String representing the repository property name.</td>
</tr>
<tr>
<td>value</td>
<td>ExpressionValue</td>
<td>The property value on which to constrain the search. For more information see ExpressionValue, page 136.</td>
</tr>
<tr>
<td>condition</td>
<td>Condition</td>
<td>Determines the logical comparison to use when evaluating repository values against search criteria. For more information see Condition, page 136.</td>
</tr>
<tr>
<td>isCaseSensitive</td>
<td>boolean</td>
<td>Forces case-sensitivity of the search. Optional. Default value is true for full-text, false for database search.</td>
</tr>
<tr>
<td>Field</td>
<td>Data Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>dataType</td>
<td>DataType</td>
<td>Optional value to indicate repository data type of the property. If not provided, the data type will be looked up in the data dictionary. Possible values are BOOLEAN, CUSTOM, DATE, DOUBLE, INTEGER, LONG, SHORT, OBJECT_ID, STRING.</td>
</tr>
<tr>
<td>repeated</td>
<td>String</td>
<td>Optional value to indicate whether this is a repeated property. If not provided, this will be looked up in the data dictionary. Possible values are &quot;true&quot;, &quot;false&quot;, and null.</td>
</tr>
</tbody>
</table>

### ExpressionValue

Table 8, page 136 describes the concrete subtypes of the ExpressionValue class.

**Table 8. ExpressionValue subtypes**

<table>
<thead>
<tr>
<th>Subtype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SimpleValue</td>
<td>Contains a single String value.</td>
</tr>
<tr>
<td>RangeValue</td>
<td>Contains two String values representing the start and end of a range. The values can represent dates (using the DateFormat specified in the StructuredQuery) or integers.</td>
</tr>
<tr>
<td>ValueList</td>
<td>Contains an ordered List of String values.</td>
</tr>
<tr>
<td>RelativeDateValue</td>
<td>Contains a TimeUnit setting and an integer value representing the number of time units. TimeUnit values are MILLISECOND, SECOND, MINUTE, HOUR, DAY, ERA, WEEK, MONTH, YEAR. The integer value can be negative or positive to represent a past or future time.</td>
</tr>
</tbody>
</table>

### Condition

Condition is an enumerated type that expresses the logical condition to use when comparing a repository value to a value in an Expression. A specific Condition is included in a PropertyExpression to determine precisely how to constrain the search on the property value.

The following values are largely self-explanatory. Note, however, that the BETWEEN Condition is only valid when the ExpressionValue instance is of subtype RangeValue, and that the test is inclusive. The BETWEEN condition is only valid for database searches.

- EQUAL
- NOT_EQUAL
- GREATER_THAN
- LESS_THAN
- GREATER_EQUAL
LESS_EQUAL
BEGINsWith
CONTAINS
DOES_NOT_CONTAIN
ENDsWith
IN
NOT_IN
BETWEEN
IS_NULL
IS_NOT_NULL

**QueryResult**

The QueryResult class is used by both the Search and Query services as a container for the set of results returned by the execute operation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>queryId</td>
<td>String</td>
<td>A unique identifier for the query, which can be passed back to retrieve cached query results. Query caching is support by the Query service, but not supported for DFS version 6 by the Search service.</td>
</tr>
<tr>
<td>dataPackage</td>
<td>DataPackage</td>
<td>Collection of object representations returned by the query. The content and metadata returned in these objects can be specified using Profile instances passed in OperationOptions.</td>
</tr>
<tr>
<td>dataObjects</td>
<td>List&lt;DataObject&gt;</td>
<td>List of object representations returned by the query. The objects are the same as those returned in the DataPackage. This is a convenience method available only with the Java client library.</td>
</tr>
<tr>
<td>queryStatus</td>
<td>QueryStatus</td>
<td>Object containing data about the status of the query that the service tried to execute. For more information, see QueryStatus, page 137.</td>
</tr>
</tbody>
</table>

**QueryStatus**

QueryStatus contains status information returned by a search operation. The status information can be examined for each search source repository.
<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryStatusInfos</td>
<td>List&lt;RepositoryStatusInfo&gt;</td>
<td>An ordered list of RepositoryStatusInfo instances, each of which provides information about status of this query result for a specific repository that was searched or queried.</td>
</tr>
<tr>
<td>isHasMoreResults</td>
<td>boolean</td>
<td>True if the returned results include the entire query results. If false, there were results returned that were above the range specified by the maxResult specified in the QueryExecution object passed to the search operation. Note that the values below startingIndex are not counted. For more information see QueryExecution, page 123.</td>
</tr>
</tbody>
</table>

**RepositoryStatusInfo**

RepositoryStatusInfo contains data related to a query or search result regarding the status of the search in a specific repository. RepositoryStatusInfo instances are returned in a List<RepositoryStatusInfo> within a QueryResult, which is returned by a search or query operation.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of this managed or external repository.</td>
</tr>
<tr>
<td>hitCount</td>
<td>int</td>
<td>Number of hits generated by this search or query on this repository. If it is not supported result will be -1.</td>
</tr>
<tr>
<td>resultCount</td>
<td>int</td>
<td>Number of results returned by this search or query on this repository.</td>
</tr>
<tr>
<td>isHasMoreResults</td>
<td>boolean</td>
<td>True if the query produced more results than were returned to the client.</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>General status of the search or query. Possible values are FAILURE or SUCCESS.</td>
</tr>
<tr>
<td>errorMessage</td>
<td>String</td>
<td>Error message returned by the query for this repository.</td>
</tr>
<tr>
<td>errorTrace</td>
<td>String</td>
<td>Stack trace for the error returned by the query for this repository.</td>
</tr>
<tr>
<td>detailedStatus</td>
<td>RepositoryStatus</td>
<td>An enumerated type indicating the search status on a specific managed or external repository.</td>
</tr>
</tbody>
</table>

**RepositoryStatus**

RepositoryStatus generally provides detail information about the status of a query that has executed, as pertains to a specific repository.
## getRepositoryList operation

### Description

The getRepositoryList operation provides list of managed and external repositories that are available to the service for searching.

<table>
<thead>
<tr>
<th>RepositoryStatusDetail value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COULD_NOT_FETCH</td>
<td>The query could not be sent to the repository because of an I/O error.</td>
</tr>
<tr>
<td>EXPIRED</td>
<td>The search execution has taken too much time and has been stopped.</td>
</tr>
<tr>
<td>EXT_SOURCE_UNAVAILABLE</td>
<td>The connection with the external repository was broken, so the query could not complete.</td>
</tr>
<tr>
<td>INTERNAL_FAILURE</td>
<td>An internal error occurred, possibly due to an uncaught exception or an unavailable adapter.</td>
</tr>
<tr>
<td>LOGIN FAILED</td>
<td>The query could not be processed on the repository because of an authentication failure.</td>
</tr>
<tr>
<td>NOT_CONSTRAINED</td>
<td>The query was not processed by the adapter because there are not valid constraints defined in the query, or the query is not supported by the repository.</td>
</tr>
<tr>
<td>OVERLOADED</td>
<td>The remote source is not available now. Try later.</td>
</tr>
<tr>
<td>QUERY_SCOPE_UNREACHABLE</td>
<td>The repository is unreachable (either not found or no access permission), and the query returned 0 results.</td>
</tr>
<tr>
<td>TOO_MANY_RESULTS</td>
<td>The query would return too many results. Try to refine the query.</td>
</tr>
<tr>
<td>TRUNCATED</td>
<td>The results have been truncated. This value can be true when the query status is SUCCESS.</td>
</tr>
<tr>
<td>TYPE_NOT_SUPPORTED</td>
<td>The repository does not support the object type defined in the query.</td>
</tr>
<tr>
<td>UNAVAILABLE</td>
<td>The managed repository is not available.</td>
</tr>
<tr>
<td>UNREACHABLE</td>
<td>The query could not be processed because it failed too many times to connect to the source (network problem).</td>
</tr>
<tr>
<td>WAS_STOPPED</td>
<td>The query was stopped by the user and terminated before completion.</td>
</tr>
</tbody>
</table>
Java syntax

List<Repository> getRepositoryList(OperationOptions options)

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. This parameter is not used by the operation in DFS version 6.</td>
</tr>
</tbody>
</table>

Returns

Returns a List of Repository instances.

Repository

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>Name of this repository.</td>
</tr>
<tr>
<td>type</td>
<td>RepositoryType</td>
<td>The general type of this repository. Possible values are MANAGED and EXTERNAL. A MANAGED repository is an EMC Documentum repository, which means that the results can be referenced using their ObjectIds. If the repository is EXTERNAL, the results metadata will be mapped to dm_document attributes. EXTERNAL results do not have an ObjectId.</td>
</tr>
<tr>
<td>properties</td>
<td>RepositoryProperty</td>
<td>Object listing properties (often representing capabilities) of this repository. For more information on RepositoryProperty, refer to the Javadoc.</td>
</tr>
</tbody>
</table>

Example

The following example demonstrates the getRepositoryList operation.

Example 8-1. Java: Getting a repository list

```java
public List<Repository> repositoryList()
{
    try
    {
        ServiceFactory serviceFactory = ServiceFactory.getInstance();
        ISearchService searchService
```
= serviceFactory.getService(ISearchService.class, serviceContext);
List<Repository> repositoryList = searchService.getRepositoryList(null);
for (Repository r : repositoryList)
{
    System.out.println(r.getName());
    return repositoryList;
} catch (Exception e)
{
    e.printStackTrace();
    throw new RuntimeException(e);
}

Example 8-2. C#: Getting a repository list

public List<Repository> RepositoryList()
{
    try
    {
        List<Repository> repositoryList = searchService.GetRepositoryList(null);
        foreach (Repository r in repositoryList)
        {
            Console.WriteLine(r.Name);
        } return repositoryList;
    } catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
        throw new Exception(e.Message);
    }

execute operation

Description

The execute operation searches a repository or set of repositories and returns search results.

Java syntax

QueryResult execute(Query query,  
QueryExecution execution,  
OperationOptions options)  
throws CoreServiceException
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>query</td>
<td>Query</td>
<td>Either a PassthroughQuery (see PassthroughQuery, page 132) or a StructuredQuery (see StructuredQuery, page 132).</td>
</tr>
<tr>
<td>queryExecution</td>
<td>QueryExecution</td>
<td>Object describing execution parameters. For information see QueryExecution, page 123.</td>
</tr>
<tr>
<td>operationOptions</td>
<td>OperationOptions</td>
<td>Contains profiles and properties that specify operation behaviors. In the case of the execute operation, the profiles primarily provide filters that modify the contents of the DataPackage returned in QueryResult. Note that in a PropertyProfile only SPECIFIED_BY_INCLUDE is supported in DFS version 6. SPECIFIED_BY_EXCLUDE is not supported. For more information see PropertyProfile, page 39.</td>
</tr>
</tbody>
</table>

Returns

Returns a QueryResult instance. For information on QueryResult see QueryResult, page 137.

Examples

The following examples demonstrate the following use cases:

- Simple passthrough query, page 142
- Structured query, page 144

Simple passthrough query

Example 8-3. Java: Simple PassthroughQuery

```java
public QueryResult simplePassthroughQuery()
{
    QueryResult queryResult;
    try
    {
        ServiceFactory serviceFactory = ServiceFactory.getInstance();
        ISearchService searchService = serviceFactory.getService(ISearchService.class, serviceContext);

        String queryString = "select distinct r_object_id from dm_document order by r_object_id ";
        int startingIndex = 0;
        int maxResults = 60;
        int maxResultsPerSource = 20;
```
PassthroughQuery q = new PassthroughQuery();
q.setQueryString(queryString);
q.addRepository(defaultRepositoryName);

QueryExecution queryExec = new QueryExecution(startingIndex,
    maxResults,
    maxResultsPerSource);

queryExec.setCacheStrategyType(CacheStrategyType.NO_CACHE_STRATEGY);

queryResult = searchService.execute(q, queryExec, null);

QueryStatus queryStatus = queryResult.getQueryStatus();
RepositoryStatusInfo repStatusInfo = queryStatus.getRepositoryStatusInfos().get(0);
if (repStatusInfo.getStatus() == Status.FAILURE)
{
    System.out.println(repStatusInfo.getErrorTrace());
    throw new RuntimeException("Query failed to return result.");
}
System.out.println("Query returned result successfully.");
DataPackage dp = queryResult.getDataPackage();
System.out.println("DataPackage contains " + dp.getDataObjects().size() + " objects.");
for (DataObject dataObject : dp.getDataObjects())
{
    System.out.println(dataObject.getIdentity());
}
}
catch (Exception e)
{
    e.printStackTrace();
    throw new RuntimeException(e);
}
return queryResult;


Example 8-4. C#: SimplePassthroughQuery

public QueryResult SimplePassthroughQuery()
{
    QueryResult queryResult;
    try
    {
        string queryString = "select distinct r_object_id from dm_document order by r_object_id ";
        int startingIndex = 0;
        int maxResults = 60;
        int maxResultsPerSource = 20;
        PassthroughQuery q = new PassthroughQuery();
        q.QueryString = queryString;
        q.AddRepository(DefaultRepository);

        QueryExecution queryExec = new QueryExecution(startingIndex,
            maxResults,
            maxResultsPerSource);

        queryExec.CacheStrategyType = CacheStrategyType.NO_CACHE_STRATEGY;

        queryResult = searchService.Execute(q, queryExec, null);

        QueryStatus queryStatus = queryResult.QueryStatus;
        RepositoryStatusInfo repStatusInfo = queryStatus.RepositoryStatusInfos[0];
        if (repStatusInfo.Status == Status.FAILURE)
        {
            Console.WriteLine(repStatusInfo.ErrorTrace);
        }
    }
    catch (Exception e)
    {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
    return queryResult;
}
Structured query

Example 8-5. Java: Structured query

```java
public void simpleStructuredQuery()
{
    try
    {
        ServiceFactory serviceFactory = ServiceFactory.getInstance();
        ISearchService searchService = serviceFactory.getService(ISearchService.class, serviceContext);

        String repoName = defaultRepositoryName;

        // Create query
        StructuredQuery q = new StructuredQuery();
        q.setObjectRepository(repoName);
        q.setObjectType("dm_document");
        q.setIncludeHidden(true);
        q.setDatabaseSearch(true);
        ExpressionSet expressionSet = new ExpressionSet();
        expressionSet.addExpression(new PropertyExpression("owner_name",
            Condition.CONTAINS,
            "admin").
        q.setRootExpressionSet(expressionSet);

        // Execute Query
        int startingIndex = 0;
        int maxResults = 60;
        int maxResultsPerSource = 20;
        QueryExecution queryExec = new QueryExecution(startingIndex,
            maxResults,
            maxResultsPerSource);
        QueryResult queryResult = searchService.execute(q, queryExec, null);

        QueryStatus queryStatus = queryResult.getQueryStatus();
        RepositoryStatusInfo repStatusInfo = queryStatus.getRepositoryStatusInfos().get(0);
        if (repStatusInfo.getStatus() == Status.FAILURE)
        {
            System.out.println(repStatusInfo.getErrorTrace());
            throw new RuntimeException("Query failed to return result.");
        }
    }
    catch (Exception e)
    {
        Console.WriteLine(e.StackTrace);
        throw new Exception(e.Message);
    }
    return queryResult;
}
```

Structured query Example 8-5.
// print results
for (DataObject dataObject : queryResult.getDataObjects())
{
    System.out.println(dataObject.getIdentity());
}

} catch (Exception e)
{
    e.printStackTrace();
    throw new RuntimeException(e);
}

System.out.println("test completed - OK");

Example 8-6. C#: Structured query

public void SimpleStructuredQuery()
{
    try
    {
        String repoName = DefaultRepository;

        // Create query
        StructuredQuery q = new StructuredQuery();
        q.AddRepository(repoName);
        q.ObjectId = "dm_document";
        q.IsIncludeHidden = true;
        q.IsDatabaseSearch = true;
        ExpressionSet expressionSet = new ExpressionSet();
        expressionSet.AddExpression(new PropertyExpression("owner_name", Condition.CONTAINS, "admin");

        q.RootExpressionSet = expressionSet;

        // Execute Query
        int startingIndex = 0;
        int maxResults = 60;
        int maxResultsPerSource = 20;
        QueryExecution queryExec = new QueryExecution(startingIndex,
                                                     maxResults,
                                                     maxResultsPerSource);

        QueryResult queryResult = searchService.Execute(q, queryExec, null);

        QueryStatus queryStatus = queryResult.QueryStatus;
        RepositoryStatusInfo repStatusInfo = queryStatus.RepositoryStatusInfos[0];
        if (repStatusInfo.Status == Status.FAILURE)
        {
            Console.WriteLine(repStatusInfo.ErrorTrace);
            throw new Exception("Query failed to return result.");
        }

        // print results
        foreach (DataObject dataObject in queryResult.DataObjects)
        {
            Console.WriteLine(dataObject.Identity);
        }
    }
    catch (Exception e)
    {
        // Handle exception
    }
Search Service

Console.WriteLine(e.Message);
Console.WriteLine(e.StackTrace);
throw new Exception(e.Message);
}
Chapter 9

Workflow Service

The Workflow service provides a getProcessTemplates operation that obtains data about workflow process templates stored in repositories, a getProcessInfo operation for obtaining information about a specific process template, and a startProcess operation that starts a workflow process instance.

This chapter covers the following topics:

- Workflow SBO dependency, page 147
- getProcessTemplates operation, page 148
- getProcessInfo operation, page 150
- startProcess operation, page 151

Workflow SBO dependency

The Workflow service depends on the Workflow SBO, which must be accessed from a global registry. This SBO is installed as part of a Workflow docapp with Content Server version 6. Therefore the Workflow service is not supported on Content Server version 5.3, and requires a global registry.

To access a global registry for local service invocation, the local dfc.properties in the DFS SDK must specify the global repository name, as well as the global registry user name and password. For remote service invocation, the dfc.properties in emc-dfs.ear deployed on the application server must have these settings.

The global registry settings would normally be set during Content Server and DFS installation; if they were set at install time there is no need to modify dfc.properties hosted by the application server. However, for local service invocation you will need to modify the dfc.properties file in the SDK.

For more information see the EMC Documentum Foundation Services Installation Guide.
getProcessTemplates operation

Description

The getProcessTemplates operation is used to obtain a list of process templates (dm_process objects) installed in the repository. If a folderPath String is passed to the operation, only the process templates within the folderPath will be returned. The process templates in subfolders descended from folderPath will also be returned.

Java syntax

```java
DataPackage getProcessTemplates(String repositoryName,
                                 String folderPath,
                                 String additionalAttrs)
     throws BpmServiceException, ServiceException;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>repositoryName</td>
<td>String</td>
<td>The name of the repository in which the process templates are stored.</td>
</tr>
<tr>
<td>folderPath</td>
<td>String</td>
<td>A path to a folder in which the process templates are linked. If null, the operation will return all of the process templates stored in the repository. For example /mycabinet/myfolder.</td>
</tr>
<tr>
<td>additionalAttrs</td>
<td>String</td>
<td>A comma-separated list of attribute names. By default, getProcessTemplates returns only ObjectIdentity instances representing dm_process repository objects, and does not return any dm_process properties. The additionalAttrs parameter allows the client to pass in a list of dm_process property names to return in each DataObject.</td>
</tr>
</tbody>
</table>

Returns

Returns a DataPackage containing DataObject instances that represent the dm_process repository objects. Properties (attributes) of the dm_process object are returned if specified in the additionAttrs argument.
Example

Example 9-1. Java: Getting process templates

```java
public DataPackage processTemplates()
{
    try
    {
        ServiceFactory serviceFactory = ServiceFactory.getInstance();
        IWorkflowService workflowService = serviceFactory.getService(IWorkflowService.class, serviceContext);
        DataPackage processTemplates = workflowService.getProcessTemplates(defaultRepositoryName, null, "object_name");
        for (DataObject dObj : processTemplates.getDataObjects())
        {
            System.out.println(dObj.getIdentity().getValueAsString());
            System.out.println(dObj.getProperties().get("object_name"));
        }
        return processTemplates;
    }
    catch (Exception e)
    {
        e.printStackTrace();
        throw new RuntimeException(e);
    }
}
```

Example 9-2. C#: Getting process templates

```csharp
public DataPackage processTemplates()
{
    try
    {
        DataPackage processTemplates = workflowService.GetProcessTemplates(DefaultRepository, null, "object_name");
        foreach (DataObject dObj in processTemplates.DataObjects)
        {
            Console.WriteLine(dObj.Identity.GetValueAsString());
            Console.WriteLine(dObj.Properties.Get("object_name"));
        }
        return processTemplates;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Console.WriteLine(e.StackTrace);
        throw new Exception(e.Message);
    }
}```
getProcessInfo operation

Description

The getProcessInfo operation is used to obtain process information about a specific process template. Call this service when you have identified a workflow process that you intend to start. getProcessInfo returns a data structure that you can use to determine all values that are required to start the workflow. The caller populates the values of this object required by the workflow, then passes it back to the startProcess operation to start the workflow.

Java syntax

```java
ProcessInfo getProcessInfo(ObjectIdentity process)
    throws BpmServiceException, ServiceException;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>process</td>
<td>ObjectIdentity</td>
<td>An ObjectIdentity representing a dm_process object about which to obtain information. For DFS version 6, only ObjectIdentity instances of subtype ObjectId are supported.</td>
</tr>
</tbody>
</table>

Returns

Returns a ProcessInfo instance containing detailed information about a process template (that is, a dm_process repository object). This ProcessInfo instances can be populated and passed to the startProcess operation to start the workflow. Refer to the Javadocs for more information on ProcessInfo.

Example

Example 9-3. Java: Getting process information

```java
public ProcessInfo processInfo(ObjectIdentity processId)
{
    try
    {
        ServiceFactory serviceFactory = ServiceFactory.getInstance();
        IWorkflowService workflowService = serviceFactory.getService(IWorkflowService.class, serviceContext);
        ProcessInfo processInfo = workflowService.getProcessInfo(processId);
    }
```


System.out.println("Process template "+processId.getValueAsString());
System.out.println("Name is " + processInfo.getProcessInstanceName());
System.out.println("isAliasAssignmentRequired == "
 + processInfo.isAliasAssignmentRequired());
System.out.println("isPerformerAssignmentRequired == "
 + processInfo.isPerformerAssignmentRequired());
    return processInfo;
}

Example 9-4. C#: Getting process information

public ProcessInfo processInfo(ObjectIdentity processId)
{
    try
    {
        ProcessInfo processInfo = workflowService.GetProcessInfo(processId);

        Console.WriteLine("Process template "+processId.GetValueAsString());
        Console.WriteLine("Name is " + processInfo.ProcessInstanceName);
        Console.WriteLine("isAliasAssignmentRequired == "
            + processInfo.IsAliasAssignmentRequired);
        Console.WriteLine("isPerformerAssignmentRequired == "
            + processInfo.IsPerformerAssignmentRequired);
        return processInfo;
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        Console.WriteLine(e.StackTrace);
        throw new Exception(e.Message);
    }
}

startProcess operation

Description

The startProcess operation executes a business process (workflow), based on a ProcessInfo object obtained using the getProcessInfo operation. The process information is obtained from the data stored in the process template. Required values are populated by the caller, then passed to the startProcess operation to start the workflow.

Java syntax

ObjectIdentity startProcess(ProcessInfo info)
throws BpmServiceException, ServiceException

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>info</td>
<td>ProcessInfo</td>
<td>A data structure containing information about the workflow. The structure is obtained for the process template using the getProcessInfo operation, required values are populated by the caller, which then passes the object to the startProcess operation.</td>
</tr>
</tbody>
</table>

Returns

Returns an ObjectIdentity uniquely identifying the process template of the process that was started. For further information on ProcessInfo, refer to the Javadoc.

Example

Example 9-5. Java: Starting a process

```java
public void startProcess(String processId,
    String processName,
    String supervisor,
    ObjectId wfAttachment,
    List<ObjectId> docIds,
    String noteText,
    String userName,
    String groupName,
    String queueName) throws Exception {

    ServiceFactory serviceFactory = ServiceFactory.getInstance();
    IWorkflowService workflowService
        = serviceFactory.getService(IWorkflowService.class, serviceContext);

    // get the template ProcessInfo
    ObjectId objId = new ObjectId(processId);
    ProcessInfo info = workflowService
        .getProcessInfo(new ObjectIdentity<ObjectId>(objId, defaultRepositoryName));

    // set specific info for this workflow
    info.setSupervisor(supervisor);
    info.setProcessInstanceName(processName + new Date());

    // workflow attachment
    info.addWorkflowAttachment("dm_sysobject", wfAttachment);

    // packages
    List<ProcessPackageInfo> pkgList = info.getPackages();
    for (ProcessPackageInfo pkg : pkgList) {
        
```
pkg.addDocuments(docIds);
pkg.addNote("note for " + pkg.getPackageName() + " " + noteText, true);
}

// alias
if (info.isAliasAssignmentRequired())
{
    List<ProcessAliasAssignmentInfo> aliasList
        = info.getAliasAssignments();
    for (ProcessAliasAssignmentInfo aliasInfo : aliasList)
    {
        String aliasName = aliasInfo.getAliasName();
        String aliasDescription = aliasInfo.getAliasDescription();
        int category = aliasInfo.getAliasCategory();
        if (category == 1) // User
        {
            aliasInfo.setAliasValue(userName);
        }
        else if (category == 2 || category == 3) // group, user or group
        {
            aliasInfo.setAliasValue(groupName);
        }
        System.out.println("Set alias: "+ aliename
            + ", description: " + aliasDescription
            + ", category: " + category
            + " to "
            + aliasInfo.getAliasValue());
    }
}

// Performer.
if (info.isPerformerAssignmentRequired())
{
    List<ProcessPerformerAssignmentInfo> perfList
        = info.getPerformerAssignments();
    for (ProcessPerformerAssignmentInfo perfInfo : perfList)
    {
        int category = perfInfo.getCategory();
        int perfType = perfInfo.getPerformerType();
        String name = "";
        List<String> nameList = new ArrayList<String>();
        if (category == 0) // User
        {
            name = userName;
        }
        else if (category == 1 || category == 2) // Group, user or group
        {
            name = groupName;
        }
        else if (category == 4) // work queue
        {
            name = queueName;
        }
        nameList.add(name);
        perfInfo.setPerformers(nameList);
        System.out.println("Set performer perfType: " + perfType + ", category: " + category + " to " + name);
    }
}
Workflow Service

ObjectIdentity wf = workflowService.startProcess(info);
System.out.println("started workflow: " + wf.getValueAsString());
}

Example 9-6. C#: Starting a process

public void startProcess(String processId,
    String processName,
    String supervisor,
    ObjectIdentity wfAttachment,
    List<ObjectId> docIds,
    String noteText,
    String userName,
    String groupName,
    String queueName)
{
    // get the template ProcessInfo
    ObjectIdentity objId = new ObjectIdentity(processId);
    ProcessInfo info = workflowService
        .GetProcessInfo(new ObjectIdentity(objId, DefaultRepository));

    // set specific info for this workflow
    info.Supervisor = supervisor;
    info.ProcessInstanceName = processName + new DateTime();

    // workflow attachment
    info.AddWorkflowAttachment("dm_sysobject", wfAttachment);

    // packages
    List<ProcessPackageInfo> pkgList = info.Packages;
    foreach (ProcessPackageInfo pkg in pkgList)
    {
        pkg.AddDocuments(docIds);
        pkg.AddNote("note for " + pkg.PackageName + " " + noteText, true);
    }

    // alias
    if (info.IsAliasAssignmentRequired())
    {
        List<ProcessAliasAssignmentInfo> aliasList
            = info.AliasAssignments;
        foreach (ProcessAliasAssignmentInfo aliasInfo in aliasList)
        {
            String aliasName = aliasInfo.AliasName;
            String aliasDescription = aliasInfo.AliasDescription;
            int category = aliasInfo.AliasCategory;
            if (category == 1) // User
            {
                aliasInfo.AliasValue = userName;
            }
            else if (category == 2 || category == 3) // group, user or group
            {
                aliasInfo.AliasValue = groupName;
            }

            Console.WriteLine("Set alias: "
                + aliasName
                + ", description: "
                + aliasDescription
                + ", category: "
                + category
                + " to "
                + aliasInfo.AliasValue);
        }
    }
// Performer.
if (info.IsPerformerAssignmentRequired())
{
    List<ProcessPerformerAssignmentInfo> perfList
        = info.PerformerAssignments;
    foreach (ProcessPerformerAssignmentInfo perfInfo in perfList)
    {
        int category = perfInfo.Category;
        int perfType = perfInfo.PerformerType;
        String name = "";
        List<String> nameList = new List<String>();
        if (category == 0) // User
        {
            name = userName;
        }
        else if (category == 1 || category == 2) // Group, user or group
        {
            name = groupName;
        }
        else if (category == 4) // work queue
        {
            name = queueName;
        }
        nameList.Add(name);
        perfInfo.Performers = nameList;
        Console.WriteLine("Set performer perfType: " + perfType + ", category: " + category + " to " + name);
    }
}

ObjectIdentity wf = workflowService.StartProcess(info);
Console.WriteLine("started workflow: " + wf.GetValueAsString());
Content transfer is one of the critical areas of functionality in content management services. It has a significant impact on the scalability and agility of services, and a particularly significant impact on performance and user experience. DFS integrates standard and propriety technologies to support optimization of both point-to-point content transfer, as well as end-to-end transfer in more complex service architectures that may involve multiple servers and a potentially multiple hops of content between locations. It does this by leveraging some sophisticated technologies like EMC Documentum Unified Client Facilities (UCF), Accelerated Content Services (ACS), and Branch Office Caching Services (BOCS) while attempting to make use of these technologies as transparent as possible for the service consumer.

DFS also provides some usability features related to content transfer, specifically to support post-transfer commands (such as the ability to open content for editing or viewing after transfer), and support for asynchronous and synchronous events, such as displaying a progress bar or modal dialog on the user’s system.

This chapter covers the following topics:

- Content transfer topologies and optimization, page 157
- ContentTransferProfile, page 162
- Content transfer modes, page 158
- Content model, page 161
- ContentTransferProfile, page 162
- UCF content transfer in DFS, page 163

**Content transfer topologies and optimization**

Moving content around in a large enterprise can be complex, and it can be inefficient. In a service-oriented architecture content may move in hops along a chain of linked services. In web applications, the client is a browser whose connection to the repository is mediated by a web application hosted by an application server, which could necessitate an additional hop. And the repository and its filestore may be located at a distance from the point to which it must transfer content, resulting in a slow hop.
To solve these problems, DFS makes use of technologies that do the following:

- minimize hops between systems participating in the transfer, ideally by establishing a direct transfer between the user’s system and a server that has access to the content filestore
- minimize distance by using a content cache near the location of the user

The technologies employed are:

- UCF (Unified Client Facilities), described in .
- ACS (Accelerated Content Services). This is a lightweight server that handles read and write content operations. ACS is hosted in a content server dedicated to handling content (not metadata).
- BOCS (Branch Office Caching Services). A BOCS server is a caching server that communicates only with ACS servers. Like the ACS server, it does not handle metadata requests. A BOCS server enables synchronous or asynchronous read or write of content to a local cache, situated in a geolocation (also sometimes called a network location or client network location) near the user’s system.

A detailed description of ACS and BOCS are outside the scope of this manual. However, you can find these topics covered in great detail in the Documentum Distributed Configuration Guide.

For a DFS consumer or service to make use of these solutions, it must use UCF content transfer.

**Note:** EMC has implemented this functionality using UCF because it could not be done using any existing content standard. We embrace available, appropriate standards, but we do not limit our functional value to only that which these standards can yield.

## Content transfer modes

DFS integrates the following standard and proprietary content transfer technologies:

- base64
- MTOM
- UCF

A service consumer can choose a transfer mode for a set of services in a ContentTransferProfile instances, which is stored in the service context or passed in operationOptions. The ContentTransferMode can also be set in the Content object. If ContentTransferMode is set in both the ContentTransferProfile and the Content object, the setting in ContentTransferProfile takes precedence.

A newly instantiated ContentTransferProfile object will have a default ContentTransferMode of MTOM.

The following sections describe these content transfer modes.

### base64

base64 is an established encoding for transfer of opaque data inline within a SOAP message (or more generally within XML). The encoded data is tagged as an element of the xs:base64Binary XML schema data type. base64 encoded data is not optimized, and in fact is known to expand
binary data by a factor of 1.33x original size. This makes base64 inefficient for sending larger data files. As a rule, it is optimal to use base64 for content smaller than around 5K bytes. For larger content files, it is more optimal to use MTOM.

**MTOM**

MTOM, an acronym for SOAP Message Transmission Optimization Mechanism, is a W3C recommendation adopted by JAX-WS. For more information see [http://www.w3.org/TR/soap12-mtom/](http://www.w3.org/TR/soap12-mtom/).

Enabling MTOM means that both the request that the SOAP client is sending to the server and the returned response go through MTOM encoding and decoding. For larger files, MTOM optimization is beneficial; however, for small files (typically those under 5K), there is a serious performance penalty for using MTOM, because the overhead of serializing and deserializing the MTOM message is greater than the benefit of using the MTOM optimization mechanism.

In DFS, it is entirely up to the consumer to determine whether using MTOM is the right thing to do from a performance standpoint. DFS makes the determination at runtime based on settings provided by the consumer in a ContentTransferProfile, or in the Content object itself.

WSDL-only clients (those that do not make use of the DFS client runtime) will have to explicitly enable MTOM.

**Note:** Developers familiar with JAX-WS may be aware that JAX-WS permits specification of an MTOM threshold in a deployment descriptor on the server. DFS does not use this setting, because it would cause the service to use MTOM regardless of settings provided by the consumer, which could break interoperability for consumers on which MTOM is not enabled; DFS instead gives the consumer direct control over the content transfer mode.

**Memory limitations associated with MTOM content transfer mode**

The DFS .NET client is based on Windows Communication Framework (WCF), which provides two modes for MTOM content transfer: buffered and streaming. To enable streaming, WCF requires that the parameter that holds the data to be streamed must be the only parameter in the method (such as Get or create). This conflicts with the design of DFS, such that DFS can only use the MTOM buffer mode with a .NET client. This results in unusually high memory requirements, especially when trying to transfer large content payloads when ACS is unavailable, because the entire content must be buffered in memory before transfer. Normally a .NET client will use ACS if it is available for content download operations, so under typical conditions the memory limitation is not encountered. However, if ACS content is unavailable, or if the client attempts to upload a very large content stream to the server using MTOM content transfer mode, the server’s capacity to buffer the content may be exceeded.
**Workarounds**

There are several options for working around this limitation:

- First, for content download operations, enable ACS/BOCS and make use of it. To ensure that the urlContent type is returned by DFS, use the urlReturnPolicy setting as described under. The client can use the urlContent returned by DFS to request content transfer from the ACS server.

- For content upload operations, use UCF as the content transfer mode. UCF will orchestrate content transfer in both directions between the client and the ACS server.

- If you don’t wish to use either of the preceding workarounds, make sure that both the DFS .NET client and JVM that runs DFS server have enough memory to buffer the content. However, be aware that in this case the application will be limited to transfer of content in the range of hundreds of megabytes for a 32-bit JVM, because on most modern 32-bit Windows systems the maximum heap size will range from 1.4G to 1.6G (see [http://www.oracle.com/technetwork/java/hotspotfaq-138619.html#gc_heap_32bit](http://www.oracle.com/technetwork/java/hotspotfaq-138619.html#gc_heap_32bit)). Although this specific limitation will not apply to a 64-bit versions of Windows, the issue will still exist if you do not ensure that there is sufficient heap space to buffer very large objects in memory.

- You can create a custom service. Due to a WCF limitation (see [http://msdn.microsoft.com/en-us/library/ms789010.aspx](http://msdn.microsoft.com/en-us/library/ms789010.aspx)) wherein the stream data transfer mode is supported only when there is a single parameter in the web service method signature. Therefore, in the custom service, all parameters must be wrapped into a single custom class object containing all input parameters of a method as follows:

```java
@DfsPojoService()
public class StreamingService {
    public DataPackage create(DataRequest request) throws ServiceException {
        // DataRequest wraps DataPackage and OperationOptions, the DataPackage might contain large content
        // do something with the content uploaded
        .......
    }
}

@XmlType(name = "DataRequest", namespace = "http://streaming.fs.documentum.emc.com")
@XmlAccessorType(XmlAccessType.FIELD)
public class DataRequest {
    private DataPackage dataPackage;
    private OperationOptions options;
    public DataPackage getDataPackage() {
        return dataPackage;
    }
    public void setDataPackage(DataPackage dataPackage) {
        this.dataPackage = dataPackage;
    }
    public OperationOptions getOptions() {
        return options;
    }
    public void setOptions(OperationOptions options) {
```
In the App.config file, to enable streaming, set the transferMode attribute of DfsDefaultService binding to Streamed.

Note:
- For downloading and uploading content, increase the time-out related attributes (closeTimeout, openTimeout, receiveTimeout, and sendTimeout) for DfsDefaultService binding based on the requirement.
- For content downloading, in App.config file, increase the value of the maxReceivedMessageSize attribute of DfsDefaultService binding to a larger value such as 1000000000 bytes. This modification is required because the maxReceivedMessageSize attribute determines the maximum size, in bytes, for a message that can be received on a channel configured with streamed binding.

**UCF**

Unified Client Facilities (UCF) is a lightweight client-server application that transfers content between a DFS consumer, a DFS service, and a content repository. The UCF APIs provide a remote UCF server on the service host with access to the client file system and registry, and provides support for:

- client-orchestrated content transfer in a web application or service chain
- integration with ACS and BOCS for optimized transfer in distributed architectures
- transfer of complex content types, such as XML content with file references and Microsoft Office documents with internal links
- post-transfer actions (which generally means opening the content in a viewer or editor)

UCF content transfer is available to consumer application using the Java and C# client libraries, as well as to WSDL-only clients. For more information see UCF content transfer in DFS, page 163.

**UCF is not supported in middle tier**

UCF is a proprietary distributed content transfer technology. UCF client is intended for a single user, either using a browser in a web application, or using a thick client. Use of UCF is not supported on the middle tier of a distributed application. Typically, the middle tier would be a web application functioning as a DFS consumer.

**Content model**

The DFS content model emphasizes flexibility by providing a number of different abstractions for representing content (see Figure 13, page 162). The client has the convenience provide any content type to a service operation that transfers content. However, the transfer can be optimal or non-optimal depending on the suitability of content type to the content transfer mode.
MTOM and base64 stream binary content, whereas UCF transfers files. If a FileContent instance is passed to an MTOM content transfer operation, the file will need to be streamed into memory for subsequent binary stream transfer using MTOM, which will incur some cost. It is recommended to use FileContent or UcfContent objects for UCF transfer, and BinaryContent for MTOM and base64.

Note: The DataHandler is a Java convenience class that provides a consistent interface to data available in many different sources and formats.

**Figure 13. Content class hierarchy**

![Content class hierarchy diagram]

**ContentTransferProfile**

Distributed content behavior is controlled through the ContentTransferProfile, which would normally be set in the service context (rather than OperationOptions). It also contains the following fields pertinent to distributed content.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>transferMode</td>
<td>ContentTransferMode</td>
<td>The transfer mode. Possible values are MTOM, BASE64, and UCF. See <a href="#">Content transfer modes</a>, page 158.</td>
</tr>
<tr>
<td>geolocation</td>
<td>String</td>
<td>Geolocation represents an area on the network's topography, generally a set of machines that are located near one another. The Geolocation is used to determine the closest location of the content storage on a repository or in a BOCS cache.</td>
</tr>
</tbody>
</table>
### UCF content transfer in DFS

UCF (Unified Client Facilities) can be used to handle content transfer in DFS consumers that use the client runtime support (either Java or .NET), and can also be used by WSDL consumers. If a DFS consumer is using client-side runtime support, most of the interaction between DFS and the UCF application is handled by the DFS framework; however, in certain cases the consumer may choose to exert more fine control of the process (see [Client-orchestrated UCF transfer](#), page 164 and [Optimization: controlling UCF connection closure](#), page 175).

### UCF transfer to and from DFS consumers

This section provides an overview of how UCF transfer is managed in the context of a DFS service operation. The process described is similar whether the process is handled by the DFS framework, or whether it is orchestrated by the DFS consumer.

When a DFS consumer requests content transfer using UCF (having specified UCF as the `ContentTransferMode` in the relevant `ContentTransferProfile`), DFS launches the UCF process on the client, providing it with an http session id on the service host (obtained by calling the AgentService), and a UCF server URL. The UCF client process uses this data to establish a connection with the UCF server on the service host, which is represented by a UCF connection ID that it passes back to the DFS caller. The DFS runtime stores the http session id and UCF connection id in an ActivityInfo, which is encapsulated in the `ContentTransferProfile` and placed in the `ServiceContext`. This activity is summarized in Figure 14, page 164.
The service that receives the request for content transfer passes the data stored in the ActivityInfo to the UCF system on the service host, which uses the established UCF connection to transfer data between a Content Server and the requesting client (see Figure 15, page 164). By default the UCF session is closed after the service call (for information on overriding this behavior, see Optimization: controlling UCF connection closure, page 175).

Figure 15. UCF transfer to client

Client-orchestrated UCF transfer

In both the Java and .NET clients, the runtime framework handles the UCF launch and transfer to and from a DFS consumer without any direct intervention from the consumer. In some cases, however, the consumer may wish to orchestrate the process by specifying a UCF connection other than the standard connection that the framework establishes between the DFS client host and the DFS service host. This technique is primarily necessary in n-tier applications where the DFS consumer is located on a middle tier, and content transfer is required between the DFS service host and the end-user.
Web applications with DFS services and consumer running in separate applications

A typical topology for a DFS production application will be one in which the DFS consumer is on a middle tier in a web application, and the DFS services are located on a separate tier and accessed by the consumer remotely, as shown in Figure 16, page 165.

Figure 16. Topology with web application and service on separate hosts

In this configuration, the UCF connection must be set up so that requests for content transfer initiated by the DFS consumer will result in content transfer between the end-user machine and the core service tier. In this topology requests from the browser may go either to the web application host, or to the DFS service host. This requires a mechanism for directing requests from the browser to the correct URL. To solve this problem, you can use a reverse proxy that supports a forwarding mechanism that remaps URLs received in browser requests to the appropriate address. (Apache provides such a capability.) This solution is shown in Figure 17, page 166.
The following section (Enabling UCF transfer in a web application, page 166) provides some more explicit instructions on enabling UCF content transfer in the topology just described.

**Note:** Note that a simplified case is also supported, in which the web application DFS consumer and DFS services are running at the same location. (In this case the DFS services will be consumed locally rather than remotely.) In this case it is not required to use a reverse proxy as a forwarder, as described above, as the services, the UCF server, and the consumer application would all be located at the same address and port.

### Enabling UCF transfer in a web application

This section provides instructions for creating a test application that enables UCF transfer, using the topology described under Web applications with DFS services and consumer running in separate applications, page 165. The sample application described here is not currently provided in the SDK.

**Note:** This sample application will also support, in a simplified form, a case where a web application DFS consumer and DFS are running at the same location. (In this case the DFS services will be consumed locally rather than remotely.) In this case it is not required to use a reverse proxy as a forwarder, as described below in Configure the Apache reverse proxy as a request forwarder, page 168, as the services, the UCF server, and the consumer application would all be located at the same address and port.

The test application includes

- an end-user machine running a browser, with an available Java Runtime Environment (JRE)
- an Apache application server used as a reverse proxy
- an application server that hosts a web application that includes a minimal user interface and a DFS consumer application
• an application server hosting DFS services
• a Content Server (repository)

For our tests of this scenario, we deployed the web application on Tomcat, and deployed DFS in freestanding mode in WebLogic. The test application shown here also requires the Java Plug-in, documented here: http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/contents.html

The Java Plug-in is part of the Java Runtime Environment (JRE), which is required on the end-user machine.

The sample application is designed to run as follows:

1. The browser sends a request to a JSP page, which downloads an applet. If the browser is configured to check for RSA certificates, the end user will need to import the RSA certificate before the applet will run. (The signing of the applet with the RSA certificate is discussed in Sign the applet, page 172.)

2. The applet instantiates a UCF connection, gets back a jsessionId and a uid, then sends these back to the JSP page by calling a JavaScript function.

3. In the web application, a servlet use the jsessionId, uid, and a filename provided by the user to create an ActivityInfo object, which is placed in a ContentTransferProfile in a service context. This enables DFS to perform content transfer using the UCF connection established between the UCF server on the DFS service host and the UCF client on the end-user machine.

The tasks required to build this test application are described in the following sections:

1. Set up the development environment, page 167.
2. Configure the Apache reverse proxy as a request forwarder, page 168
3. Code an HTML user interface for serving the applet, page 168
4. Write the applet code for deploying and launching UCF, page 171
5. Build and bundle the applet, page 172
6. Sign the applet, page 172
7. Create a servlet for orchestrating the UCF content transfer, page 172

**Set up the development environment**

The environment required for the test consists of the following:

• An end-user machine, which includes a browser, and which must have a Java Runtime Environment available in which to run UCF (and the Java Plug-in). The browser should be configured to use a Java 5 JRE for applets (we tested this sample application using JRE 1.5.0_10).

• A proxy set up using the Apache application server (we tested using version 2.2).

• An application server hosting the web application components, including the DFS consumer.

• An application server hosting the DFS services and runtime (which include the required UCF server components). This can be a freestanding DFS installation, or DFS running on a Content
Server. The DFS installation must have its dfc.properties configured to point to a connection broker through which the Content Server installation can be accessed.

- A Content Server installation.

To create a test application, each of these hosts must be on a separate port. They do not necessarily have to be on separate physical machines. For purposes of this sample documentation, we assume the following:

- The proxy is at http://localhost:80.
- The web application is at http://localhost:8080.
- The DFS services (and the UCF components, which are included in the DFS ear file) are a freestanding installation at http://localhost:8888/services/core

**Configure the Apache reverse proxy as a request forwarder**

The Apache reverse proxy can be configured as a forwarder by adding text similar to the following to httpd.conf. Additional text may be necessary to configure access rights. (Note that the ⇒ symbol in the following listing indicates a line continuation.)

```
ProxyPass /runtime/AgentService.rest
  ⇒http://localhost:8888/services/core/runtime/AgentService.rest
ProxyPass /ucf.installer.config.xml
  ⇒http://localhost:8888/services/core/ucf.installer.config.xml
ProxyPass /servlet/com/documentum/ucf/
  ⇒http://localhost:8888/services/core/servlet/com/documentum/ucf/
ProxyPass / http://localhost:8080/
```

LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so

With this mapping set up, the browser will only need to know about a single URL, which is the root folder of the proxy server at http://localhost:80. The proxy will take care of forwarding requests to the appropriate URL, based on the specific paths appended to the proxy root.

For example, assuming the proxy is listening a http://localhost:80, the proxy will forward a request to http://localhost:80/runtime/AgentService.rest to http://localhost:8888/services/core/runtime/AgentService.rest. The default mapping is to the application server that hosts UI and DFS consumer, so it will map http://localhost:80/dfsWebApp/DfsServiceServlet to http://localhost:8080/dfsWebApp/DfsServiceServlet.

**Code an HTML user interface for serving the applet**

The sample HTML presents the user with two buttons and a text box. When the user clicks the **Use Ucf** button, a second popup is launched while the UCF connection is established by the applet. When the applet finishes, the second windows closes and the user can import a file specified by a file path entered in the text box.
Figure 18. User interface for UCF test application

![User interface for UCF test application](image)

**Note:** This sample has been implemented with two buttons for demonstration purposes. A button with the sole function of creating the UCF connection would probably not be a useful thing to have in a production application. Make sure not to click this button then close the browser without performing the import: this will leave the UCF client process running.

**Example 10-1. HTML for user interface**

```html
<html>
<head>
<title>Sample Applet Main Page</title>
<script type="text/javascript">

var winPop;

function OpenWindow()
{
    var props = "top=0,
                   left=0,
                   toolbar=1,
                   location=1,
                   directories=1,
                   status=1,
                   menubar=1,
                   scrollbars=0,
                   resizable=0,
                   width=300,
                   height=400";

    winPop = window.open("dfsSample-popup.html", "winPop", props);
}

function validate()
{
    if(document.form1.jsessionId.value == "" || document.form1.uid.value="")
    {
        alert("UCF connection is not ready, please wait");
        return false;
    }
```
else if (document.form1.file.value == "")
{
    alert("Please enter a file path");
    return false;
}
else
{
    return true;
}
}

</script>
</head>
<body>
<h2>DFS Sample</h2>
<form name="form1" onSubmit="return validate()"
    method="post"
    action="http://localhost:80/dfsWebApp/DfsServiceServlet">
    Enter File Path: <input name="file" type="text" size=20><br/>
    <input name="jsessionId" type="hidden"><br/>
    <input name="uid" type="hidden"><br/>
    <input type="button" value="Use Ucf" onclick="OpenWindow()">
    <input type="submit" value="Import">
</form>
</body>
</html>

Note that hidden input fields are provided in the form to store the jsessionId and uid values that will be obtained by the applet when it instantiates the UcfConnection.

Example 10-2. HTML for calling applet (dfsSample-popup.html)

<html>
<head>
<TITLE>Sample Applet PopUp Page</TITLE>
<script type="text/javascript">

function setHtmlFormIdsFromApplet()
{
    if (arguments.length > 0)
    {
        window.opener.document.form1.jsessionId.value = arguments[0];
        window.opener.document.form1.uid.value = arguments[1];
    }
    window.close();
}

</script>
</head>
<body>
<center><h2>Running Applet ........</h2></center>
<center>
<applet CODE=SampleApplet.class
    CODEBASE=/dfsWebApp
    WIDTH=40
The popup HTML downloads the applet, and also includes a Javascript function for setting values obtained by the applet in dfsSample.html (see HTML for user interface, page 169). The applet will use the Java Plug-in to call this JavaScript function.

Write the applet code for deploying and launching UCF

The applet must perform the following tasks:

1. Instntiates a UcfConnection, passing the constructor the value that the root URL of the proxy.
2. Get the values for the UCF connection (uid) and http session (jsessionId) and sets these values in the html form by calling the Javascript function defined in the JSP page.

This applet code depends on classes included in ucf-connection.jar and ucf-installer.jar (these will be added to the applet in the subsequent step).

Note that this Java code communicates with the Javascript in the JSP using the Java Plug-in (JSObject). For more information on the Java Plug-in, see http://java.sun.com/j2se/1.4.2/docs/guide/plugin/developer_guide/contents.html.

```java
import com.emc.documentum.fs.rt.ucf.UcfConnection;
import java.applet.*;
import java.net.URL;
import netscape.javascript.JSONObject;

public class SampleApplet extends Applet {

    public void init () {
        //init UCF
        System.out.println("SampleApplet init.......");
        try {
            UcfConnection conn = new UcfConnection(new URL("http://localhost:80"));
            System.out.println("jsessionId=" + conn.getJsessionId() + ", uid=" + conn.getUid());
            JSONObject win = JSONObject.getWindow(this);
            win.call("setHtmlFormIdsFromApplet", new Object[]{conn.getJsessionId(),
                conn.getUid()});
        }
        catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void start () {

EMC Documentum Foundation Services Version 6 SP1 Development Guide 171
Build and bundle the applet

The applet that you construct must contain all classes from the following archives, provided in the SDK:

- ucf-installer.jar
- ucf-connection.jar

To create the applet, extract the contents of these two jar files and place them in the same folder with the compiled SampleApplet class, shown in the preceding step. Bundle all of these classes into a new jar file called dfsApplet.jar.

Sign the applet

Applets must run in a secure environment, and therefore must include a signed RSA certificate issued by a certification authority (CA), such as VeriSign or Thawte. The certificate must be imported by the end user before the applet code can be executed. You can obtain a temporary certificate for test purposes from VeriSign, and sign the jar file using the Java jarsigner utility. Detailed instructions regarding this are available at http://java.sun.com/javase/6/docs/technotes/guides/plugin/developer_guide/rsa_signing.html#signing.

Create a servlet for orchestrating the UCF content transfer

The function of the servlet is to perform the following tasks:

1. Receive the jsessionId and uid from the browser and use this data to configure an ActivityInfo, ContentTransferProfile, and ServiceContext such the DFS service will use the UCF connection established between the UCF client running on the end-user machine and the UCF server hosted in the DFS server application.

2. Instantiate the DFS Object service and run a create operation to test content transfer.

Example 10.3. Sample servlet code for orchestrating UCF transfer

```java
import com.emc.documentum.fs.datamodel.core.content.ActivityInfo;
import com.emc.documentum.fs.datamodel.core.content.ContentTransferMode;
import com.emc.documentum.fs.datamodel.core.content.Content;
import com.emc.documentum.fs.datamodel.core.content.FileContent;
import com.emc.documentum.fs.datamodel.core.context.RepositoryIdentity;
import com.emc.documentum.fs.datamodel.core.profiles.ContentTransferProfile;
import com.emc.documentum.fs.datamodel.core.DataPackage;
```
import com.emc.documentum.fs.datamodel.core.DataObject;
import com.emc.documentum.fs.datamodel.core.ObjectIdentity;
import com.emc.documentum.fs.rt.context.IServiceContext;
import com.emc.documentum.fs.rt.context.ContextFactory;
import com.emc.documentum.fs.rt.context.ServiceFactory;
import com.emc.documentum.fs.rt.ServiceInvocationException;
import com.emc.documentum.fs.services.core.client.IObjectService;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;
import java.io.IOException;
import java.io.PrintWriter;

public class DfsServiceServlet extends HttpServlet {
    public void doPost (HttpServletRequest req,
            HttpServletResponse res)
                throws ServletException, IOException {
        String file = req.getParameter("file");
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        try {
            IObjectService service = getObjectService(req);
            DataPackage dp = new DataPackage();
            DataObject vo = new DataObject(new ObjectIdentity(docbase), "dm_document");
            vo.getProperties().set("object_name", "testobject");
            int fileExtIdx = file.lastIndexOf(".");
            Content content = new FileContent(file, file.substring(fileExtIdx + 1));
            vo.getContents().add(content);
            dp.addDataObject(vo);
            DataPackage result = service.create(dp, null);
            System.out.println("result: " + result);
            if (result != null) {
                out.println("Create success: " + result.getDataObjects().get(0).getIdentity().getValueAsString());
            } else {
                out.println("Create failed ");
            }
        } catch (Exception ce) {
            throw new ServletException(ce);
        }
    }

    public void doGet (HttpServletRequest req,
            HttpServletResponse res)
                throws ServletException, IOException {
        doPost(req, res);
    }

    private IObjectService getObjectService (HttpServletRequest req)
throws ServiceInvocationException
{
    String jsessionId = req.getParameter("jsessionId");
    String uid = req.getParameter("uid");
    System.out.println("params:" + jsessionId + ", " + uid);
    IServiceContext context = ContextFactory.getInstance().newInstance();
    context.addIdentity(new RepositoryIdentity(docbase, username, password, ")");
    ActivityInfo activity = new ActivityInfo(jsessionId, null, uid, true);
    ContentTransferProfile ct = new ContentTransferProfile();
    ct.setTransferMode(ContentTransferMode.UCF);
    ct.setActivityInfo(activity);
    context.setProfile(ct);
    IObjectService service = ServiceFactory.getInstance().getRemoteService(
        IObjectService.class, context, "core", serverUrl + "/services");
    return service;
}

private static String username = "_USERNAME_";
private static String password = "_PASSWORD_";
private static String docbase = "_DOCBASE_
private static String serverUrl = "http://localhost:8888";

Note that you will need to provide values for username, password, and docbase fields to enable DFS to connect to your test repository.

In the sample, the getObjectService method does the work of obtaining the jsessionId and the uid from the http request.

String jsessionId = req.getParameter("jsessionId");
String uid = req.getParameter("uid");

It then constructs an ActivityInfo object, which it adds to a ContentTransferProfile, which in turn is added to the service context.

IServiceContext context = ContextFactory.getInstance().newInstance();
context.addIdentity(new RepositoryIdentity(docbase, username, password, ")");

ActivityInfo activity = new ActivityInfo(jsessionId, null, uid, true);
ContentTransferProfile ct = new ContentTransferProfile();
ct.setTransferMode(ContentTransferMode.UCF);
ct.setActivityInfo(activity);
context.setProfile(ct);

Notice that in addition to the jsessionId and uid, the ActivityInfo is instantiated with two other values. The first, which is passed null, is the initiatorSessionId. This is a DFS internal setting to which the consumer should simply pass null. The second setting, which is pass true, is autoCloseConnection. Setting this to true (which is also the default), cause DFS to close the UCF connection after the service operation that transfers content. For more information on using this setting see Optimization: controlling UCF connection closure, page 175.

Finally, getObjectService instantiates the Object service using the newly created context.

IObjectService service = ServiceFactory.getInstance().getRemoteService(
    IObjectService.class, context, "core", serverUrl + "/services");
The key is that the context has been set up to use the UCF connection to the UCF client running on the end user machine, obtained by the applet rather than the standard connection to the UCF client machine.

The doPost method finishes by using the service to perform a test transfer of content, using the Object service create method.

**Alternative methods of supplying ActivityInfo and their relative precedence**

A client that constructs its own ActivityInfo instance can supply it to the service by directly adding it to a ContentTransferProfile, or by adding it to an instance of UcfContent. The ContentTransferProfile is generally added to the service context, but may also be passed with an OperationOptions instance.

In all cases, if the client-supplied ActivityInfo has a properly initialized activityInfo and sessionId settings, and if its closed flag is set to false, and if the ContentTransferMode is set to UCF, the DFS framework will use the client-supplied settings and will not launch the UCF session on the client. (It will assume that the client has taken responsibility for this.)

In the case that an ActivityInfo is supplied in both the ContentTransferProfile and the UcfContent, the framework will use the ActivityInfo that is stored in the ContentTransferProfile.

**Optimization: controlling UCF connection closure**

The default behavior of the DFS framework is to close an active UCF connection (from the server side) after it has been used by a service operation and to terminate the client UCF process. In some applications this can incur an unnecessary overhead. This behavior can be overridden using the ActivityInfo.autoCloseConnection flag. The consumer can set up the ActivityInfo and supply it to the service using either method described in Alternative methods of supplying ActivityInfo and their relative precedence, page 175. The ActivityInfo should have the following settings:

<table>
<thead>
<tr>
<th>ActivityInfo field</th>
<th>Supplied value</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoCloseConnection</td>
<td>false</td>
</tr>
<tr>
<td>closed</td>
<td>false</td>
</tr>
<tr>
<td>activityId</td>
<td>null</td>
</tr>
<tr>
<td>sessionId</td>
<td>null</td>
</tr>
<tr>
<td>initiatorSessionId</td>
<td>null</td>
</tr>
</tbody>
</table>

The client runtime provides a constructor that permits the consumer to set autoCloseConnection only, and the remaining settings are provided by default. With these settings, the DFS framework will supply standard values for activityId and sessionId, so that content will be transferred between the standard endpoints: the UCF server on the DFS host, and the UCF client on the DFS consumer.

If the consumer sets autoCloseConnection to false, the consumer is responsible for closing the connection. This can be accomplished by setting autoCloseConnection to true before the consumer
application's final content transfer using that connection. If the consumer fails to do this, the UCF connection will be left open, and the UCF client process will not be terminated.

This optimization removes the overhead of launching the UCF client multiple times. It is only effective in applications that will perform multiple content transfer operations between the same endpoints. If possible, this overhead can be more effectively avoided by packaging multiple objects with content in the DataPackage passed to the operation.

## Opening a transferred document in a viewer/editor

You can specify an action to perform on the client after an operation that transfers content using the `setContentTransferAction` method of `ContentProfile`. This feature is available only if the content is transferred using the UCF transfer mode. The `setContentTransferAction` method takes a String argument, which can have any of the values described in Table 9, page 176.

### Table 9. PostTransferAction strings

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null or empty string</td>
<td>Take no action.</td>
</tr>
<tr>
<td>dfs:view</td>
<td>Open the file in view mode using the application associated with the file type by the Windows operating system.</td>
</tr>
<tr>
<td>dfs:edit</td>
<td>Open the file in edit mode using the application associated with the file type by the Windows operating system.</td>
</tr>
<tr>
<td>dfs:edit?app=<em>EXE</em></td>
<td>Open the file for editing in a specified application. To specify the application replace <em>EXE</em> with a fully-qualified path to the application executable; or with just the name of the executable. In the latter case the operating system will need to be able to find the executable; for example, in Windows, the executable must be found on the %PATH% environment variable. Additional parameters can be passed to the application preceded by an ampersand (&amp;).</td>
</tr>
</tbody>
</table>
Chapter 11

Building Custom Services Using DFS

This chapter is intended to introduce you to the design-time tools provided in the DFS SDK for creating custom services, and to give a practical example of how to create a service using the templates provided in the SDK. The DFS design tools consist of a set of Apache Ant tasks used to generate services from source artifacts, as well as a sample service that can be used as a template for developing and configuring your own service-generation build environment. This chapter will discuss some of the design and implementation principles that the developer will need to consider when creating Java classes and other artifacts as input to the design-time tools, it will discuss the service tools themselves, and it will examine the structure and configuration of the sample service, which can be found in the SDK in a directory called AcmeCustomService.

This chapter will also discuss the development of a Java test consumer that exercises the test service using the client runtime library. It covers the following topics:

• Service design considerations, page 177
• Creating inputs to DFS tools, page 178
• Sample service, page 186
• Service test consumers, page 188
• Tools for generating services, page 190
• Exploring AcmeCustomService, page 196

Service design considerations

The following sections discuss a few of the design considerations you may need to take into account when planning your custom service.

SBO or POJO services

DFS services can be implemented either as Business Object Framework (BOF) Service-based Business Objects (SBOs), or as Plain Old Java Objects (POJOs). The following two factors may have bearing on your decision regarding which approach to take.

• your organization’s current investment in SBOs
• the degree from which your organization would benefit from the SBO deployment model
If you have existing SBOs that are used in DFC clients or projected as EMC Documentum 5.3 web services, the optimal route to DFS may be to convert the existing services into DFS services. However, bear in mind that not all SBOs are suitable for projection as web services, and those that are technically suitable may still be lacking an optimal SOA design. As an alternative strategy you could preserve current SBOs and make their functionality available as a DFS service by creating DFS services as facades to the existing SBOs.

The SBO approach may also be of value if you wish to design services that are deployed across multiple repositories and multiple DFC client applications (including WDK-based applications). An SBO implementation is stored in a single location, the global registry, from which it is dynamically downloaded to client applications. If the implementation changes, the changes can be deployed in a single location. The BOF runtime framework automatically propagates the changed implementation to all clients. (Note that the SBO interface must be deployed to each DFC client.)

If neither of these considerations is compelling, POJO services may be the more attractive choice, as it removes the small technical overhead and vendor-specific requirements of implementing and deploying SBOs. Note that choosing POJO services will in no way preclude the effective use of BOF objects that extend the Content Server type system (Type-based Business Objects and aspects).

**DFS object model**

Your custom DFS service will be more intuitive to use in combination with DFS core services if it makes appropriate use of the DFS object model (see Chapter 2, DFS Data Model) and of design principles embodied in the DFS object model.

For example, a service should always return a DFS DataPackage rather than a specialized object representing a DFC typed object. Services should always be designed so that no DFC client is required on the service consumer.

**Creating inputs to DFS tools**

The following sections describe how to package, implement, and annotate Java classes to provide as input to the DFS SDK tools.

**Service packaging, namespace, and address**

The DFS design tools use service artifact packaging conventions to generate consistent and logical namespace and address for the service. Location transparency (the ability to run the service in remote and local modes) depends on the consistency between the package structure and the service namespace. Table 10, page 179 shows a package naming convention, suggested for when namespace is not specified in the service annotation (in which case the tools generate the namespace based on the package naming convention):
Table 10. Sample service packages

<table>
<thead>
<tr>
<th>Sample service package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>com.acme.services.samples</td>
<td>This package functions as a service module, containing a subset of services for the acme organization.</td>
</tr>
<tr>
<td>com.acme.services.samples.common</td>
<td>This package contains common classes that define data types passed as arguments to the service and/or returned by the service. In the AcmeCustomService directory this package is contained under the src directory and contains JAXB annotated JavaBeans that define data types that will be specified by XML schemas and passed in the SOAP body. The design tools will package binaries generated from these classes so that they will be available to the service and to the remote client.</td>
</tr>
<tr>
<td>com.acme.services.samples.client</td>
<td>This package will contain the public interface for the service. The package and the interface are generated by the DFS design tools. In the AcmeCustomService sample they are placed under the generated\src directory.</td>
</tr>
</tbody>
</table>

If no targetNamespace is specified in the service annotation, the service package name (in the example this is com.acme.services.samples) is used to generate the service namespace. The DFS tools do this by reversing the package name nodes and prepending a ws (to avoid name conflicts between original and JAX-WS generated classes). The following targetNamespace attribute from the sample service WSDL is generated by the tools based on the preceding package names:

targetNamespace="http://ws.samples.services.acme.com/"

The service address generation depends on parameters set in DFS tools to designate two nodes of the package structure as (1) the context root of the service, and (2) as the service module. The following service address is generated for the AcmeCustomService sample where "services" is specified as the context root and "samples" is specified as the service module.

http://127.0.0.1:7001/services/samples/AcmeCustomService?wsdl

When instantiating a service, a Java client application can pass the module name and the fully-qualified context root to ServiceFactory.getRemoteService, as shown here:

```java
mySvc = serviceFactory.getRemoteService(IAcmeCustomService.class, context, "samples", "http://localhost:7001/services");
```

Alternatively, the client can call an overloaded method of getRemoteService that does not include the module and contextRoot parameters. In this case the client runtime obtains the module and contextRoot values from the dfs-client.xml configuration file, which specifies default service addressing values. The dfs-client.xml used by AcmeCustomService is located in resources\config. Its contents are shown here:

```xml
<DfsClientConfig defaultModuleName="samples" registryProviderModuleName="samples">
    <ModuleInfo name="samples" protocol="http" host="127.0.0.1" port="7001" contextRoot="services">
    </ModuleInfo>
</DfsClientConfig>
```
The order of precedence is as follows. The DFS runtime will first use parameters passed in the getRemoteService method. If these are not provided, it will use the values provided in the DfsClientConfig configuration file. If these are not provided, it will use in-code defaults, which are

coreRoot = "http://localhost:7001/services"
module = "core".

**Overriding default targetNamespace and artifact packaging**

As mentioned under Service packaging, namespace, and address, page 178, if targetNamespace is not specified in the service annotation, DFS tools will generate the namespace by inverting the package name and prepending "ws". For example, if the annotated service implementation class AcmeCustomService is in a package com.acme.services.samples, DFS tools will generate DFS service artifacts (such as IAcmeCustomService) in the package com.acme.services.samples.client, and place artifacts created by JAX-WS, including the generated service implementation class, in the package com.acme.services.samples.ws. The targetNamespace of the service would be http://ws.samples.services.acme.com. If this is acceptable, then there is no need to specify a targetNamespace in the service annotation.

To change this behavior, the following procedure is recommended. (Note that this is the approach used in the provided AcmeCustomService sample.)

1. Place the annotated input service class in a package other than the one associated with the targetNamespace, for example com.acme.services.samples.impl.
2. Specify the targetNamespace using the inverse of the package name of the service module.

The annotation and packaging of the service implementation class would look like this:

```java
package com.acme.services.samples.impl;
import com.emc.documentum.fs.rt.annotations.DfsPojoService;

@DfsPojoService(targetNamespace = http://samples.services.acme.com)
public class AcmeCustomService
```

With this input, the DFS tools will generate the service interface and other DFS artifacts as before in the package com.acme.services.samples.client. However, it would place the service implementation and other files generated by JAX-WS in the com.acme.services.samples package; and the service namespace would be "http://samples.services.acme.com" as specified in the service annotation attribute.
The well-behaved service implementation

There are intrinsic differences between an efficient local interaction and an efficient remote interaction. A well-behaved service should be optimized to support an efficient remote interaction, and should exhibits the following characteristics (note that this is not an exhaustive list).

- The service should have an appropriate level of granularity. The most general rule is that the service granularity should be determined by the needs of the service consumer. However, in practice services are generally more coarse-grained than methods in tightly bound client/server applications. They should avoid “chattiness”, be sensitive to round-trip overhead, and anticipate relatively low bandwidth and high latency.

- As mentioned previously, if the service is intended to be used as an extension of DFS services, it should use the DFS object model where possible, and conform to the general design features of the DFS services.

- The service should specify stateless operations that perform a single unambiguous function that the service consumer requires. The operation should stand alone and not be overly dependent on consumer calls to auxiliary services.

- The service should specify parameters and return values that are easily bound to XML, and which are faithfully transformed in interactions between the client and the service.

Not all intrinsic Java types map into identical XML intrinsic types; and not all intrinsic type arrays exhibit are transformed identically to and from XML. Service developers should therefore be aware of the tables Table 11, page 181 and Table 12, page 181 when designing service interfaces.

Table 11. Java intrinsic type to XML mappings

<table>
<thead>
<tr>
<th>Java intrinsic type</th>
<th>Mapped XML type</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
</tr>
<tr>
<td>byte</td>
<td>byte</td>
</tr>
<tr>
<td>char</td>
<td>int</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
</tr>
<tr>
<td>float</td>
<td>float</td>
</tr>
<tr>
<td>int</td>
<td>int</td>
</tr>
<tr>
<td>long</td>
<td>long</td>
</tr>
<tr>
<td>short</td>
<td>short</td>
</tr>
</tbody>
</table>

Table 12. Java intrinsic type to XML mappings for arrays

<table>
<thead>
<tr>
<th>Java array</th>
<th>XML equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean[]</td>
<td>boolean</td>
</tr>
<tr>
<td>byte[]</td>
<td>byte[]</td>
</tr>
<tr>
<td>char[]</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>double[]</td>
<td>List&lt;Double&gt;</td>
</tr>
<tr>
<td>float[]</td>
<td>List&lt;Float&gt;</td>
</tr>
</tbody>
</table>
### Java array

<table>
<thead>
<tr>
<th>Java array</th>
<th>XML equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>int[]</td>
<td>List&lt;Integer&gt;</td>
</tr>
<tr>
<td>long[]</td>
<td>List&lt;Long&gt;</td>
</tr>
<tr>
<td>short[]</td>
<td>List&lt;Short&gt;</td>
</tr>
</tbody>
</table>

### Service annotation

DFS specifies two Java annotations that are used to annotate service classes that service developers provide as input to DFS tools. The annotations, DfsBofService and DfsPojoService, are defined in the package com.emc.documentum.fs.rt.annotations. Insert the annotation immediately above the service class declaration.

```java
import com.emc.documentum.fs.rt.annotations.DfsPojoService;

@DfsPojoService()
public class AcmeCustomService implements IAcmeCustomService {
    // service implementation
}
```

For an SBO, use the @DfsBofService annotation:

```java
import com.emc.documentum.fs.rt.annotations.DfsBofService;

@DfsBofService()
public class MySBO extends DfService implements IMySBO {
    // SBO service implementation
}
```

The annotation attributes, described in the following tables, generally provide overrides to default DFS tools behavior.

#### Table 13. DfsBofService attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceName</td>
<td>The name of the service. Required to be non-empty.</td>
</tr>
<tr>
<td>targetNamespace</td>
<td>Overrides the default Java-package-to-XML-namespace conversion algorithm. Optional.</td>
</tr>
<tr>
<td>requiresAuthentication</td>
<td>When set to &quot;false&quot;, specifies that this is an open service, requiring no user authentication. Default value is &quot;true&quot;.</td>
</tr>
</tbody>
</table>

#### Table 14. DfsPojoService attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>implementation</td>
<td>Name of implementation class. Required to be non-empty if the annotation applies to an interface declaration.</td>
</tr>
</tbody>
</table>
### Attribute Description

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>targetNamespace</td>
<td>Overrides the default Java-package-to-XML-namespace conversion algorithm. Optional.</td>
</tr>
<tr>
<td>targetPackage</td>
<td>Overrides the default Java packaging algorithm. Optional.</td>
</tr>
<tr>
<td>requiresAuthentication</td>
<td>When set to &quot;false&quot;, specifies that this is an open service, requiring no user authentication. Optional; default value is &quot;true&quot;.</td>
</tr>
</tbody>
</table>

**Note:** Although DFS leverages JAX-WS, it does not support JSR-181 annotations. This is due to the difference in emphasis between DFS (service orientation approach) and JAX-WS (web service implementation). DFS promotes an XML-based service model and adapts JAX-WS tools (specifically wsgen and wsimport) to this service model.

## Data type and field annotation

Classes defining data types passed to and returned by services must conform to the definition of a Javabean, and must be annotated with JAXB annotations. The following shows the AcmeServiceInfo class from the AcmeCustomService sample service. Note the specification of the namespace and its correspondence to the package name.

```java
package com.acme.services.samples.common;

import javax.xml.bind.annotation.*;
import java.util.List;

@XmlType(name = "AcmeServiceInfo", namespace = "http://common.samples.services.acme.com/")
@XmlAccessorType(XmlAccessType.FIELD)
public class AcmeServiceInfo {
    public boolean isSessionPoolingActive() {
        return isSessionPoolingActive;
    }

    public void setSessionPoolingActive(boolean sessionPoolingActive) {
        isSessionPoolingActive = sessionPoolingActive;
    }

    public boolean isHasActiveSessions() {
        return hasActiveSessions;
    }

    public void setHasActiveSessions(boolean hasActiveSessions) {
        this.hasActiveSessions = hasActiveSessions;
    }

    public List getRepositories() {
        return repositories;
    }

    public void setRepositories(List repositories) {
        this.repositories = repositories;
    }
```
Best practices for data type naming and annotation

The following recommendations support predictable and satisfactory XML generation of XML data types from Java source, which will in turn support predictable and satisfactory proxy generation from the WSDL using Visual Studio and other tools.

Data type annotation

When annotating data type classes, the following annotations are recommended:

- @XmlType for example:
  ```
  @XmlType(name = "AcmeServiceInfo",
    namespace = "http://common.samples.services.acme.com/")
  ```
  Note that specifying the namespace is mandatory.

- @XmlAccessorType(XmlAccessType.FIELD)

- @XmlEnum (for enumerated types)

- For complex types that have subtypes, use @XmlSeeAlso([subtype_0, subtype_1, ...subtype_n]). For example, the Relationship class has the following annotation:
  ```
  @XmlSeeAlso([ReferenceRelationship.class, ObjectRelationship.class])
  ```

Fields and accessors

When naming fields and accessors, the following conventions are recommended:

- With naming lists and arrays, use plurals; for example:
  ```
  String value
  ```
List<String> values

- As a basic requirement of Javabeans and general Java convention, a field’s accessors (getters and setters) should incorporate the exact field name. This leads to desired consistency between the field name, method names, and the XML element name.

```java
@XmlAttribute
private String defaultSchema;

public String getDefaultSchema()
{
    return defaultSchema;
}

public void setDefaultSchema(String defaultSchema)
{
    this.defaultSchema = defaultSchema;
}
```

- Annotate primitive and simple data types (int, boolean, long, String, Date) using @XmlAttribute.

- Annotate complex data types and lists using @XmlElement, for example:

```java
@XmlElement(name = "Repositories")
private List repositories;

@XmlElement(name = "MyComplexType")
private MyComplexType myComplexTypeInstance;
```

- Fields should work without initialization.
- The default of boolean members should be false.

**Things to avoid**

The following should be avoided when implementing classes that bind to XML types.

- Avoid exposing complex collections as an XML type, other than List<Type>. One-dimensional arrays are also safe.

- Avoid adding significant behaviors to a type, other than convenience methods such as map interfaces and constructors.

- Avoid use of the @XmlElements annotation. This annotation results in an <xsd:choice>, to which inheritance is preferred. Annotate the base class with @XmlSeeAlso instead (see Data type annotation, page 184).
Other pitfalls

The following conditions can also lead to problems either with the WSDL itself, or with .NET WSDL import utilities.

- Use of the @XmlRootElement annotation can cause namespace problems with JAXB 2.1. As a result, the .NET WSDL import utility may complain about "incompatibility of types."

- It is highly recommended that you always use the @XmlAccessorType(XmlAccessType.FIELD) to annotate data type classes. If you use the default value for @XmlAccessorType (which is PROPERTY), the service generation tools will parse all methods beginning with "get" and "set", which makes it difficult to control how the text following "get" and "set" is converted to XML. If one then adds an explicit @XmlElement or @XmlAttribute on a field that already has a getter and setter, the field is likely to be include more than once in the XML schema with slightly different naming conventions.

- Exercise caution using the @XmlContent annotation. Not all types can support it. We recommend using it only for representations of long strings.

Sample service

AcmeCustomService is intended to serve as a minimal example that demonstrates fundamental techniques that you will need to get started developing your own services. It gets a DFC session manager to begin using the DFC API, invokes (chains in) a core DFS service from your custom service (the Schema service), and populates an AcmeCustomInfo object with information obtained from these two sources. For information on how the class for the AcmeCustomInfo object is implemented and annotated, see Data type and field annotation, page 183.

Note that the sample service provides hard-coded values for the address and port of the invoked chained service. You may need to edit these values, providing the address where you have deployed DFS.

```java
ISchemaService schemaService
    = ServiceFactory.getInstance()
        .getRemoteService(ISchemaService.class,
                    context,
                    "core",
                    "http://127.0.0.1:8888/services");
```

The AcmeCustomService source is shown here:

```java
package com.acme.services.samples.impl;

import com.acme.services.samples.common.AcmeServiceInfo;
import com.documentum.fc.client.IDfSessionManager;
import com.documentum.fc.client.IDfSessionManagerStatistics;
import com.emc.documentum.fs.datamodel.core.OperationOptions;
import com.emc.documentum.fs.datamodel.core.schema.SchemaInfo;
import com.emc.documentum.fs.rt.annotations.DfsPojoService;
import com.emc.documentum.fs.rt.context.IServiceContext;
import com.emc.documentum.fs.rt.context.ServiceFactory;
import com.emc.documentum.fs.rt.context.impl.DfcSessionManager;
import com.emc.documentum.fs.services.core.client.ISchemaService;
```
Getting a DFC session manager

To get a session manager from a service, import the DFS runtime class com.emc.documentum.fs.rt.context.DfcSessionManager and instantiate the session manager using the DfcSessionManager.getSessionManager factory method.

IDfSessionManager manager = DfcSessionManager.getSessionManager();

Note: This technique is recommended for service implementations, but not for service consumers. DFS does not require or promote DFC on the consumer, especially on a remote consumer.

Invoking a DFS-provided service

To invoke a DFS service, the sample service first obtains a reference to its own service context:
IServiceContext context = ContextFactory.getInstance().getContext();

Note the use of getContext rather than newContext, which enables the calling service to share identities and any other service context settings with the invoked service.

The service then passes the context, and an explicit service module name and contextRoot, to getRemoteService to invoke the service.

ISchemaService schemaService = ServiceFactory.getInstance()
    .getRemoteService(ISchemaService.class,
    context,
    "core",
    "http://127.0.0.1:7001/services");

Service test consumers

The SDK includes two test consumers of AcmeCustomService, one Java consumer, which can invoke the custom service locally or remotely, and a C# sample consumer that invokes the service remotely.

The AcmeCustomService sample includes an Ant target that compiles and runs a test service consumer. As delivered, the consumer calls the service remotely, but it can be altered to call the service locally by commenting out serviceFactory.getRemoteService and uncommenting serviceFactory.getLocalService.

Note: If you are developing consumers in .NET or using some other non-Java platform, you may still wish to initially test the service using the Java client library, because this will enable you to use local invocation and other conveniences to test your service more quickly. However, it is still advisable to create test consumers on your target consumer platform to confirm that the JAXB markup has generated a WSDL from which your tools generate acceptable proxies.

Example 11-1. Java: Test consumer for AcmeCustomService

package com.acme.services.samples.client;

import com.emc.documentum.fs.datamodel.core.context.RepositoryIdentity;
import com.emc.documentum.fs.rt.ServiceInvocationException;
import com.emc.documentum.fs.rt.context.ContextFactory;
import com.emc.documentum.fs.rt.context.IServiceContext;
import com.emc.documentum.fs.rt.context.ServiceFactory;
import com.emc.documentum.fs.rt.context.ServiceInstantiationException;
import com.acme.services.samples.client.IAcmeCustomService;
import com.acme.services.samples.common.AcmeServiceInfo;

import java.util.ArrayList;
import java.util.Iterator;

public class AcmeCustomServiceDemo
{
    public static void main(String[] args) throws ServiceInvocationException
    {
        ContextFactory contextFactory = ContextFactory.getInstance();
        IServiceContext context = contextFactory.newContext();
        RepositoryIdentity repoId = new RepositoryIdentity();
        repoId.setRepositoryName("YOUR_REPOSITORY_NAME");
        repoId.setUserName("YOUR_USER_NAME");
        repoId.setPassword("YOUR_PASSWORD");
    
}
Building Custom Services Using DFS

context.addIdentity(repoId);
ServiceFactory serviceFactory = ServiceFactory.getInstance();

IAcmeCustomService mySvc;
try {
    mySvc = serviceFactory.getRemoteService(IAcmeCustomService.class,
                                             context,
                                             "samples",
                                             "http://localhost:7001/services);
    // mySvc = serviceFactory.getLocalService(IAcmeCustomService.class, context);
    AcmeServiceInfo acmeServiceInfo = mySvc.getAcmeServiceInfo();
    boolean poolingActive = acmeServiceInfo.isSessionPoolingActive();
    boolean activeSessions = acmeServiceInfo.isHasActiveSessions();
    System.out.println("poolingActive == " + poolingActive);
    System.out.println("activeSession == " + activeSessions);
    ArrayList repositories = (ArrayList)acmeServiceInfo.getRepositories();
    Iterator repositoryIterator = repositories.iterator();
    System.out.println("Repositories:");
    while (repositoryIterator.hasNext())
    {
        System.out.println(repositoryIterator.next());
    }
    String defaultSchema = acmeServiceInfo.getDefaultSchema();
    System.out.println("Default schema:" + defaultSchema);
}
catch (ServiceInstantiationException e)
{
    e.printStackTrace();
}
catch (ServiceInvocationException e)
{
    e.printStackTrace();
}
catch (Throwable t)
{
    t.printStackTrace();
}
}

Example 11-2. C#: Test consumer for AcmeCustomService

using System;
using System.Collections;
using client.Service;
using Emc.Documentum.FS.DataModel.Core.Context;
using Emc.Documentum.FS.Runtime;
using Emc.Documentum.FS.Runtime.Context;

namespace client
{
    class Program
    {
        static void Main(string[] args)
        {
            ContextFactory contextFactory = ContextFactory.Instance;
            IServiceContext context = contextFactory.NewContext();
            RepositoryIdentity repoId = new RepositoryIdentity();
            repoId.RepositoryName = "yourreponame";
            repoId.UserName = "yourusername";
            repoId.Password = "yourpwd";
context.AddIdentity(repoId);
// context = contextFactory.Register(context);
ServiceFactory serviceFactory = ServiceFactory.Instance;

// Remote service invocation
IAcmeCustomService mySvc;
try
{ mySvc =
    serviceFactory
    .GetRemoteService<IAcmeCustomService>(context,
        "samples",
        "http://localhost:7001/services");
AcmeServiceInfo acmeServiceInfo = mySvc.getAcmeServiceInfo();
bool poolingActive = acmeServiceInfo.isSessionPoolingActive;
bool activeSessions = acmeServiceInfo.hasActiveSessions;
Console.WriteLine("poolingActive == " + poolingActive);
Console.WriteLine("activeSession == " + activeSessions);
IList repositories = acmeServiceInfo.Repositories;
Console.WriteLine("Repositories:");
foreach (object repository in repositories)
{ Console.WriteLine(repository);
}

String defaultSchema = acmeServiceInfo.defaultSchema;
Console.WriteLine("Default schema:" + defaultSchema);
} catch (ServiceInvocationException e)
{
    Console.Error.WriteLine(e.StackTrace);
} catch (Exception e)
{
    Console.Error.WriteLine(e.StackTrace);
}
**generateModel task**

The `generateModel` Ant task takes the annotated source code as input to the tools and generates a service model XML file named `|contextRoot|-|serviceName|-service-model.xml`, which describes service artifacts to be generated by subsequent processes. The `generateModel` task is declared as follows:

```xml
<taskdef name="generateModel" classname="com.emc.documentum.fs.tools.GenerateModelTask">
  <classpath location="${dfs.sdk.libs}/emc-dfs-tools.jar"/>
  <classpath location="${dfs.sdk.libs}/emc-dfs-rt.jar"/>
  <classpath location="${dfs.sdk.libs}/emc-dfs-services.jar"/>
  <classpath location="${dfs.sdk.libs}/utilis/aspectjrt.jar"/>
  <classpath location="${dfs.sdk.libs}/jaxws/jaxb-impl.jar"/>
  <classpath location="${dfs.sdk.libs}/jaxws/jaxws-api.jar"/>
  <classpath location="${dfs.sdk.libs}/jaxws/jaxws-api.jar"/>
  <classpath location="${dfs.sdk.libs}/commons/commons-lang-2.1.jar"/>
  <classpath location="${dfs.sdk.libs}/utils/log4j.jar"/>
</taskdef>
```

The `generateModel` task takes the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contextRoot</td>
<td>Attribute representing the root of the service address. For example, in the URL <a href="http://127.0.0.1:7001/services/">http://127.0.0.1:7001/services/</a> &quot;services&quot; signifies the context root.</td>
</tr>
<tr>
<td>moduleName</td>
<td>Attribute representing the name of the service module.</td>
</tr>
<tr>
<td>destDir</td>
<td>Attribute representing a path to a destination directory into which to place the output service-model XML.</td>
</tr>
<tr>
<td>&lt;services&gt;</td>
<td>An element that provides a list (a &lt;fileset&gt;), specifying the annotated source artifacts.</td>
</tr>
<tr>
<td>&lt;classpath&gt;</td>
<td>An element providing paths to binary dependencies.</td>
</tr>
</tbody>
</table>

In the sample service build.xml, the `generateModel` task is configured and as follows:

```xml
<generateModel contextRoot="${context.root}"
  moduleName="${module.name}"
  destdir="${project.artifacts.folder}/src">
  <services>
    <fileset dir="${src.dir}"
      include name="**/*.java"/>
  </fileset>
</generateModel>
```

**generateArtifacts task**

The `generateArtifacts` Ant task takes the source modules and service model XML as input, and creates all output source artifacts required to build and package the service. These include the service interface and implementation classes, data and exception classes, runtime support classes, and service WSDL with associated XSDs. The `generateArtifacts` task is declared as follows:

```xml
<taskdef name="generateArtifacts"...>
The generateArtifacts task takes the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceModel</td>
<td>Attribute representing a path to the service model XML created by the generateModel task.</td>
</tr>
<tr>
<td>destDir</td>
<td>Attribute representing the folder into which to place the output source code. Client code is by convention placed in a &quot;client&quot; subdirectory, and server code in a &quot;ws&quot; subdirectory.</td>
</tr>
<tr>
<td>&lt;src&gt;</td>
<td>Element containing location attribute representing the location of the annotated source code.</td>
</tr>
<tr>
<td>&lt;classpath&gt;</td>
<td>An element providing paths to binary dependencies.</td>
</tr>
</tbody>
</table>

In the sample service build.xml, the generateArtifacts task is configured and executed as follows:

```xml
classname="com.emc.documentum.fs.tools.build.ant.GenerateArtifactsTask">
<classpath location="${dfs.sdk.home}/lib/commons/commons-io-1.2.jar"/>
<classpath location="${dfs.sdk.home}/lib/emc-dfs-rt.jar"/>
<classpath location="${dfs.sdk.home}/lib/emc-dfs-tools.jar"/>
<classpath location="${dfs.sdk.home}/lib/emc-dfs-services.jar"/>
<classpath location="${dfs.sdk.home}/lib/dfc/aspectjrt.jar"/>
</taskdef>

The generateArtifacts task takes the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceModel</td>
<td>Attribute representing a path to the service model XML created by the generateModel task.</td>
</tr>
<tr>
<td>destDir</td>
<td>Attribute representing the folder into which to place the output source code. Client code is by convention placed in a &quot;client&quot; subdirectory, and server code in a &quot;ws&quot; subdirectory.</td>
</tr>
<tr>
<td>&lt;src&gt;</td>
<td>Element containing location attribute representing the location of the annotated source code.</td>
</tr>
<tr>
<td>&lt;classpath&gt;</td>
<td>An element providing paths to binary dependencies.</td>
</tr>
</tbody>
</table>

In the sample service build.xml, the generateArtifacts task is configured and executed as follows:

```xml
<generateArtifacts
    serviceModel="${project.artifacts.folder}/src/${context.root}-${module.name}-service-model.xml"
    destdir="${project.artifacts.folder}/src">
    <src location="${src.dir}"/>
    <classpath>
        <path location="${basedir}/${build.folder}/classes"/>
        <path location="${dfs.sdk.home}/lib/emc-dfs-rt.jar"/>
        <path location="${dfs.sdk.home}/lib/emc-dfs-services.jar"/>
        <pathelement location="${dfs.sdk.home}/lib/dfc/dfc.jar"/>
        <fileset dir="${dfs.sdk.home}/lib/ucf">
            <include name="**/*.jar"/>
        </fileset>
    </classpath>
</generateArtifacts>
```

### buildService task

The buildService tasks takes the original annotated source, as well as output from the buildArtifacts task, and builds two JAR files:

- A remote client package: {moduleName}-remote.jar
- A server (and local client) package: {moduleName}.jar

The buildService task is declared as follows:

```xml
<taskdef name="buildService" classname="com.emc.documentum.fs.tools.build.ant.BuildServiceTask">
    <classpath location="${dfs.sdk.home}/lib/emc-dfs-tools.jar"/>
    <classpath location="${dfs.sdk.home}/lib/emc-dfs-rt.jar"/>
    <classpath location="${dfs.sdk.home}/lib/emc-dfs-services.jar"/>
</taskdef>
```
The buildService task takes the following arguments.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>serviceName</td>
<td>Attribute representing the name of the service module.</td>
</tr>
<tr>
<td>destDir</td>
<td>Attribute representing the folder into which to place the output JAR files.</td>
</tr>
<tr>
<td>&lt;scr&gt;</td>
<td>Element containing location attribute representing the locations of the input source code, including the original annotated source and the source output by generateArtifacts.</td>
</tr>
<tr>
<td>&lt;classpath&gt;</td>
<td>Element providing paths to binary dependencies.</td>
</tr>
</tbody>
</table>

In the sample service build.xml, the buildService task is configured as follows:

```xml
<buildService
    serviceName="${service.name}"
    destDir="${basedir}/${build.folder}"
    generatedArtifactsDir="${project.resources.folder}"
>
  <src>
    <path location="${src.dir}"/>
    <path location="${project.artifacts.folder}/src"/>
  </src>

  <classpath>
    <pathelement location="${dfs.sdk.home}/lib/dfc/dfc.jar"/>
  </classpath>
</buildService>
```

### packageService task

The packageService packages all service artifacts into an EAR file that is deployable to the application server. The packageService task is declared as follows:

```xml
<taskdef name="packageService"
    classname="com.emc.documentum.fs.tools.build.ant.PackageServiceTask">
  <classpath location="${dfs.sdk.home}/lib/emc-dfs-tools.jar"/>
  <classpath location="${dfs.sdk.home}/lib/emc-dfs-rt.jar"/>
  <classpath location="${dfs.sdk.home}/lib/emc-dfs-services.jar"/>
  <classpath location="${dfs.sdk.home}/lib/jaxws/jaxb-api.jar"/>
  <classpath location="${dfs.sdk.home}/lib/jaxws/jaxb-impl.jar"/>
  <classpath location="${dfs.sdk.home}/lib/jaxws/jsr173_api.jar"/>
  <classpath location="${dfs.sdk.home}/lib/rtdfc.jar"/>
  <classpath location="${dfs.sdk.home}/lib/dfc/aspectjrt.jar"/>
</taskdef>
```

The packageService task takes the following arguments:

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deployment-Name</td>
<td>Attribute representing the name of the service module.</td>
</tr>
<tr>
<td>destDir</td>
<td>Attribute representing the folder into which to place the output archives.</td>
</tr>
</tbody>
</table>
Generating C# proxies

To generate C# proxies for the custom service, use the DfsProxyGen.exe utility supplied in the DFS SDK. DfsProxyGen is a Windows form application that generates C# proxies based on a DFS service WSDL and the generateArtifacts ant task (see generateArtifacts task, page 191). You will need to build and deploy the service before creating the C# proxies.

**Figure 19. DfsProxyGen form**

To generate C# proxies:

---

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generatedArtifactsFolder</td>
<td>Path to folder in which WSDL and associated files have been generated.</td>
</tr>
<tr>
<td>&lt;libraries&gt;</td>
<td>Element specifying paths to binary dependencies.</td>
</tr>
<tr>
<td>&lt;resources&gt;</td>
<td>Element providing paths to resource files.</td>
</tr>
</tbody>
</table>
1. In the **Shared assemblies** field, add any shared assemblies used by the service. (There are none for AcmeCustomService.) For more information on this see *Creating shared assemblies for data objects shared by multiple services*, page 195.

2. In the **Service model file** field, browse to the service model file created by the generateArtifacts ant task. For AcmeCustomService this will be emc-dfs-sdk-6.0\samples\AcmeCustomService\resources\services-samples-service-model.xml.

3. In the **Wsdl uri** field, supply the name of the WSDL of the deployed service, for example http://localhost:7001/services/samples/AcmeCustomService?wsdl. Only URLs are permitted, not local file paths, so you should use the URL of the WSDL where the service is deployed.

4. In the Output namespace, supply a namespace for the C# proxy (for example samples.services.acme).

5. Optionally supply a value in the **Output FileName** field. If you don't supply a name, the proxy file name will be the same as the name of the service, for example AcmeCustomService.cs.

6. Click **Create proxy**.

The results of the proxy generation will appear in the **Log** field. If the process is successful, the name and location of the result file will be displayed.

---

**Creating shared assemblies for data objects shared by multiple services**

If you are creating multiple services that share data objects, you will want to generate C# proxies for the shared classes only once and place them in a shared assembly. The following procedure describes how to do this, based on the following scenario: you have created two services ServiceA and ServiceB; the two services share two data object classes, DataClass1 and DataClass2.

1. Run DfsProxyGen against the WSDL and service model file for ServiceA.
   - This will generate the proxy source code for the service and its data classes DataClass1 and DataClass2.

2. Create a project and namespace for the shared classes, DataClass1 and DatasClass2, that will be used to build the shared assembly. Cut DataClass1 and DataClass2 from the generated proxies source generated for ServiceA, and add them to new source code file(s) in the new project.

3. Annotate the shared data classes using XmlSerializer’s `[XmlType()]` attribute, specifying the WSDL namespace of the shared classes (for example `XmlType(Namespace=http://myservices/datamodel/)`).

4. Build an assembly from the shared datamodel project.

5. Run DfsProxyGen against the WSDL and service model for ServiceB, referencing the shared assembly created in step 4 in the **Shared assemblies** field.
Exploring AcmeCustomService

The AcmeCustomService sample is a demo build environment that utilizes the DFS Ant tasks in a build.xml file to generate service artifacts and deployable service archive files from input Java source files and configuration files. This section will provide a brief tour of the AcmeCustomService sample, and show you how to generate, deploy, and test the AcmeCustomService service.

Service test consumers

The service test consumers (see Service test consumers, page 188) is located under AcmeCustomService\src\client-remote and client-remote.net. To prepare the consumers for running, you will need to replace the hard-coded values for repository name, user name, and password with values required by your test environment.

Example 11-3. Java: Service test consumer hardcoded values

```java
// replace these values
repoId.setRepositoryName("YOUR_REPOSITORY_NAME");
repoId.setUserName("YOUR_USER_NAME");
repoId.setPassword("YOUR_PASSWORD");
```

Example 11-4. C#: Service test consumer hardcoded values

```csharp
repoId.RepositoryName = "yourreponame";
repoId.UserName = "yourusername";
repoId.Password = "yourpwd";
```

The call to getRemoteService assumes that the instance of WebLogic that you are deploying to is running on the local host on port 7001. You must change this value if you are deploying to an instance of WebLogic running at another address and/or at another port.

Example 11-5. Java: Sample service invocation

```java
// modify if your app server is running somewhere else
mySvc = serviceFactory.getRemoteService(IAcmeCustomService.class,
context,
"samples",
"http://localhost:7001/services");
```

Example 11-6. C#: Sample service invocation

```csharp
mySvc = serviceFactory.GetRemoteService<IAcmeCustomService>(context,
"samples",
"http://localhost:7001/services");
```

build.properties

The build.properties file under AcmeCustomService contains property settings required by the Ant build.xml file. To generate and deploy AcmeCustomService there is no need to change any of these
settings, unless you have moved the AcmeCustomService directory to another location relative to the root of the SDK. In this case you will need to change the dfs.sdk.home property.

```bash
# EMC DFS SDK 6.0 build properties template
dfs.sdk.home=../..  
# Compiler options
compiler.debug=on
compiler.generate.no.warnings=off
compiler.args=
fork = true
nonjava.pattern = **/*.java,/**.svn,/**_svn
# Establish the production and tests build folders
build.folder = build
module.name = samples
context.root = services

#Debug information
debug=true
keep=true
verbose=false
extension=true
```

**autodeploy.properties**

The autodeploy.properties file configures properties that are used by build.xml to deploy the service EAR file to a directory on a WebLogic server domain. You will need to modify this file to match your WebLogic installation if you are going to use the Deploy ant target. This target is only useful if you have WebLogic in Developer mode and you are deploying to the Autodeploy directory.

```bash
# Deploy params
autodeploy.dir=C:/bea/user_projects/domains/WS/autodeploy

# Deployment information
server.ip=127.0.0.1
server.protocol=http
server.port=7001
```

**dfs-client.xml**

The dfs-client.xml file contains properties used by the Java client runtime for service addressing. The AcmeCustomService test consumer provides the service address explicitly when instantiating the service object, and so does not use these defaults. However, it's important to know that these defaults are available and where to set them.

```xml
<DfsClientConfig defaultModuleName="samples"
    registryProviderModuleName="samples">
    <ModuleInfo name="samples"
        protocol="http"
        host="127.0.0.1"
        port="7001"
        contextRoot="services">
    </ModuleInfo>
</DfsClientConfig>

**Note:** If dfs-client.xml is missing, the client runtime will look for it at a higher level of the SDK folder structure.
Building Custom Services Using DFS

Note: .NET consumers use app.config instead of dfs-client.xml, as application configuration infrastructure is built into .NET itself. See .NET client configuration, page 22.

**dfc.properties**

The service-generation tools package a copy of dfc.properties within the service EAR file. The properties defined in this dfc.properties file configure the DFC client utilized by the DFS service runtime. The copy of dfc.properties is obtained from the DFS SDK etc directory. The dfc.properties must specify the address of a docbroker that can provide access to any repositories required by the service and its clients, for example:

```
dfc.docbroker.host[0]=10.8.13.190
```

The docbroker can be specified by IP address or by computer name.

**build.xml**

The Ant build.xml file drives all stages of generating and deploying the custom service. It contains the targets shown in Table 15, page 198, which can be run in order to generate and deploy the custom service.

**Table 15. Sample service build.xml Ant targets**

<table>
<thead>
<tr>
<th>Ant target</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>clean</td>
<td>Deletes the build directory in which the service binaries are generated.</td>
</tr>
<tr>
<td>artifacts</td>
<td>Executes the generateModel task (see generateModel task, page 191) to create the service definition; executes the generateArtifacts task (see generateArtifacts task, page 191) to generate the service class files, WSDL, and XML schemas.</td>
</tr>
<tr>
<td>package</td>
<td>Executes the buildService task (see buildService task, page 192) to build the service jar files for remote and local invocation; executes the packageService task (see packageService task, page 193) to build the service EAR file for deployment to the application server.</td>
</tr>
<tr>
<td>deploy</td>
<td>Copies the EAR file generated by the packageService task to the WebLogic autodeploy directory defined as a directory path in the autodeploy.properties file. This is useful only if your WebLogic domain is set up in developer mode.</td>
</tr>
<tr>
<td>run</td>
<td>Compiles and runs the service test consumer.</td>
</tr>
</tbody>
</table>

**Running the tools**

After making any required modifications to the configuration files described in the preceding sections, run the Ant targets as follows:

```
C:\emc-dfs-sdk-6.0\quick-start\AcmeCustomService>ant clean artifacts package
```
You may prefer to run the targets individually and examine the output of each step. After running the package target, use the WebLogic Server Administration Console to deploy your service.

Once the service is deployed, you can test it by compiling and running the test consumer. The build.xml run target does this:

C:\emc-dfs-sdk-6.0\quick-start\AcmeCustomService>ant run
Appendix A

Guidelines for Migrating from Web Services Framework to DFS

This appendix presents some general guidelines for migrating SBOs projected as web services using the EMC Documentum Web Services Framework to Enterprise Content Services that work within the DFS framework. This appendix discusses the following topics:

- WSF and DFS, page 201
- Candidates for direct conversion, page 201
- DFS facade, page 202
- Building SBO services, page 202
- Security model and service context, page 202
- Content transfer, page 203

WSF and DFS

DFS is not an evolution of the Web Services Framework (WSF), but an entirely new set of products that completely replaces WSF. While WSF is a relatively limited set of technologies designed around the goal of projecting SBOs as web services, DFS is a multifaceted technology set used for developing and deploying ECM services and consumers using a service-oriented architecture. For this reason migration from WSF to DFS is not a matter of converting a WSF service to a DFS service, but rather of taking the source code of the SBO used to generate the WSF service, refactoring the code if necessary to make use of the DFS data model and runtime support, and then using the source code to generate an Enterprise Content Service using DFS tools. In fact, migrating a service from WSF to DFS is in no way different from leveraging any existing SBO as a DFS service, whether or not the SBO has been projected as a WSF service.

Candidates for direct conversion

An SBO designed as an Enterprise Content Service would use primitive types and DFS data model objects as its parameter and return types. It would also make use of DFS runtime services and classes to manage UCF content transfer. And by virtue of its having been built using DFS tools, it would run in a DFS service context, and therefore make use of the DFS WS-Security compliant security
model. Obviously a preexisting SBO would not have any of these features, so a suitable candidate would be one that lacks features that would impede the implementation of DFS features. A good candidate SBO might:

• Use primitives as parameters and return values, or
• Use simple Javabeans as parameters and return values
• Avoid UCF content transfer. Good candidates will not transfer content, or use base64

For other criteria to consider when designing a DFS service implementation, see The well-behaved service implementation, page 181

DFS facade

If an SBO is not suitable for direct conversion to a DFS service, an effective strategy to leverage the SBO code is to build a DFS service as a facade to the SBO. The facade would delegate to the SBO while handling conversion between DFS data model types and types expected and returned by the SBO. The facade could also provide behaviors common to DFS services, such as awareness of profile settings in the service context.

This is an effective strategy for preserving working SBO service code with minimal risk, as well as avoiding modification to DFC clients that currently use the SBO.

Building SBO services

SBO services, like POJO services, are generated using tools provided with the DFS SDK. For procedures and guidelines for building services, refer to Chapter 11, Building Custom Services Using DFS.

Security model and service context

Services generated using DFS tools use a uniform service context and security model. Consumers of DFS services must create a service context using the ContextRegistry runtime service. Java clients do not have to invoke this runtime service directly, but instead use convenience methods that invoke the ContextRegistry service behind the scenes.

This means that if you convert a WSF service to DFS, any client code that calls the WSF service will need to be modified to use the DFS security model.

For more information refer to Service Context, page 56.
Content transfer

DFS provides sophisticated support for content transfer, including support for the base64 and MTOM standards, as well as for UCF transfer. UCF transfer support involves a number of specialized types (such as ActivityInfo and UcfContent), as well as use of the Agent runtime service. DFS support for UCF transfer enables minimization of hops in end-to-end transfers in complex multi-tier application, and use of ACS and BOCS to minimize the distance of the content transfer by accessing a content repository close to the user. The UCF transfer mechanism provided with WSF is considerably simpler, and it is not supported in DFS. If you are migrating a WSF service that makes use of UCF content transfer, both the service and any service clients will need to be refactored to use the more powerful DFS UCF support.

For more information on content transfer see Chapter 10, Content and Content Transfer.
A
Accelerated Content Services, 157
AcmeCustomService, 196
address
  service, 178
annotation
  best practices, 184
data type, 183
fields, 184
service, 182
Ant, 190
Ant targets, 198
Ant tasks
  buildService, 192
generateArtifacts, 191
generateModel, 191
packageService, 193
ArrayProperty, 37
attribute. See property autodeploy.properties, 197

B
base64, 158, 162
BasicIdentity, 57
BOCS. See Branch Office Caching Services
Branch Office Caching Services, 157, 162
build.properties, 196
build.xml, 198
buildService task, 192

cached query processing, 127
CacheStrategyType, 124
cancelCheckout operation, 99
chaining services, 58, 187
checkin operation, 95
  CheckinProfile, 97
  VersionStrategy, 96
checkout operation, 93

client library, .NET, 22
client library, Java, 20
compound DataObject, 49
  with references, 50
compound permissions, 43
Condition, 136
consumer, dfs, 20
content, 40, 157
Content model, 161
content transfer, 157
  optimization, 157
ContentProfile, 40
  with Object service get operation, 71
ContentTransferMode, 158
  base64, 158
  MTOM, 159
  UCF, 161
ContentTransferProfile, 162
copy operation, 80
  across repositories, 82
  with modifications, 83
CopyProfile, 81
create operation, 61
createPath operation, 66
custom service, 177

data graph, 46
data model, DFS, 27
data type
  annotating, 183 to 184
database search, 132
DataObject, 28
  as data graph, 46
  compound, 49
  compound with references, 50
  standalone, 47
  with references, 48
DataPackage, 27
date and time, 133
delete operation, 78
deleteAllVersions operation, 101
DeleteProfile, 79
deleteVersion operation, 100
DepthFilter, 52
design-time tools, 25
DFC. See Documentum Foundation Classes
dfc:properties, 198
DFS. See Documentum Foundation Services
dfs-client.xml, 197
@DfsBofService, 182
@DfsPojoService, 182
Document Query Language, 30 to 31, 40,
   124, 131
   hints file, 132
Documentum Foundation Classes, 187
   configuration, 198
Documentum Foundation Services, 15
DQL. See Document Query Language

E
ECS. See Enterprise Content Services
editor, opening document in, 41, 176
Enterprise Content Services, 24, 55
   common features, 56
execute operation, of Query service, 125
execute operation, of Search service, 141
Expression model, 134
   FullTextExpression, 135
   PropertyExpression, 135
ExpressionSet, 134
ExpressionValue, 136

F
full-text search, 131 to 132
FullTextExpression, 135

G
generateArtifacts task, 191
generateModel task, 191
geoLocation, 162
get operation, 67
getCheckoutInfo operation, 91
gGetCurrent operation, 102
getDynamicAssistValues operation, 118
gGetObjectContentUrls operation, 88
getProcessInfo operation, 150
getProcessTemplates operation, 148
getPropertyInfo operation, 117
getRepositoryList operation, 139
getSchemaInfo operation, 111, 113
getTypeInfo operation, 115
getVersionInfo operation, 104

H
hierarchical permissions, 43

I
identity, 28, 57
   object, 32

J
Java client library, 20
Javabeans, 183
JAXB
   annotations, 183 to 184

L
location transparency, 20, 178

M
mode, content transfer. See
   ContentTransferMode
move operation, 84
MoveProfile, 85
MTOM, 162
MTOM content transfer mode, 159

N
namespace
   overriding default, 180
   secondary, 178

O
Object service, 61
   copy operation, 80
   create operation, 61
   createPath operation, 66
delete operation, 78
get operation, 67
gGetObjectContentUrls operation, 88
move operation, 84
update operation, 73
validate operation, 87
ObjectId, 30
ObjectIdentity, 30
ObjectIdentitySet, 32
ObjectPath, 31
ObjectRelationship, 44
removing, 50
returning DataObject as, 51
updating, 77
OperationOptions, 60
packages
DFS public, 18
of custom service, 178
packageService task, 193
PassthroughQuery, 124
in Search service, 132
PermissionProfile, 43
permissions, 42
compound, 43
Plain Old Java Object, 16, 177
POJO. See Plain Old Java Object
PostTransferAction, 41, 176
profile
CopyProfile, 81
DeleteProfile, 79
MoveProfile, 85
passing in OperationOptions, 60
property, 39
relationship, 51
with create operation, 62
with Object service get operation, 69
with Object service update operation, 73
property, 33
array, 37
delete repeating, 38
loading, 35
model, 33
of service context, 58
profile, 39
repeating, 37, 75
transient, 34
PropertyExpression, 135
PropertyInfo, 108
PropertyProfile, 39
with Object service get operation, 69
PropertySet, 38
Q
Qualification, 31
Query model, 123
PassthroughQuery, 124
Query service, 123
cached query processing, 127
CacheStrategyType, 124
execute operation, 125
QueryExecution, 123
QueryResult, 137
QueryStatus, 137
R
ReferenceRelationship, 44
returning DataObject as, 51
relationship, 44
filters, 52
object, 44, 51
reference, 44, 51
removing, 45, 50
TargetRole, 46
updating ObjectRelationship, 77
RelationshipInfo, 110
RelationshipIntentModifier, 45
RelationshipProfile, 51
DepthFilter, 52
with Object service get operation, 70
repeating property, 37, 75
deleting, 38
RepositoryIdentity, 57
RepositoryScope, 134
RepositoryStatus, 138
RepositoryStatusInfo, 138
resultDataMode, 51
S
SBO. See Service-based Business Object
Schema service, 107
getDynamicAssistValues operation, 118
getPropertyInfo operation, 117
getSchemaInfo operation, 111, 113
getTypeInfo operation, 115
PropertyInfo, 108
RelationshipInfo, 110
SchemaProfile, 110
TypeInfo, 107
ValueInfo, 109
SchemaProfile, 110
SDK. See Software Development Kit
search
full-text and database, 132
Search service, 131
Condition, 136
execute operation, 141
ExpressionSet, 134
ExpressionValue, 136
getRepositoryList operation, 139
PassThroughQuery, 132
QueryResult, 137
QueryStatus, 137
RepositoryScope, 134
RepositoryStatus, 138
RepositoryStatusInfo, 138
StructuredQuery, 132
service
address, 178
annotating, 182
best practices, 181
chaining, 58, 187
custom, 177
generation tools, 25, 190
generation tools, configuring, 196
generation tools, running, 198
getting DFC session manager in, 187
instantiating, 59
namespace, 180
Object. See Object service
package, 180
packaging, namespace, and
address, 178
POJO and SBO, 177
sample, 186, 196
test consumer, 188
service context, 56
getting current, 58
identity, 57
properties, 58
registering, 58
token, 58
Service-based Business Object, 16, 177
service-oriented architecture, 16
SOA, 16
SOAP, 16, 20, 59, 158, 179
Software Development Kit, 17
standalone DataObject, 47
startProcess operation, 151
StructuredQuery, 132
T
targetNamespace, 180
targtNameSpace, 178
TargetRole, 46
time and date, 133
token
service context, 58
transient property, 34
TypeInfo, 107
U
Unified Client Facilities, 20, 40, 97, 157, 161
to 162
update operation, 73
V
validate operation, 87
ValidationInfo, 87
ValidationInfoSet, 87
ValueAction, 37
ValueInfo, 109
VersionControl service, 91
cancelCheckout operation, 99
checkin operation, 95
checkout operation, 93
deleteAllVersions operation, 101
deleteVersion operation, 100
getCheckoutInfo operation, 91
current operation, 102
current operation, 102
current operation, 104
VersionStrategy, 96
viewer, opening document in, 41, 176
W
Web Services Framework, 16, 201
Windows Communication Foundation
(WCF), 22
Workflow service, 147
getProcessInfo operation, 150
getProcessTemplates operation, 148
startProcess operation, 151
WSF. See Web Services Framework
X

XML
  data types, 181
  @XmlAccessType, 186
  @XmlContent, 186
  @XmlRootElement, 186