FATWIRE CORPORATION PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. In no event shall FatWire be liable for any loss of profits, loss of business, loss of use of data, interruption of business, or for indirect, special, incidental, or consequential damages of any kind, even if FatWire has been advised of the possibility of such damages arising from this publication. FatWire may revise this publication from time to time without notice. Some states or jurisdictions do not allow disclaimer of express or implied warranties in certain transactions; therefore, this statement may not apply to you.

Copyright © 2010 FatWire Corporation. All rights reserved.

This product may be covered under one or more of the following U.S. patents: 4477698, 4540855, 4720853, 4742538, 4742539, 4782510, 4797911, 4894857, 5070525, RE36416, 5309505, 5511112, 5581602, 5594791, 5675637, 5708780, 5715314, 5724424, 5812776, 5828731, 5909492, 5924090, 5963635, 6012071, 6049785, 6055522, 6118763, 6195649, 6199051, 6205437, 6212634, 6279112 and 6314089. Additional patents pending.

FatWire, Content Server, Content Integration Platform, Content Server Bridge Enterprise, Content Server Bridge XML, Content Server COM Interfaces, Content Server Desktop, Content Server Direct, Content Server Direct Advantage, Content Server DocLink, Content Server Engage, Content Server InSite Editor, Content Server Satellite, and Transact are trademarks or registered trademarks of FatWire Corporation in the United States and other countries.

Java, J2EE, Solaris, Sun, and other Sun products referenced herein are trademarks or registered trademarks of Sun Microsystems, Inc. AIX, IBM, WebSphere, and other IBM products referenced herein are trademarks or registered trademarks of IBM Corporation. WebLogic is a registered trademark of BEA Systems, Inc. Documentum is a registered trademark of the EMC Corporation. Microsoft, Windows, SharePoint, Microsoft Visual C++ 2008 Redistributable Package and other Microsoft products referenced herein are trademarks or registered trademarks of Microsoft Corporation. Linux is a trademark registered to Linus Torvalds. UNIX is a registered trademark of The Open Group. Any other trademarks and product names used herein may be the trademarks of their respective owners.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/) and software developed by Sun Microsystems, Inc. This product contains encryption technology from Phaos Technology Corporation.

You may not download or otherwise export or reexport this Program, its Documentation, or any underlying information or technology except in full compliance with all United States and other applicable laws and regulations, including without limitations the United States Export Administration Act, the Trading with the Enemy Act, the International Emergency Economic Powers Act and any regulations thereunder. Any transfer of technical data outside the United States by any means, including the Internet, is an export control requirement under U.S. law. In particular, but without limitation, none of the Program, its Documentation, or underlying information of technology may be downloaded or otherwise exported or reexported (i) into (or to a national or resident, wherever located, of) Cuba, Libya, North Korea, Iran, Iraq, Sudan, Syria, or any other country to which the U.S. prohibits exports of goods or technical data; or (ii) to anyone on the U.S. Treasury Department’s Specially Designated Nationals List or the Table of Denial Orders issued by the Department of Commerce. By downloading or using the Program or its Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not located in, under the control of, or a national or resident of any such country or on any such list or table. In addition, if the Program or Documentation is identified as Domestic Only or Not-for-Export (for example, on the box, media, in the installation process, during the download process, or in the Documentation), then except for export to Canada for use in Canada by Canadian citizens, the Program, Documentation, and any underlying information or technology may not be exported outside the United States or to any foreign entity or “foreign person” as defined by U.S. Government regulations, including without limitation, anyone who is not a citizen, national, or lawful permanent resident of the United States. By using this Program and Documentation, you are agreeing to the foregoing and you are representing and warranting that you are not a “foreign person” or under the control of a “foreign person.”

FATWIRE Content Integration Platform Administrator’s Guide
Publication Date: Jul. 13, 2010
Product Version 1.5

FatWire Technical Support
www.fatwire.com/Support

FatWire Headquarters
FatWire Corporation
330 Old Country Road
Suite 207
Mineola, NY 11501
www.fatwire.com
Table of Contents

1 Integrating with FatWire Content Server ................................. 7
   Overview ........................................................................... 8
   System Defaults.................................................................. 10
      Default Mapping Framework .......................................... 10
      Implications for Publishing and Synchronization ............... 12
   Event Notification Option .................................................. 12

Part 1. Installing and Publishing

2 Installing FatWire Content Integration Platform ....................... 15
   Installation Overview ......................................................... 16
   Prerequisites ..................................................................... 16
   Packaging ......................................................................... 16
   Where to Install the Files .................................................. 16
   Installing FatWire Content Integration Platform .................... 17
      Step I. Installing Content Integration Agent ....................... 17
      Step II. Installing Content Server Agent Services ............... 20
      Step III. Installing Schema on Content Server ................... 21
      Step IV. Optional. Configuring Event Notification ............... 22
   Verifying the Installation .................................................... 22
   Publishing Production Data ................................................ 23
   Customizing Your Installation ........................................... 23

3 Publishing ......................................................................... 25
   Overview ........................................................................... 26
   Publishing from TeamUp ..................................................... 26
   Synchronization ................................................................. 26
   Unpublishing ..................................................................... 26
Publishing to Content Server .................................................. 26
Examples ............................................................................. 29
When an Object is Published .................................................. 31
Synchronization .................................................................... 31
Unpublishing Command ......................................................... 32

Part 2. Customizing CIP Installations

4 Configuring Event Notification .......................................... 35
   Overview ................................................................. 36
   Custom Workflows ..................................................... 36
   Installing Sample Workflows ........................................ 37

5 Remapping ....................................................................... 41
   When to Remap ........................................................ 42
   Remapping Procedures ............................................... 43
      File Systems ......................................................... 43
      Documentum Systems ......................................... 46
      SharePoint Systems ........................................... 50
      TeamUp Systems ................................................ 54

6 Adding Assets to a Flex Family ....................................... 57
   Adding a New Attribute .............................................. 58
   Adding a New Child Definition Asset ............................ 61

Part 3. System Specifications

A. File Systems: Default Mapping Specifications ................. 65
   Overview ................................................................. 66
   Default mappings.xml ............................................... 69
   ‘FileSystem’ Flex Family Specifications ....................... 70

B. Documentum Systems: Default Mapping Specifications .... 71
   Overview ................................................................. 72
   Default mappings.xml ............................................... 75
   ‘Documentum’ Flex Family Specifications .................... 76

C. SharePoint Systems: Default Mapping Specifications ........ 77
   Overview ................................................................. 78
   Default mappings.xml ............................................... 81
   ‘SharePoint’ Flex Family Specifications ....................... 82
D. TeamUp Systems: Default Mapping Specifications .......................... 83
   Overview .................................................................................. 84
   Default mappings.xml .............................................................. 87
   ‘TeamUp’ Flex Family Specifications ...................................... 88
Chapter 1

Integrating with FatWire Content Server

FatWire Content Integration Platform is a comprehensive solution for publishing objects to FatWire Content Server from various source systems: file systems, EMC Documentum, Microsoft SharePoint, and FatWire TeamUp.

This chapter begins with an overview of the FatWire Content Integration Platform. The middle section provides information about the default mapping framework, which supports default publishing scenarios. The last section outlines conditions that require you to customize mappings.

This chapter contains the following sections:

- **Overview**
- **System Defaults**
- **Event Notification Option**

**Note**

This guide is for administrators with experience installing and configuring enterprise-level software. An administrator’s knowledge of both the source system and Content Server is required, including a detailed understanding of Content Server’s flex asset model, the process of creating and configuring content management sites, and procedures for enabling asset types.
Overview

Publishing objects to FatWire Content Server from an external source is accomplished with the two components of the FatWire Content Integration Platform: Content Integration Agent and Content Server Agent Services, both shown in Figure 1, “System Architecture.”

- **Content Integration Agent** contains CIPCommander, used to initiate publishing sessions from the command line, where you specify which folder to publish. The contents of the folder are also published. They include subfolders and documents (and pictures in SharePoint systems).

Content Integration Agent, itself, is a process (either daemon or standalone) responsible for synchronizing the metadata of objects that are selected for publication (via CIPCommander). The metadata mapping is defined in mappings.xml, within Content Integration Agent. Using mappings.xml, the Content Integration Agent process extracts the metadata from the source system, converts the metadata to a format recognizable by Content Server, and invokes the synchronization engine to publish the CS-compliant metadata to Agent Services.

Initially, when CIPCommander is run, the synchronization engine publishes the metadata. Then, every time a new object is created, deleted, or modified in the monitored folder, the synchronization engine updates Content Server Agent Services with the new metadata. (The synchronization interval is configurable.)

- **Content Server Agent Services** is a web application responsible for receiving CS-compliant metadata from Content Integration Agent and storing it in Content Server’s database. Agent Services also runs a background process that extracts and
stores the primary binary contents associated with the metadata. Agent Services exposes the Web Services interface needed by Content Integration Agent to perform the synchronization process.

- catalog.xml (also within Content Integration Agent), stores information about published objects. When the objects are “unpublished,” their information is deleted from catalog.xml.

The publishing process, illustrated in Figure 2, also applies to the synchronization event that takes place when published objects are modified or deleted, or new objects are added to the monitored repository.

**Figure 2:** Publishing to Content Server
System Defaults

Publishing an object to Content Server requires:

- A flex family to store the published object and its metadata (i.e., object type and attributes) as flex assets.
- A mappings file (mappings.xml), which maps the object’s metadata to counterpart asset types and assets in the flex family.

To give you a quick start, Content Integration Platform provides both a flex family for each supported source system and the default mappings.xml file. If your source objects are of the types specified in the mappings file (see Table 1), then those objects can be published to Content Server without your having to reconfigure either the mappings file or the flex family.

Default Mapping Framework

The default mapping framework in Content Integration Platform defines publishable objects to be of the type shown in Table 1, with the attributes shown in Table 1, on page 11.

The default mapping framework consists of two major components:

- A flex family for each source system. Each flex family is named after its source system: FileSystem, Documentum, SharePoint, and TeamUp.
  The purpose of a flex family is to model the source system’s default object types and attributes (listed in Table 1), in order to provide storage tables (i.e., asset type tables) for objects of those types and for their attributes.
- A pre-configured mappings.xml file, which maps the default object types and attributes in Table 1 to their counterpart assets in the flex families described above. (The mappings.xml file is located on the server that hosts Content Integration Agent.)

For complete specifications on the default mapping framework, see the appendices, at the end of this guide.
### Table 1: Source-System Metadata Supported by Default

<table>
<thead>
<tr>
<th>Source System</th>
<th>Default Object Types and Attributes</th>
<th>See Also ...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File System</strong></td>
<td><strong>File Type:</strong>&lt;br&gt;Folder type named <strong>Folder</strong>&lt;br&gt;Document type named <strong>Document</strong></td>
<td><strong>Attributes:</strong>&lt;sup&gt;a&lt;/sup&gt; contentAttr, contentURL, FileSize</td>
</tr>
<tr>
<td><strong>Documentum</strong></td>
<td><strong>Object Type:</strong>&lt;br&gt;Folder type named <strong>dm_folder</strong>&lt;br&gt;Document type named <strong>dm_document</strong></td>
<td><strong>Attributes:</strong>&lt;sup&gt;b&lt;/sup&gt; contentURL, contentAttr, r_version_label, r_full_content_size</td>
</tr>
<tr>
<td><strong>SharePoint</strong></td>
<td><strong>Content Type:</strong>&lt;br&gt;Folder type named <strong>Folder</strong>&lt;br&gt;Document type named <strong>Document</strong>&lt;br&gt;Picture type named <strong>Picture</strong></td>
<td><strong>Attributes:</strong>&lt;sup&gt;a&lt;/sup&gt; Documents: contentAttr, contentURL, File Size, Pictures: Date Picture Taken, Description, Keywords, Picture Height, Picture Width</td>
</tr>
<tr>
<td><strong>TeamUp</strong></td>
<td><strong>Metadata Type:</strong>&lt;br&gt;Folder&lt;br&gt;Document</td>
<td><strong>Attributes:</strong> keywords, description, contentAttr, contentURL</td>
</tr>
</tbody>
</table>

a. Attribute names are display names. The less commonly used system-defined attributes have been omitted from the default mappings.xml file and flex family. The attributes are: DateCreated, DateModified, MimeType

b. Attribute names are those listed in the “Types” node of the Documentum WebTop interface.
Implications for Publishing and Synchronization

The success of publishing and synchronization processes depends on the mappings.xml file and the source system’s flex family.

Default mappings.xml and Flex Families

Any objects of the type that are mapped in the default mappings.xml file (see also Table 1, on page 11) can be published to Content Server without your having to modify either mappings.xml or the default flex family. During publishing, the objects are automatically re-created in their respective flex family as either flex parent assets (if they are folders) or flex assets (if they are documents or pictures).

Following the publication process, changes to monitored folders and their contents (on the source side) are propagated to the flex family by the synchronization engine. For example, if you modify or delete published objects on the source side, or you create new objects (of the mapped types) in the monitored folders, your changes will be automatically propagated to the flex family (by the synchronization engine).

Custom mappings.xml and Flex Families

As the schema of the source system changes, the source system’s flex family must be updated in order for publishing and synchronization to produce the expected results. The mappings.xml file, however, may or may not require updates, depending on the nature of the schema changes. For example:

- If new document types are created for documents that will be published, both the flex family and mappings.xml must be updated with the new document types.
- If a newly added attribute will be propagated to Content Server, that attribute must be added to the relevant flex family and assigned to the relevant asset type definition.

Mapping an attribute, however, is conditional:

An attribute must be mapped (in mappings.xml) if it will be propagated to Content Server and it is named differently on the source and Content Server systems. (If attributes are named identically, they can be omitted from mappings.xml.)

Incorrect mapping of attributes does not stop the publication process, but it does produce a warning message and an entry in the log file. The publication process continues by skipping to the next publishable object.

Event Notification Option

When events occur in published folders on the source system, Content Integration Platform responds by synchronizing the source and target systems, as necessary. Content Integration Platform can be configured for event notification. Events, such as asset creation, would then trigger notices to CIP administrators, informing them of the events and whether synchronization took place. Notices are delivered to CIP administrators in a simple workflow process.

Content Integration Platform ships with several sample workflows. You have the option to enable any or all of the workflows during the CIP installation procedure.
Part 1

Installing and Publishing

This part contains the following chapters:

- Chapter 2, “Installing FatWire Content Integration Platform”
- Chapter 3, “Publishing”
Chapter 2

Installing FatWire Content Integration Platform

This chapter contains procedures for installing and configuring the FatWire Content Integration Platform to support publishing from file systems, Documentum installations, SharePoint systems, and TeamUp.

This chapter contains the following sections:

- Installation Overview
- Installing FatWire Content Integration Platform
- Verifying the Installation
- Publishing Production Data
Installation Overview

- Prerequisites
- Packaging
- Prerequisites

Prerequisites

- Microsoft Visual C++ 2008 redistributable (x86), which can be downloaded from http://www.microsoft.com
- OpenSSL, which can be downloaded from http://www.openssl.org

Packaging

FatWire Content Integration Platform is delivered as the following set of files:

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cipagent-vNo.msi (for Windows)</td>
<td>These files install Content Integration Agent (CIPCommander and service), and configuration files used to control the Agent process.</td>
</tr>
<tr>
<td>cipagent-vNo.rpm.bin (for Linux)</td>
<td></td>
</tr>
<tr>
<td>csagentservices.war</td>
<td>This file installs Content Server Agent Services, including property files used to set the detail of log files and regulate access to Content Server’s database.</td>
</tr>
<tr>
<td>cs_filesystem_schema.zip</td>
<td>This file installs the “FileSystem” flex family.</td>
</tr>
<tr>
<td>cs_documentum_schema.zip</td>
<td>This file installs the “Documentum” flex family.</td>
</tr>
<tr>
<td>cs_sharepoint_schema.zip</td>
<td>This file installs the “SharePoint” flex family.</td>
</tr>
<tr>
<td>cs_teamup_schema.zip</td>
<td>This file installs the “TeamUp” flex family.</td>
</tr>
</tbody>
</table>

Where to Install the Files

FatWire Content Integration Platform uses native executables. To install Content Integration Platform, you will do the following:

1. Install Content Integration Agent on any computer that runs on a supported operating system and can access both the source and target systems.

Note

For information about supported systems in this release, see the Supported Platform Document (SPD) for this content integration product. The SPD is available at:

http://e-docs.fatwire.com

to users with a FatWire Technical Support account. To obtain an account, register at:

http://support.fatwire.com
2. Deploy csagentservices.war to a system that has access to the Content Server Shared directory.

The Content Server system must be fully functional. **It must not be a production (delivery) system.**

3. Install the applicable schema on Content Server:
   - cs_filesystem_schema.zip
   - cs_documentum_schema.zip
   - cs_sharepoint_schema.zip
   - cs_teamup_schema.zip

Complete installation instructions are provided in the next section, “Installing FatWire Content Integration Platform.”

### Installing FatWire Content Integration Platform

**Note**

To ensure a smooth installation process, read the steps below to gain an understanding of the installation procedure and the information you will be asked to provide. To efficiently complete “Step III. Installing Schema on Content Server,” you must be an experienced Content Server administrator.

In this section, you will complete the following steps:

- Step I. Installing Content Integration Agent
- Step II. Installing Content Server Agent Services
- Step III. Installing Schema on Content Server

#### Step I. Installing Content Integration Agent

1. If you are using a Windows operating system, install Microsoft Visual C++ 2008 Redistributable Package (x86) on the same computer that will host Integration Agent. (The redistributable package is available for download from the Microsoft web site.)

2. Run the cipagent file on a computer that runs on a supported operating system and can access both the source and target systems.

   - **Windows:**

     Run cipagent-1.5.0.msi and follow the steps on the screen.

     The following folders are created in the target directory:

     ```
     bin
     cipagent.exe
     cipcommander.exe
     conf
     ... all conf files...
     security
     ... all certificates and private keys...
     logs
     ..log file...
     licenses
     ..licenses...
     ```
- **Linux:**

Run as a root user the following command on the source system:

```
./cipagent-1.0.0.rpm.bin
```

This command installs the following directories:

```
usr
local
bin
cipagent -exe
cipcommander
lib
cipagent
    ..all libraries...
share
cipagent
cnf
security
logs
licenses
```

3. **Back up the configuration file** `catalog.xml` *(located in integration_agent/conf/)*.

4. **Edit** `catalog.xml`.

The `catalog.xml` file stores configuration settings that are required by Integration Agent to connect to the source system and Content Server. You will edit this file to provide Integration Agent with system location and user information.

a. **Using a text editor**, open `catalog.xml`.

b. **Edit the connector to FatWire Content Server**.

   Locate the provider element with name “cs” and id “70b1e307-26a1-499c-9295-cf0b6bd01342” and set the following parameters:

   - **urlAS**: Point to the Web Services module deployed with Content Server. Only the host name and port need to be modified. Typically, these are the name of the host and port where Content Server is running. Do not alter the context name and context-related path unless you are sure they differ from the default (http://localhost:8080/csagentservices/InfostoriaService).

   - **username**: User name of the account that has permissions to modify Content Server’s database tables (e.g., fwadmin, the general administrator).

   - **password**: Above user’s password (e.g., xceladmin, assuming fwadmin as the username).

   - **context**: Leave this blank

   c. **If you are using Documentum**, edit the connector to the EMC Documentum installation.

      Locate the provider element with name “documentum” and id “d7a96a63-e78c-407c-8d7f-e84988806e49” and set the following parameters:

      - **urlDocumentum**: URL pointing to the server where Documentum Foundation Services (DFS) are running. Include the context name, but omit any context-related paths. Typically you need to modify only the host name (the default value is http://localhost:9090).
Chapter 2. Installing FatWire Content Integration Platform

Installing FatWire Content Integration Platform

19

Content Integration Platform 1.5 Administrator’s Guide

- **username**: User name for the account that has permissions to publishable content.
- **password**: Above user’s password.
- **filter**: This parameter contains a `where` clause which is applied to filter out document versions that are not for publication. For example, to publish the latest approved version of a document, you can specify values such as the following:
  ```
  r_policy_id = '0000000000000000'
  - or -
  r_current_state = 1
  ```
  (taking into account that the approved state has “1” as an index)

d. If you are using SharePoint, edit the connector to the SharePoint installation.
Locate the provider element with name “sharepoint” and id “7137dd5d-9ed7-4327-b4fd-8caeebd5889a”, and set the following parameters:
- **urlSharepoint**: URL pointing to the SharePoint site from which you plan to publish. Typically you need to modify only the host name (the default value is http://localhost).
- **username**: User name for the account that has permissions to publishable content.
- **password**: Above user’s password.

e. If you are using TeamUp, edit the connector to the TeamUp installation.
Locate the provider element with name “teamup” and id “91716a13-3955-41f7-80ee-f85629a1010d”, and set the following parameters:
- **urlAS**: URL pointing to the TeamUp Agent Services web application. Used to retrieve metadata. Typically you need to modify only the host name.
- **urlCDir**: URL pointing to the TeamUp Content Directory web application. Used to retrieve binary content. Typically you need to modify only the host name.
- **username**: User name for the account that has permissions to publishable content.
- **password**: Above user’s password.

f. Save `catalog.xml`.

5. Restart the Integration Agent executable:
- **For Windows**: Restart the FatWire Content Integration Agent service.
- **For Linux**: Type as root user: `/sbin/service restart cipagent`
Chapter 2. Installing FatWire Content Integration Platform

Chapter 2. Installing FatWire Content Integration Platform

Installing FatWire Content Integration Platform

6. Continue to the next step, “Step II. Installing Content Server Agent Services.”

Step II. Installing Content Server Agent Services

Note
The Integration Agent executable can be run as a standalone process or as a system daemon. The executable will start a simple HTTP server on the default port 7070, which is reserved for CIPCommander communications with Integration Agent. Port 7070 is bound to the localhost, and therefore does not expose your system to any additional security risks.

The fileserver facility default configuration takes port 7071 and attempts to automatically detect the host name. If you have more than one network interface installed on the machine where Agent is running, we advise changing auto to the DNS name or the IP address that is accessible from the CS Agent Services installation.

Should you need to change the port, edit the port designation in facilities.xml and add \(-p <\text{port}>\) to all commands that start CIPCommander.

1. Edit the following files in csagentservices.war (all the files are located in csagentservices/WEB-INF/classes):
   - commons-logging.properties: defines the log file and log detail settings
   - csAgentServices.properties: enables access to Content Server’s database
   a. Using a text editor, edit commons-logging.properties to point to the Agent Services log file (agentservices.log).
   b. Create a data source specific to the application server (more information is available in the Content Server installation guide for the application server you are using).
   c. Modify csAgentServices.properties to enable access to Content Server’s database.
      1) Using a text editor, set the following properties:
         - uploader.username: User name of an account with permissions to edit flex families.
         - uploader.password: Password for the provided user name.
         - cs.installDir: Content Server installation directory (e.g., C:\CS)
         - cs.url: Content Server URL. Point to the Content Server web application. The default value is: http://localhost:8080/cs
      2) Save csAgentServices.properties.
2. Deploy `csagentservices.war` on the application server on Content Server’s host.
3. Restart the application server.

**Step III. Installing Schema on Content Server**

In this step, you will import the applicable zip file(s), listed below, into Content Server:

- `cs_filesystem_schema.zip`
- `cs_documentum_schema.zip`
- `cs_sharepoint_schema.zip`
- `cs_teamup_schema.zip`

To install the schema

1. Run `catalogmover.bat` (or `catalogmover.sh` on Linux) from the Content Server installation directory.

    **Note**

    To use CatalogMover, you must connect it to Content Server:

    1. Choose **Server > Connect**.
    2. Provide the following information:
       - **Server**: The name of the HTTP server you want to connect to, and the port on which the server is running.
       - **Name**: `ContentServer`
       - **Password**: `<password>`
       - Below the “Password” field, select (or enter) a value that applies to your Content Server installation.
    3. Click **Connect**.

2. Go to **Catalog > Auto Import Catalog(s)**.
   a. Select the file to import.
   b. In the import dialog, fill in the fields as shown below:
      - **Catalog Data Directory**: Leave the default value
      - **Catalog ACL List**: `Browser,SiteGod,xceleditor,xceladmin`
   c. If necessary, import the remaining files.

3. Log in to Content Server’s Advanced interface as the general administrator (`fwadmin` / `xceladmin`, by default) and continue as follows:
   a. Enable each imported flex family for an existing content management site (names of flex family members begin with the name of the source system). You can also create a new site for the flex family (or families).
   b. For easy access to published content, create a tree tab (for example, `FileSystem` tab, `Documentum` tab, `SharePoint` tab, or `TeamUp` tab).
For instructions on enabling flex families, creating sites, and creating tree tabs, see the Content Server Administrator’s Guide.


**Step IV. Optional. Configuring Event Notification**

In this step, you will enable any or all of the sample workflows in order to notify CIP administrators of events at the source system and their synchronization.

---

**Note**

The following workflows must be installed enabled before an asset is published: CIPAssetDeleted and CIPAssetDeletionFailed

---

For information and instructions on installing sample workflows, see Chapter 4, “Configuring Event Notification.”

**Verifying the Installation**

In this step, you will publish a test folder to Content Server. You will also verify the synchronization process by adding, deleting, and modifying the test folder’s contents.

**To verify the installation**

1. **Back up** `mappings.xml` (located on the server that hosts Integration Agent).

2. **Select** (or create) a test folder with default metadata (defined in the `default mappings.xml` file). That is:
   - The folder type matches the folder type in `mappings.xml` (for quick reference, see Table 1, on page 11).
   - The test folder contains subfolders and documents whose folder type, document type, and attributes match those in `mappings.xml` (or Table 1).

   **Note**

   If you are verifying the publishing process from a SharePoint system, run a test on both the document library and picture library, using `mappings.xml` (or Table 1) to determine the default content types and attributes.

3. Test the publishing process. For instructions, see “Publishing to Content Server,” on page 26.

4. Test the synchronization process by renaming, moving, deleting, and creating subfolders and documents (including pictures in SharePoint). Test the attributes by deleting and adding them. When adding attributes, follow instructions in Chapter 5, “Remapping.”

5. Test the unpublish process by running the `unpublish` command (page 32).
Publishing Production Data

The quickest way to publish objects is to use the default `mappings.xml` file and flex families provided with CIP. Complete one of the following steps, depending on how your source system is configured:

- Objects are ready for publishing if their schema matches the default schema in `mappings.xml` and the default flex family. Follow the steps in Chapter 3, “Publishing.”

- Objects cannot be published successfully if their schema differs from the default schema in `mappings.xml` and the default flex family. Before publishing the objects, remap the schema. This requires you to update the flex family for your source system and reconfigure `mappings.xml`. For instructions, refer to Chapter 5, “Remapping.”

Customizing Your Installation

Content Integration Platform can be customized in many ways. For example:

- Default flex families can be modified for custom scenarios, or they can be replaced with custom flex families.

- Flex filters can be added to flex families. (If you implement flex filters, make sure to add the corresponding `jar` files to both the Content Server and CS Agent Services applications.)

- Attributes can be modified on the source system, or added to the source system.

- New document types can be added to the source system.

- Workflows can be implemented in order to notify administrators of CIP-related events that occur or fail to occur in Content Server, in response to changes on the source system.

Information about customizing a CIP installation can be found in Part 2, “Customizing CIP Installations.”
Chapter 3

Publishing

This chapter contains information about publishing to Content Server from file systems, Documentum installations, SharePoint systems, and FatWire TeamUp.

This chapter contains the following sections:

- Overview
- When an Object is Published
- Synchronization
- Unpublishing Command
Overview

Once the source system schema is mapped to Content Server, any object that is based on the schema can be published to Content Server.

Publishing from TeamUp

Only folders and their contents can be published from TeamUp to Content Server. All folders and their files must be local to TeamUp. (If they are first published to TeamUp, publishing to Content Server will fail.)

Synchronization

Manual publishing is unnecessary after the initial session, because the synchronization engine takes over. The engine monitors published folders (on the source system) and mirrors the changes it detects:

- Modification to published objects
- Deletion of published objects
- Addition of objects to the monitored folder(s), as long as their object types are mapped
- Attributes require special treatment, as explained in Chapter 5, “Remapping.”

The synchronization interval can be configured in the `publish` command (page 27).

Event notification for the synchronization process can be enabled by installing CIP-related default workflows, or specially creating custom workflows. For more information, see Chapter 4, “Configuring Event Notification.”

Unpublishing

The `unpublish` command is used to clear `catalog.xml` of all entries that are associated with published objects. Including the `-delete` parameter removes the same entries from Content Server’s database. For more information, see “Unpublishing Command,” on page 32.

Publishing to Content Server

If the source system schema is mapped to Content Server, you can successfully publish objects that are based on the schema. (Otherwise, you will need to remap the schema. Instructions are available in Chapter 5, “Remapping.”)

To publish to Content Server

1. Make sure Integration Agent is running.
2. Run the CIPCommander executable (located in the bin folder of the system where Integration Agent is installed):

```bash
cipcommander
  publish <source_providerid> <target_providerid>
  -source_repname <source_repname>
  -source_path <source_path>
  -source_repid <source_repid>
  -source_itemid <source_itemid>
  -target_repname <target_repname>
  -mapping <mapping_id>
  -replic_mode <full | ingestion>
  -bulk_resynch_interval <seconds>
```

where:

- `<source_providerid>` is the provider ID for the source system:
  - **File System**: 2023b849-688e-4009-af1f-903fe62d85b7
  - **Documentum**: d7a96a63-e78c-407c-8d7f-e84988806e49
  - **SharePoint**: 7137dd5d-9ed7-4327-b4fd-8caeebd5889a
  - **TeamUp**: 91716a13-3955-41f7-80ee-f85629a1010d

- `<target_providerid>` is the Content Server provider ID:
  - 70b1e307-26a1-4995-9295-cf0b6bd01342

Parameters and values for the `publish` command are defined in Table 2, “Publishing Parameters.”

Examples on the usage of the `publish` command are available on page 29.

Information about an object’s publication data is available in “When an Object is Published,” on page 31.

### Table 2: Publishing Parameters

<table>
<thead>
<tr>
<th>Publishing Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
| -source_repname      | `<source_repname>`: Name of the source repository from which content will be published. Enter the name exactly as it appears in the URL. Legal values:  
  - **File System**: Empty string (““).  
  - **Documentum**: Name of the cabinet that contains the folder to be published.  
  - **SharePoint**: Name of the document library or picture library that contains the folder to be published.  
  - **TeamUp**: Examples: See page 29. |
Table 2: Publishing Parameters

<table>
<thead>
<tr>
<th>Publishing Parameter</th>
<th>Value</th>
</tr>
</thead>
</table>
| -source_path         | `<source_path>`: Path to the object you want to publish. Legal values:  
  **File System:**  
  `/<folder>/<folder>/ ...`  
  (to publish the last folder in the path)  
  **Documentum:**  
  `/`  
  (to publish the cabinet specified by `<source_repname>`, including its contents)  
  `/<folder>/<folder> ...`  
  (to publish the last folder in the path)  
  **SharePoint:**  
  `/`  
  (to publish a library [document or picture] and its contents)  
  `/<folder>/<folder> ...`  
  (to publish the last folder in the path) |
| -source_repid        | For TeamUp systems only. `<source_repid>` is the repository ID of the **top folder** that you intend to publish. To obtain the repository ID, click the folder and note the integer in **Link: {repository : integer}** (in the “Tips” section, at the right side of the page). |
| -source_itemid       | For TeamUp systems only. `<source_itemid>` is the ID of the subfolder that you intend to publish (from within the top folder represented by `<source_repid>`, above). To obtain the subfolder’s `itemid`, click the subfolder and note the integer in **Link: {item : integer}** (in the “Tips” section, at the right side of the page). **Note:** If you omit -source_itemid, all content within the top folder `<source_repid>` will be published. |
| -target_repname      | `<target_repname>`: Name of the content management site (in Content Server) in which the flex family is enabled for this source system. |
| -mapping             | `<mapping_id>`: Value of the mapping id in mappings.xml. If you are using the default mappings.xml, enter one of the following values:  
  - documentum2cs  
  - sharepoint2cs  
  - filesystem2cs  
  - teamup2cs |
| -replic_mode         | `full | ingestion`  
  - `full` means that a full replication will be performed (by default).  
  - `ingestion` means that only item creation events will be propagated. Modifications and deletions on the source side will not be reflected on the target. |
| -bulk_resynch_interval | `<seconds>`: Number of seconds between two successive synchronization events. An optional publishing parameter. For more information, see “Synchronization,” on page 31. |
Examples

- Publishing from a File System
- Publishing from Documentum
- Publishing from SharePoint
- Publishing from TeamUp

Publishing from a File System

To publish the C:\publish folder to the “CIPDemo” content management site, using filesystem2cs default mapping:

```
cipcommander publish 2023b849-688e-4009-af1f-903fe62d85b7
70b1e307-26a1-499c-9295-cf0b6bd01342
-source_repname ""
-source_path C:\publish
-mapping filesystem2cs
-target_repname CIPDemo
```

Publishing from Documentum

- To publish the Images cabinet to the “CIPDemo” content management site, using documentum2cs default mapping:

```
cipcommander publish d7a96a63-e78c-407c-8d7f-e84988806e49
70b1e307-26a1-499c-9295-cf0b6bd01342
-source_repname Images
-source_path /Images
-mapping documentum2cs
-target_repname CIPDemo
```

- To publish the /Sample/Trees folder in the Images cabinet to the “CIPDemo” content management site, using documentum2cs default mapping:

```
cipcommander publish d7a96a63-e78c-407c-8d7f-e84988806e49
70b1e307-26a1-499c-9295-cf0b6bd01342
-source_repname Images
-source_path /Images/Sample/Trees
-mapping documentum2cs
-target_repname CIPDemo
```

Publishing from SharePoint

- To publish the Images picture library to the “CIPDemo” content management site using, sharepoint2cs default mapping:

```
cipcommander publish 7137dd5d-9ed7-4327-b4fd-8caeebd5889a
70b1e307-26a1-499c-9295-cf0b6bd01342
-source_repname Images
-source_path /
-mapping sharepoint2cs
-target_repname CIPDemo
```
• To publish the Cool/Bright folder in the Images picture library to the “CIPDemo” content management site, using sharepoint2cs default mapping:

```
cipcommander publish 7137dd5d-9ed7-4327-b4fd-8caeebd5889a
  70b1e307-26a1-499c-9295-cf0b6bd01342
  -source_repname Images
  -source_path /Cool/Bright
  -mapping sharepoint2cs
  -target_repname CIPDemo
```

**Publishing from TeamUp**

To publish the entire TeamUp repository to the “CIPDemo” content management site, using the teamup2cs default mapping:

```
cipcommander publish 91716a13-3955-41f7-80ee-f85629a1010d
  70b1e307-26a1-499c-9295-cf0b6bd01342
  -source_repid 3
  -mapping teamup2cs
  -target_repname CIPDemo
```
When an Object is Published

When an object is published, catalog.xml (located in integration_agent/conf/) is updated with an entry that identifies both the source system and the Content Server system (within the <workspace> tags), and specifies replication data for the published object (within the <replication> tag). The code below is an example of a publication entry for an object that was published from a file system:

```xml
<workspace id="8a55488d-97c9-4290-92e1-d7bb9e476dc7">
  <provider-ref refid="2023b849-688e-4009-afff-903fe62d85b7" />
  <init-params>
    <param name="repname" />
    <param name="path">c:/temp/CIP_xml/test1/test3</param>
    <param name="repid" />
  </init-params>
</workspace>

<workspace id="f917ddcb-1cbb-46c5-8840-fb194b693629">
  <provider-ref refid="70b1e307-26a1-499c-9295-cf0b6bd01342" />
  <init-params>
    <param name="repname">CIPTest</param>
    <param name="repid">39e1e988-f1a6-4913-a16d-45a1c5ad9976</param>
  </init-params>
</workspace>

<replication>
  <link id="04e4f0f2-8535-492b-8590-691a510884d4">
    <source-ref refid="8a55488d-97c9-4290-92e1-d7bb9e476dc7" />
    <target-ref refid="f917ddcb-1cbb-46c5-8840-fb194b693629" />
    <mapping-ref refid="filesystem2cs" />
    <init-params>
      <param name="BulkResynchInterval">600</param>
      <param name="ReplicMode">full</param>
    </init-params>
  </link>
</replication>

Synchronization

Once objects are published, the synchronization engine monitors the status of the folder(s) from which the objects were published and mirrors the changes it detects. For optimal performance, set the synchronization interval to a value that agrees with the frequency of updates to the source system. To set the synchronization interval, include the bulk_resynch_interval parameter in the publish command (page 27).
Unpublishing Command

You can unpublish objects from catalog.xml alone (and additionally, from Content Server) by executing the cipcommander unpublish command with parameters that suit your requirements.

**Note**

The `unpublish` command clears `catalog.xml` of all entries that are associated with published objects (for a sample publication entry, see the code on page 31). To remove the same entries from Content Server’s database, you must include the `-delete` parameter.

The `unpublish` command takes the following form and parameters:

```
cipcommander unpublish <parameters>
```

**Table 3: Unpublish Parameters**

<table>
<thead>
<tr>
<th>Unpublish Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-all</code></td>
<td>Use this parameter to clear <code>catalog.xml</code> of all publication entries.</td>
</tr>
</tbody>
</table>
| `-linkid`           | Use this parameter to clear `catalog.xml` of selected publication entries. `linkid` specifies the published object’s link to the Content Server system. Use the value in the published object’s `<link>` tag, which is nested within the object’s `<replication>` tag (for sample code, see page 31). For example:  
  - To unpublish a single object from `catalog.xml`, obtain its `<linkid>` and issue the following command:  
    ```
cipcommander unpublish linkid 04e4f0f2-8535-492b-8590-691a510884d4
    ```  
  - To unpublish multiple objects, add their `linkid`’s to the `unpublish` command. |
| `-delete`           | Use this parameter to remove, from Content Server’s database, the same objects that you are unpublishing from `catalog.xml`.  
 **Legal values:** `<true | false>`  
 **Default value:** `true` |
Part 2

Customizing CIP Installations

This part contains the following chapters:

- Chapter 4, “Configuring Event Notification”
- Chapter 5, “Remapping”
- Chapter 6, “Adding Assets to a Flex Family”
Chapter 4
Configuring Event Notification

• Overview
• Installing Sample Workflows
Overview

When changes are made to monitored folders (published folders on the source system), CIP administrators need confirmation that the same changes are either successfully propagated to Content Server during the synchronization process, or they fail to occur. To this end, Content Integration supports event notification.

Content Integration Platform can be configured to trigger notices to Content Server administrators, informing them of events that occur or fail to occur on the Content Server side when changes are made at the source system. The notices are delivered to administrators in a simple workflow process.

A default workflow is available for each of the following events, or failed events, on the Content Server side:

<table>
<thead>
<tr>
<th>Event in Content Server</th>
<th>Workflow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asset creation</td>
<td>CIPAssetCreated, invoked when an asset is created on the source system.</td>
</tr>
<tr>
<td>Asset deletion</td>
<td>CIPAssetDeleted, invoked when an asset is deleted from the source system.</td>
</tr>
<tr>
<td>Asset deletion failure</td>
<td>CIPAssetDeletionFailed, invoked when:</td>
</tr>
<tr>
<td></td>
<td>• The asset being deleted on the source system is checked out on the Content Server system.</td>
</tr>
<tr>
<td></td>
<td>• The asset being deleted has dependencies that become unresolved on the Content Server system if the asset were to be deleted.</td>
</tr>
<tr>
<td>Asset modification</td>
<td>CIPAssetModified, invoked when an asset is modified.</td>
</tr>
<tr>
<td>Asset modification failure</td>
<td>CIPAssetModificationFailed, invoked when the asset being modified on the source system is checked out on the Content Server system.</td>
</tr>
</tbody>
</table>

Custom Workflows

Although CIP-related workflows can be created from scratch, in most cases it is more convenient to start with the sample workflows packaged with the Content Integration Platform. (For instructions, see “Installing Sample Workflows,” on page 37.)

If you wish to create custom workflows, refer to the Content Server Administrator’s Guide for instructions.
Installing Sample Workflows

## Note
If you have already published to Content Server, install and enable only the following workflows for the published content: CIPAssetCreated, CIPAssetModified, and CIPAssetModificationFailed. The remaining workflows (CIPAssetDeleted and CIPAssetDeletionFailed) must be installed and enabled before an asset is published.

To install sample workflows

1. Run `catalogmover.bat` (or `catalogmover.sh` on Linux) from the Content Server installation directory.
2. Go to Catalog > Auto Import Catalog(s).
   a. Select `workflows.zip` (in the same directory or level as all `cs_*_schema.zip` files).
   b. In the import dialog, fill in the fields as shown below:
      - **Catalog Data Directory**: Leave the default value
      - **Catalog ACL List**: Browser,SiteGod,xceleditor,xceladmin
3. Create the sample workflows by invoking the following URL:

   ```
   http://<host>:<port>/<context_path>/ContentServer?pagename=OpenMarket/Xcelerate/Installation/CIPCreateWorkflows&username=<username>&<password>=<password>
   ```

   where:
   - **host** is the address of the Content Server installation
   - **port** is the port of the Content Server installation
   - **context_path** is the context path where the Content Server web application is deployed
   - **username** is the Content Server administrator’s user name
   - **password** is the Content Server administrator’s password

   For example, the URL of the default configuration is:

   ```
   ```

   When the workflows are installed, the following message will be displayed:

   “Workflows for Content Integration Platform were created successfully”
Verifying Sample Workflows

When the sample workflows are created, associated objects are also created in Content Server.

To verify the sample workflows and associated objects

1. Log in to the Content Server Advanced interface as an administrator.
2. Verify that the following objects have been created:
   - CIPAdmin role, which will be used as the management role for all CIP workflows. All users with the CIPAdmin role will be notified about all CIP events in the sample workflows.
   - Workflow processes: CIP Asset Created, CIP Asset Deleted, CIP Asset Deletion Failed, CIP Asset Modified, and CIP Asset Modification Failed
   - Workflow states: CIP Asset Created, CIP Asset Deleted, CIP Asset Deletion Failed, CIP Asset Modified, and CIP Asset Modification Failed
   - Workflow step action: CIP Asset Deleted, which results in an email notice to the CIP administrators.
   - Email object: CIP Asset Event

Enabling Sample Workflows

Sample workflows are pre-configured in the default mappings.xml file. Each asset type that is listed in the default mappings.xml file contains a commented workflow configuration section.

To enable a CIP workflow

1. Uncomment the following section for each asset type that you wish to enable for event notification:

   ```xml
   <descriptor-mapping sourceid="assetCreatedProcess" targetid="CIPAssetCreated" type="static" />
   <descriptor-mapping sourceid="assetModifiedProcess" targetid="CIPAssetModified" type="static" />
   <descriptor-mapping sourceid="assetDeletedProcess" targetid="CIPAssetDeleted" type="static" />
   <descriptor-mapping sourceid="assetDeletionFailedProcess" targetid="CIPAssetDeletionFailed" type="static" />
   ```

2. Assign the CIPAdmin role to CIP administrators. Ensure that CIP administrators are able to receive email. For instructions, see the Content Server Administrator’s Guide.
3. If the number of events occurring on the source system is relatively large, it is best to use workflow groups, as they will allow you to resolve tasks in bulk. Workflow groups are not packaged by default. They must be created manually, if required. For instructions on creating workflow groups, see the Content Server Administrator’s Guide.

**Note**

If a workflow group has the name of the invoked workflow process, the workflow process will be automatically added to the group.

Sample workflows behave in the following way:

- For creation, deletion failure, modification, and modification failure events, a task is assigned to all Content Server users with the CIPAdmin role. The task is simply a way of notifying the users of events and their propagation to the target system. The task can be removed; there is no obligation to take a step.

- When an event occurs, only the first step of the corresponding workflow is taken. If the option “Assign from list of participants” for the first step is chosen, all members of the selected roles will be assigned the next task. (For simplicity, notification workflows comprise one state and two steps.)

- For deletion events, all members with the CIPAdmin role receive an email notice. Because the asset no longer exists once the deletion event occurs, no tasks are displayed.
Chapter 5

Remapping

If you plan to publish from a source system whose schema differs from the default schema defined by the Content Integration Platform, you will have to update the flex family corresponding to the source system and possibly `mappings.xml`, depending on how the schema differ. This chapter identifies the commonly performed schema changes and provides remapping procedures.

This chapter contains the following sections:

- When to Remap
- Remapping Procedures
When to Remap

If you wish to publish an object whose schema somehow differs from the schema defined in the default `mappings.xml` file and flex family, you must update the flex family and possibly `mappings.xml`, depending on how the schema differ. Table 4 lists commonly made schema changes and the pages where you can find procedures for updating the relevant components.

**Table 4: Common Schema Changes**

<table>
<thead>
<tr>
<th>Common Schema Changes</th>
<th>For Remapping Procedures, See ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add new attribute</td>
<td>File System</td>
</tr>
<tr>
<td>Add new document type</td>
<td>page 43</td>
</tr>
<tr>
<td>Add new document type and attribute</td>
<td></td>
</tr>
<tr>
<td>Add new picture type</td>
<td></td>
</tr>
<tr>
<td>Add new picture type and attribute</td>
<td></td>
</tr>
</tbody>
</table>

**Note**

Procedures in the rest of this chapter call for updates to the default `mappings.xml` file and the default flex families, both supplied as part of the Content Integration Platform.

If you wish to create your own flex family, refer to the *Content Server Developer’s Guide* for instructions, keeping in mind the source system’s default flex family as a model. Basic procedures for remapping to a custom flex family remain the same (except for the names you choose for the family and its members).

If you create flex filters (for either a default flex family or a custom flex family), make sure to add the corresponding `.jar` files to both the Content Server and the Content Server Agent Services applications.
Remapping Procedures

- File Systems
- Documentum Systems
- SharePoint Systems
- TeamUp Systems

File Systems

If you create new attributes for documents of type Document and wish to propagate the attributes to Content Server, you must update the “FileSync” flex family with the new attributes. You must also update mappings.xml if the new attributes will be named differently on the Content Server side. (A list of default attributes and document types is available in Appendix A, “FileSync: Default Mapping Specifications.”)

To update the flex family and mappings.xml

1. Stop the Integration Agent.
2. Follow the remapping procedures in Table 5. For descriptions of tags and parameters, see page 44.
3. Test your changes by running a publishing session. For instructions, see Chapter 3, “Publishing.”

Table 5: Remapping File System Schema

<table>
<thead>
<tr>
<th>1. File System</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new attribute.</td>
<td>Create a “FileSync Attribute” asset. Create the attribute in the “FileSync” flex family and assign the attribute to the relevant asset type definition. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58.</td>
<td>Map the new attribute in mappings.xml only if the attribute is named differently on the Content Server side. Map the attribute within the &lt;descriptor-mapping&gt; tag, and nest the tag in the relevant object type mapping (example below).</td>
</tr>
<tr>
<td>E.g., CreationDate</td>
<td>E.g., fs_CreationDate</td>
<td>E.g., In this example, the CreationDate attribute belongs to the Document file type. Nest the attribute in the Document type mapping (as shown in bold type, below).</td>
</tr>
</tbody>
</table>

```
<assettype-mapping
  sourceid="Document"
  targetid="FileSync_Document;fs_document"
  id="fs_document">
  <descriptor-mapping
    sourceid="CreationDate"
    targetid="fs_CreationDate"
    type="static" />
</assettype-mapping>
```
### File System to CS Mapping: Tags and Parameters

```xml
<assettype-mapping
  sourceid="FileType"
  targetid="ChildAssetType | ParentAssetType;
             ChildDefinitionInstance | ParentDefinitionInstance"
  id="assettypeMappingID">
  <descriptor-mapping
    sourceid="FileSystemAttribute"
    targetid="CSAttribute"/>
</assettype-mapping>
```

#### File Systems

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-mapping</td>
<td>sourceid</td>
<td>Specifies the type of document that will be published. Legal values: Document</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Content Server’s counterpart to sourceid. For document objects, targetid takes the following value: ChildAssetType;ChildDefinitionInstance</td>
</tr>
</tbody>
</table>

**For document objects, targetid takes the following value:**
- ChildAssetType: Stores file system documents.
  - Default value: FileSystem_Document
- ChildDefinitionInstance:
  - Asset of type FileSystem Child Definition.
  - Defines the type of document that will be stored in ChildAssetType.
  - Default value: fs_document
  - Sample value: pdfDoc

**Example:**
If sourceid="pdfDoc" then targetid could be the following:
```
targetid="FileSystem_Document;fs_pdfDoc"
```
i.e., the (child) asset type named FileSystem_Document stores documents of type fs_pdfDoc.

**Note:** If you were to map a folder type, targetid would take the following value:
- ParentAssetType;ParentDefinitionInstance
  - ParentAssetType:
    - Stores file system folders.
    - Default value: FileSystem_Folder
  - ParentDefinitionInstance:
    - Asset of type FileSystem Parent Definition.
    - Defines the type of folder that will be stored in ParentAssetType.
    - Default value: fs_folder

For more information about the “FileSystem” flex family, see Appendix A, “File Systems: Default Mapping Specifications.”

| id | Unique value that identifies the asset type mapping. Default value: fs_document | fs_folder |
## File Systems

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>descriptor-mapping</td>
<td>sourceid</td>
<td>Attribute’s name in the file system.</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Attribute’s name in Content Server.</td>
</tr>
</tbody>
</table>
Documentum Systems

If you add new attributes and/or document types for the documents you will publish, you must update the “Documentum” flex family accordingly. You may also have to update mappings.xml, as explained in this section. (A list of default attributes and document types is available in Appendix B, “Documentum Systems: DefaultMapping Specifications.”)

To update the flex family and mappings.xml

1. Stop the Integration Agent.
2. Follow the procedures in Table 6. For descriptions of tags and parameters, see page 48.
3. Test your changes by running a publishing session. For instructions, see Chapter 3, “Publishing.”

Table 6: Remapping Documentum Schema

<table>
<thead>
<tr>
<th>1. Documentum</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new attribute.</td>
<td>Create a “Documentum Attribute” asset. Create the attribute in the “Documentum” flex family and assign the attribute to the relevant asset type definition. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58.</td>
<td>Map the new attribute in mappings.xml only if the attribute is named differently on the Content Server side. Map the attribute within the &lt;descriptor-mapping&gt; tag, and nest the tag in the relevant object type mapping (example below).</td>
</tr>
</tbody>
</table>

E.g., CreationDate | E.g., dm_CreationDate | E.g.,

In this example, the CreationDate attribute belongs to the dm_document object type. Nest the attribute in the dm_document type mapping (as shown in bold type, below).

<assettype-mapping
  sourceid="dm_document"
  targetid="Documentum_Document;dm_document"
  id="dm_document" extends="dm_sysobject">
  <descriptor-mapping
    sourceid="CreationDate"
    targetid="dm_CreationDate" />
</assettype-mapping>
### Table 6: Remapping Documentum Schema *(continued)*

<table>
<thead>
<tr>
<th>1. Documentum</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new document type.</td>
<td>Create a “Documentum Child Definition” asset. For instructions on creating child definition assets, see chapter 6.</td>
<td>Map the new document type in <code>mappings.xml</code>, using the <code>&lt;assettype-mapping&gt;</code> tag.</td>
</tr>
<tr>
<td>E.g., PDF</td>
<td>E.g., <code>dm_PDF</code></td>
<td>E.g., <code>&lt;assettype-mapping sourceid=&quot;PDF&quot; targetid=&quot;Documentum_Document;dm_PDF&quot; id=&quot;dm_PDF&quot; extends=&quot;dm_sysobject&quot;&gt;</code></td>
</tr>
</tbody>
</table>
| If you add a new attribute and new document type. | a. Create a “Documentum Attribute” asset. Create the attribute in the “Documentum” flex family. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58. 

b. Create a “Documentum Child Definition” asset. When creating the child definition asset, assign the new attribute to the asset. For instructions, see chapter 6. | a. Map the new document type `mappings.xml`, using the `<assettype-mapping>` tag (example below). 
b. Map the new attribute in `mappings.xml` only if the attribute is named differently on the Content Server side. Map the attribute within the `<descriptor-mapping>` tag, and nest the tag in the relevant object type mapping (example below). |
| E.g., Attribute: `CreationDate` New Document type: PDF | E.g., Attribute: `dm_CreationDate` New Document type: `dm_PDF` | E.g., `<assettype-mapping sourceid="PDF" targetid="Documentum_Document;dm_PDF" id="dm_PDF" extends="dm_sysobject">` 
<descriptor-mapping sourceid="CreationDate" targetid="dm_CreationDate" />` |

*Table 6: Remapping Documentum Schema (continued)*
Documentum to CS Mapping: Tags and Parameters

```xml
<assettype-mapping
  sourceid="DocumentumObjectType"
  targetid="ChildAssetType | ParentAssetType;
  ChildDefinitionInstance | ParentDefinitionInstance"
  id="assettypeMappingID" extends="dm_sysobject">
  <descriptor-mapping
    sourceid="DocumentumAttribute"
    targetid="CSAttribute"/>
</assettype-mapping>
```

### Documentum Systems

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-mapping</td>
<td>sourceid</td>
<td>Specifies the type of object that can be published.</td>
</tr>
<tr>
<td></td>
<td>Default value: dm_document</td>
<td>dm_folder</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Content Server’s counterpart to sourceid.</td>
</tr>
<tr>
<td></td>
<td>For document objects, targetid takes the following value:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ChildAssetType;ChildDefinitionInstance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ChildAssetType: Stores Documentum documents.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default value: Documentum_Document</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ChildDefinitionInstance:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asset of type Documentum Child Definition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines the type of document that will be stored in ChildAssetType.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default value: dm_document</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sample value: pdfDoc</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Example:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If sourceid=&quot;pdfDoc&quot;, then targetid could be the following:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>targetid=&quot;Documentum_Document;dm_pdfDoc&quot;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>i.e., the (child) asset type named Documentum_Document stores documents of type dm_pdfDoc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Note:</strong> If you were to map a folder type, targetid would take the following value:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ParentAssetType;ParentDefinitionInstance</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ParentAssetType: Stores Documentum folders.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default value: Documentum_Folder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- ParentDefinitionInstance:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Asset of type Documentum Parent Definition.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Defines the type of folder that will be stored in ParentAssetType.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Default value: dm_folder</td>
<td></td>
</tr>
<tr>
<td></td>
<td>For more information about the “Documentum” flex family, see Appendix B,</td>
<td></td>
</tr>
</tbody>
</table>

| id         | Unique value that identifies the asset type mapping.                                |
|            | By default, assettypeMappingID takes the following values:                          |
|            | dm_document | dm_folder |
Remapping Procedures

Asset Type-Mapping (continued)

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-mapping (continued)</td>
<td>id (continued)</td>
<td>extends is a pointer to the mapping id for the next-level object up the schema hierarchy. Because the Documentum object model requires that object types always have dm_sysobject defined as one of their top-level objects, the extends parameter is always used, although it is technically not required. Including the extends parameter allows attributes defined in parent types to be inherited by child types, starting at the dm_sysobject level. By default: extends=dm_sysobject</td>
</tr>
</tbody>
</table>

Descriptor-Mapping

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>descriptor-mapping</td>
<td>sourceid</td>
<td>Attribute’s name in the Documentum system.</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Attribute’s name in Content Server.</td>
</tr>
</tbody>
</table>
SharePoint Systems

If you add new attributes and/or document types for the documents you will publish, you must update the “SharePoint” flex family accordingly. You may also have to update mappings.xml, as explained in this section. (A list of default attributes and document types is available in Appendix C, “SharePoint Systems: Default Mapping Specifications.”)

To modify the flex family and mappings.xml

1. Stop the Integration Agent.
2. Follow the remapping procedures in Table 7. For descriptions of tags and parameters, see page 52.
3. Test your changes by running a publishing session. For instructions, see Chapter 3, “Publishing.”

Table 7: Remapping SharePoint Schema

<table>
<thead>
<tr>
<th>1. SharePoint</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new attribute.</td>
<td>Create a “SharePoint Attribute” asset. Create the attribute in the “SharePoint” flex family and assign the attribute to the relevant asset type definition. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58.</td>
<td>Map the new attribute in mappings.xml only if the attribute is named differently on the Content Server side. Map the attribute within the &lt;descriptor-mapping&gt; tag, and nest the tag in the relevant object type mapping (example below).</td>
</tr>
<tr>
<td>E.g., CreationDate</td>
<td>E.g., sp_CreationDate</td>
<td>E.g., In this example, the CreationDate attribute belongs to the Document content type. Nest the attribute in the Document type definition (as shown in bold type, below).</td>
</tr>
<tr>
<td>CreationDate</td>
<td></td>
<td>&lt;assettype-mapping sourceid=&quot;Document&quot; targetid=&quot;SharePoint_Document;sp_document&quot; id=&quot;sp_PDF&quot; extends=&quot;sp_item&quot;&gt; &lt;descriptor-mapping sourceid=&quot;CreationDate&quot; targetid=&quot;sp_CreationDate&quot;/&gt; &lt;/assettype-mapping&gt;</td>
</tr>
</tbody>
</table>
## Table 7: Remapping SharePoint Schema (continued)

<table>
<thead>
<tr>
<th>1. SharePoint</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new document type.</td>
<td>Create a “SharePoint Child Definition” asset. For instructions on creating child definition assets, see chapter 6.</td>
<td>Map the new document type in mappings.xml, using the <code>&lt;assettype-mapping&gt;</code> tag.</td>
</tr>
<tr>
<td>E.g., PDF</td>
<td>E.g., sp_PDF</td>
<td>E.g., <code>&lt;assettype-mapping sourceid=&quot;PDF&quot; targetid=&quot;SharePoint_Document;sp_PDF&quot; id=&quot;sp_PDF&quot; extends=&quot;sp_item&quot;&gt;</code></td>
</tr>
</tbody>
</table>

| If you add a new attribute and new document type. | a. Create a “SharePoint Attribute” asset. Create the attribute in the “SharePoint” flex family. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58. | a. Map the new document type in mappings.xml, using the `<assettype-mapping>` tag (example below). |
| | b. Create a “SharePoint Child Definition” asset. When creating the child definition asset, assign the new attribute to the asset. For instructions, see chapter 6. | b. Map the new attribute in mappings.xml only if the attribute is named differently on the Content Server side. Map the attribute within the `<descriptor-mapping>` tag, and nest the tag in the relevant object type mapping (example below). |
| E.g., Attribute: CreationDate New Document type: PDF | E.g., Attribute: sp_CreationDate New Document type: sp_PDF | E.g., `<assettype-mapping sourceid="PDF" targetid="SharePoint_Document;sp_PDF" id="sp_PDF" extends="sp_item">` `<descriptor-mapping sourceid="CreationDate" targetid="sp_CreationDate" />` |
SharePoint to CS Mapping: Tags and Parameters

```xml
<assettype-mapping
    sourceid="SharePointContentType"
    targetid="ChildAssetType | ParentAssetType;
    ChildDefinitionInstance | ParentDefinitionInstance"
    id="assettypeMappingID" extends="sp_item">
    <descriptor-mapping
        sourceid="SharePointAttribute"
        targetid="CSAttribute"/>
</assettype-mapping>
```

**SharePoint Systems**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-mapping</td>
<td>sourceid</td>
<td>Specifies the type of content that will be published. Default value: Document</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Content Server's counterpart to sourceid. For document and picture objects, targetid takes the following value: ChildAssetType;ChildDefinitionInstance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ChildAssetType: Stores SharePoint documents and pictures. Default value: SharePoint_Document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ChildDefinitionInstance: Asset of type SharePoint Child Definition. Defines the type of document (or picture) that will be stored in ChildAssetType. Default value: sp_document</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sample value: pdfDoc</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: If sourceid=&quot;pdfDoc&quot;, then targetid could be the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>targetid=&quot;SharePoint_Document;sp_pdfDoc&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i.e., the (child) asset type named SharePoint_Document stores documents of type sp_pdfDoc.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Note: If you were to map a folder type, targetid would take the following value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ParentAssetType;ParentDefinitionInstance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ParentAssetType: Stores SharePoint folders. Default value: SharePoint_Folder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ParentDefinitionInstance: Asset of type SharePoint Parent Definition. Defines the type of folder that will be stored in ParentAssetType. Default value: sp_folder</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For more information about the “SharePoint” flex family, see Appendix C, SharePoint Systems: Default Mapping Specifications.</td>
</tr>
<tr>
<td>id</td>
<td></td>
<td>Unique value that identifies the asset type mapping. By default, assettypeMappingID takes the following values: sp_document</td>
</tr>
</tbody>
</table>
Remapping Procedures

Chapter 5. Remapping

Remapping Procedures

Assettype-mapping (continued)  

id (continued)  

extends is a pointer to the mapping id for the next-level object up the schema hierarchy. Because the SharePoint content model requires that content types always have “item” defined as one of their top-level objects, the extends parameter will always be used, although it is not technically a required field. Including the extends parameter allows attributes defined in parent types to be inherited by child types, starting at the item level. By default:  

extends=sp_item

Descriptor-mapping

sourceid  

Attribute’s name in the SharePoint system.

targetid  

Attribute’s name in Content Server.
TeamUp Systems

If you add new properties for the documents you will publish, you must update the “TeamUp” flex family accordingly. You may also have to update mappings.xml, as explained in this section. (A list of default attributes and document types is available in Appendix D, “TeamUp Systems: Default Mapping Specifications.”)

To modify the flex family and mappings.xml

1. Stop the Integration Agent.
2. Follow the remapping procedures in Table 8. For descriptions of tags and parameters, see page 55.
3. Test your changes by running a publishing session. For instructions, see Chapter 3, “Publishing.”

Table 8: Remapping TeamUp Schema

<table>
<thead>
<tr>
<th>1. TeamUp</th>
<th>2. Content Server</th>
<th>3. Content Integration Agent: mappings.xml</th>
</tr>
</thead>
<tbody>
<tr>
<td>If you add a new property.</td>
<td>Create a “TeamUp Attribute” asset. Create the attribute in the “TeamUp” flex family and assign the attribute to the relevant asset type definition. For instructions, see chapter 6 and take note of the “Value” fields in step 5 on page 58.</td>
<td>Map the new property in mappings.xml only if the attribute is named differently on the Content Server side. Map the attribute within the &lt;descriptor-mapping&gt; tag, and nest the tag in the relevant object type mapping (example below).</td>
</tr>
</tbody>
</table>
| E.g., CreationDate | E.g., tu_CreationDate | E.g., In this example, the CreationDate property belongs to documents (and folders). Nest the property in the Document type definition (as shown in bold type, below).<assettype-mapping
sourceid="Document"
targetid="TeamUp_Document;tu_document">
  <descriptor-mapping
sourceid="CreationDate"
targetid="tu_CreationDate"/>
</assettype-mapping> |
TeamUp to CS Mapping: Tags and Parameters

```xml
<assettype-mapping
    sourceid="Folder | Document;http://ns.infostoria.com/wikis/RepositorySchema/topFolder_cipID/RepositoryItem"
    targetid="ChildAssetType | ParentAssetType;ChildDefinitionInstance | ParentDefinitionInstance"
    id="assettypeMappingID">
  <descriptor-mapping
    sourceid="TeamUpProperty"
    targetid="CSAttribute" />
</assettype-mapping>
```

### TeamUp Systems

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-mapping</td>
<td>sourceid</td>
<td>Specifies the type of item that will be published: folder or document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Note:</strong> For TeamUp, the folder type is always identified as <em>Folder</em>. The document type is always identified as <em>Document</em>. Sourceid contains the variable <em>topFolder_cipID</em>. This variable applies to the top folder that contains the subfolders and documents you plan to publish. The <em>topFolder_cipID</em> is displayed as a numeric value, called “CIP ID” in the “Tips” sidebar, when you click the folder.</td>
</tr>
<tr>
<td></td>
<td>targetid</td>
<td>Content Server’s counterpart to sourceid.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>While TeamUp folders and documents are always identified as <em>Folder</em> and <em>Document</em>, they can be differentiated on the Content Server side. For example, <em>Folder</em> can be mapped to <em>tu_pdfFolder</em>, and <em>Document</em> can be mapped to <em>tu_pdfDoc</em>. For document objects, targetid takes the following value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td><em>ChildAssetType;ChildDefinitionInstance</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <em>ChildAssetType</em>: Stores TeamUp documents.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Default value:</strong> <em>TeamUp_Document</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <em>ChildDefinitionInstance</em>: Asset of type TeamUp Child Definition. Defines the type of document that will be stored in ChildAssetType.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Default value:</strong> <em>tu_document</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>Sample value:</strong> <em>tu_pdfDoc</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Example:</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If sourceid=&quot;Document;<a href="http://ns.infostoria.com/wikis/RepositorySchema/4316/RepositoryItem">http://ns.infostoria.com/wikis/RepositorySchema/4316/RepositoryItem</a>&quot; then targetid could be the following:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>targetid=&quot;TeamUp_Document;tu_pdfDoc&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i.e., the (child) asset type named <em>TeamUp_Document</em> stores documents of type <em>tu_pdfDoc</em>.</td>
</tr>
</tbody>
</table>
### TeamUp Systems

<table>
<thead>
<tr>
<th>Tag</th>
<th>Parameter</th>
<th>Parameter Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>assettype-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(continued)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>id</td>
<td></td>
<td>Unique value that identifies the asset type mapping.</td>
</tr>
<tr>
<td>descriptor-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sourceid</td>
<td></td>
<td>Attribute’s name in TeamUp.</td>
</tr>
<tr>
<td>targetid</td>
<td></td>
<td>Attribute’s name in Content Server.</td>
</tr>
</tbody>
</table>

**Note:** If you were to map the Folder type, targetid would take the following value:

- **ParentAssetType:** Stores TeamUp folders.
  - **Default value:** TeamUp_Folder
- **ParentDefinitionInstance:** Asset of type TeamUp Parent Definition. Defines the type of folder that will be stored in ParentAssetType.
  - **Default value:** tu_folder

For more information about the “TeamUp” flex family, see Appendix C, “SharePoint Systems: Default Mapping Specifications.”
Chapter 6

Adding Assets to a Flex Family

This chapter contains procedures that support the remapping steps in chapter 5.
This chapter contains the following sections:
• Adding a New Attribute
• Adding a New Child Definition Asset
Adding a New Attribute

If you define new attributes on the source system and associate them with an object that is or will be published, you must create the corresponding attributes in Content Server. Otherwise, objects associated with the attributes cannot be properly synchronized (or published).

**To add a new attribute to Content Server**

1. Log in to Content Server’s Advanced interface as an administrator.
2. Select the site in which the flex family for your source system is enabled.
3. Select New from the top navigation bar.
4. Select New SourceSystem Attribute from the list of options.
5. When filling in fields in the “SourceSystem Attribute” form, fill in the following fields as explained below:
   - **Value Type**: Select a value type that is as close as possible to the corresponding attribute’s data type on the source system. Although you can map any data type to a string, you gain flexibility by mapping to a similar data type. (For example, a date for a given locale cannot be modified for any other locale unless it is mapped to the date type.)
     - If you are using EMC Documentum or Microsoft SharePoint, use the tables on page 59 for suggested conversions.
     - For TeamUp, all attributes map to string.
   - **Number of Values**:
     - If you are using EMC Documentum and re-creating a “repeating value” attribute, select the multivalued option.
     - If you are using SharePoint and re-creating a multivalued attribute, select the multivalued option.
     - For file systems, all attributes are single-valued.
6. Save the attribute.
7. Assign the attribute to the relevant flex parent definition or flex child definition.
Documentum attributes:

Table 9: Suggested Conversions for Documentum Attributes

<table>
<thead>
<tr>
<th>Documentum Attribute Type</th>
<th>CS Attribute Type</th>
<th>Documentum Attribute Type</th>
<th>CS Attribute Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>string</td>
<td>Integer</td>
<td>int</td>
</tr>
<tr>
<td>Double</td>
<td>float</td>
<td>String</td>
<td>string</td>
</tr>
<tr>
<td>ID</td>
<td>string</td>
<td>Time</td>
<td>date</td>
</tr>
</tbody>
</table>
### SharePoint attributes:

**Table 10: Suggested Conversions for SharePoint Attributes**

<table>
<thead>
<tr>
<th>SharePoint Attribute Type</th>
<th>CS Attribute Type</th>
<th>Multi-valued</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single line of text</td>
<td>string / text</td>
<td></td>
</tr>
<tr>
<td>Multiple lines of text</td>
<td>string / text</td>
<td></td>
</tr>
<tr>
<td>Choice (menu to choose from)</td>
<td>string</td>
<td>single- or multi-valued, depending on selection</td>
</tr>
<tr>
<td>Number (1, 1.0, 100)</td>
<td>float</td>
<td></td>
</tr>
<tr>
<td>Currency ($, ¥, €)</td>
<td>money</td>
<td></td>
</tr>
<tr>
<td>Date and Time</td>
<td>date</td>
<td></td>
</tr>
<tr>
<td>Lookup (information already on this site)</td>
<td>string</td>
<td>single- or multi-valued, depending on selection</td>
</tr>
<tr>
<td>Yes/No (check box)</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Person or Group</td>
<td>string</td>
<td>single- or multi-valued, depending on selection</td>
</tr>
<tr>
<td>Hyperlink or Picture</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td>Calculated (calculation based on other columns)</td>
<td>string</td>
<td></td>
</tr>
</tbody>
</table>
Adding a New Child Definition Asset

Note
If you are creating a child definition asset with new attributes, create the attributes first (as shown on page 61). You will assign them to the child definition asset during the procedure below.

To add a new child definition asset to Content Server
1. Log in to Content Server’s Advanced interface as an administrator.
2. Select the site in which the flex family is enabled.
3. Select New from the top navigation bar.
4. Select New Source System Child Definition from the list of options and fill in the form that appears.
5. Save the asset.
This part contains specifications for the default mappings that come with FatWire Content Integration Platform.

This section contains the following appendices:

- Appendix D, “TeamUp Systems: Default Mapping Specifications”
Appendix A

File Systems: Default Mapping Specifications

This appendix contains the following sections:

- Overview
- Default mappings.xml
- ‘FileSystem’ Flex Family Specifications
Overview

The default mapping framework in the Content Integration Platform enables publishing from file systems, as long as the publishable content is based on the following default constructs of the file system: Folder and Document file types, and attributes contentAttr, contentURL, and FileSize.

The mapping framework supplies the following default components:

- The “FileSystem” flex family, pre-configured to match the file types and attributes listed above.
- A mappings.xml file, in which file types and attributes (listed above) are mapped to assets in the “FileSystem” flex family:
  - The Folder file type is mapped to a flex parent definition asset named fs_folder.
  - The Document file type is mapped to a flex definition asset named fs_document.
  - Attributes are mapped to flex assets of type “FileSystem Attribute.”

The mapping is illustrated in Figure A-1, on page 67, coded in mappings.xml, on page 69, and summarized in Table A-1, on page 70.

Once the mapping is established, folders of type Folder (and their contents) can be published:

- Folders are published as flex parent assets to the “FileSystem Folder” asset type.
- Documents are published as flex assets to the “FileSystem Document” asset type.

Publishing to the “FileSystem” flex family is summarized below.

During publishing, the Content Integration Platform refers to the mappings.xml file to determine the types of objects to publish (Folder and Document). The folder that is named in the publish command is the starting point of the publication process. The folder is published as a flex parent asset of type “FileSystem Folder,” along with all the subfolders and documents it contains.

To reproduce the folder’s structure, (subfolders and documents), the Content Integration Platform refers to path information. If subfolders exist, the Content Integration Platform chains their counterpart “FileSystem Folder” assets to reproduce the hierarchy.

Documents, treated as flex assets of type “FileSystem Document,” are placed under their respective “FileSystem Folder” parent assets.
Figure A-1: Mapping the Folder Type to the “FileSystem” Flex Family

File System

Files of type Folder published as Assets of type “FileSystem Folder”

Published Content:
- Folder: Gems
- Folder: Minerals

Folder File Type: Folder maps to <AssetType>;<ParentDefinitionInstance>

Mapping:

<assettype-mapping
  sourceid="Folder" targetid="FileSystem_Folder;fs_folder"
  id="fs_folder">
</assettype-mapping>

See “Default mappings.xml,” on page 69 for the code in its context. See also page 70 for flex family specifications.
Figure A-2: Mapping the Document Type to the “FileSystem” Flex Family

Files of type Document published as Assets of type “FileSystem Document”

Published Content:

Document: Sapphire.jpg
Document: Emerald.png

Asset: Sapphire.jpg
Asset: Emerald.png

Document File Type: 

Mapping:

<assettype-mapping
  sourceid="Document" targetid="FileSystem_Document;fs_document"
  id="fs_document">
</assettype-mapping>

See “Default mappings.xml,” on page 69 for the code in its context (including the attributes that are associated with the Document file type). See also page 70 for flex family specifications.
The default mappings.xml file, located in the Content Integration Agent, maps the file system’s default schema to Content Server’s “FileSystem” flex family. The mapping is uniquely identified by the ID in line 3. Parameters for the <assettype-mapping> and <descriptor-mapping> tags are defined in “File System to CS Mapping: Tags and Parameters,” on page 44.

```xml
<assettype-mapping
    sourceid="Document"
    targetid="FileSystem_Document;fs_document" id="fs_document">  
<descriptor-mapping
    sourceid="contentAttr"
    targetid="file" type="static" /> 
<descriptor-mapping
    sourceid="contentURL"
    targetid="contentURL" type="dynamic" /> 
<descriptor-mapping
    sourceid="FileSize"
    targetid="file_size" />

```
‘FileSystem’ Flex Family Specifications

Table A-1 summarizes the default mapping of file system schema to Content Server’s “FileSystem” flex family. For customized implementations, you can either re-use the flex family or create your own.

Table A-1: FileSystem Default Data and Flex Family Analogs

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>File System Default Data</th>
<th>Maps To:</th>
<th>Assets of This Type Are Created By ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td></td>
<td>Flex Asset Type</td>
<td>Flex Family Member</td>
</tr>
<tr>
<td>Attribute</td>
<td>Attributes (for documents):<strong>a</strong>&lt;br&gt;• contentURL&lt;br&gt;• contentAttr&lt;br&gt;• FileSize</td>
<td>FileSystem Attribute&lt;br&gt;Stores attribute instances:<strong>b</strong>&lt;br&gt;• contentURL&lt;br&gt;• file&lt;br&gt;• file_size</td>
<td>Flex Attribute</td>
</tr>
<tr>
<td>Folder File Type</td>
<td>Folder</td>
<td>FileSystem Parent Definition&lt;br&gt;Stores parent definition instance: fs_folder</td>
<td>Flex Parent Definition</td>
</tr>
<tr>
<td>Document File Type</td>
<td>Document</td>
<td>FileSystem Child Definition&lt;br&gt;Stores child definition instance: fs_document</td>
<td>Flex (Child) Definition</td>
</tr>
<tr>
<td>Content</td>
<td>Published Folders&lt;br&gt;Folders of file type Folder</td>
<td>FileSystem Folder&lt;br&gt;Stores flex parent assets (of type fs_folder and any other type)</td>
<td>Flex Parent</td>
</tr>
<tr>
<td></td>
<td>Published Documents&lt;br&gt;Documents of file type Document</td>
<td>FileSystem Document&lt;br&gt;Stores flex (child) assets (of type fs_document and any other type)</td>
<td>Flex (Child) Asset</td>
</tr>
</tbody>
</table>

**a**. Attribute names are display names. The less commonly used system-defined attributes have been omitted from the default mappings.xml file and flex family. The attributes are: DateCreated, DateModified, MimeType

**b**. “Instance” means “asset.” “Instance” is used only to help differentiate metadata (instances) from published content (assets).
Appendix B

Documentum Systems: Default Mapping Specifications

This appendix contains the following sections:

- Overview
- Default mappings.xml
- ‘Documentum’ Flex Family Specifications
Overview

The default mapping framework in the Content Integration Platform enables publishing from Documentum systems, as long as the publishable content is based on the following default Documentum constructs: Object types `Document(dm_document)` and `Folder(dm_folder)`, and attributes `contentURL`, `contentAttr`, `r_version_label`, and `r_full_content_size`.

The mapping framework supplies the following default components:

- The “Documentum” flex family, pre-configured to match the object types and attributes listed above.
- A `mappings.xml` file, in which object types and attributes (listed above) are mapped to assets in the “Documentum” flex family:
  - The `dm_folder` type is mapped to a flex `parent` definition asset named `dm_folder`.
  - The `dm_document` type is mapped to a flex definition asset named `dm_document`.
  - Attributes are mapped to flex assets of type “Documentum Attribute.”

The mappings are illustrated in Figure B-1, on page 73, coded in `mappings.xml`, on page 75, and summarized in Table B-1, on page 76.

Once the mapping is established, folders of type `dm_folder` (and their contents) can be published:

- Folders are published as flex `parent` assets to the “Documentum Folder” asset type.
- Documents are published as flex assets to the “Documentum Document” asset type.

Publishing to the “Documentum” flex family is summarized below.

During publishing, the Content Integration Platform refers to the `mappings.xml` file to determine the types of objects to publish (`dm_folder` and `dm_document`). The folder that is named in the `publish` command is the starting point of the publication process. The folder is published as a flex parent asset of type “Documentum Folder,” along with all the subfolders and documents it contains.

To reproduce the folder’s structure, (subfolders and documents), the Content Integration Platform refers to path information. If subfolders exist, the Content Integration Platform chains the counterpart “Documentum Folder” assets in order to reproduce the hierarchy. Documents, treated as flex assets of type “Documentum Document,” are placed under their respective “Documentum Folder” parent assets.

![Diagram showing folder structure with nested subfolders and documents.](image-url)
Figure B-1: Mapping the dm_folder Object Type to the "Documentum" Flex Family

Objects of type dm_folder published as Assets of type "Documentum Folder"

Folder: Gems
Folder: Minerals
Asset: Gems
Asset: Minerals

Folder Object Type: dm_folder maps to <AssetType>;<ParentDefinitionInstance>

Mapping:

<assettype-mapping
    sourceid="dm_folder" targetid="Documentum_Folder:dm_folder"
    id="dm_folder">
</assettype-mapping>

See "Default mappings.xml," on page 75 for the code in its context. See also page 76 for flex family specifications.
Figure B-2: Mapping the dm_document Object Type to the “Documentum” Flex Family

Documentum

Objects of type dm_document published as Assets of type “Documentum Document”

Published Content:
- Document: Sapphire.jpg
- Document: Emerald.png

Document Object Type: dm_document maps to <AssetType>;<ChildDefinitionInstance>

Mapping:

```xml
<assettype-mapping
  sourceid="dm_document" targetid="Documentum_Document;dm_document"
  id="dm_document">
</assettype-mapping>
```

See “Default mappings.xml,” on page 75 for the code in its context (including attributes that are associated with the dm_document object type). See also page 76 for flex family specifications.
Appendix B. Documentum Systems: Default Mapping Specifications

The default mappings.xml file, located in the Content Integration Agent, maps Documentum schema to Content Server’s “Documentum” flex family. The mapping is uniquely identified by the ID in line 3. Parameters for the <assettype-mapping> and <descriptor-mapping> tags are defined in “Documentum to CS Mapping: Tags and Parameters,” on page 48.

1 <mappings>
2 <!-- EMC Documentum to FatWire CS -->
3 <mapping id="documentum2cs">
4  <assettype-mapping
5     sourceid="dm_sysobject"
6     targetid=""
7     id="dm_sysobject">
8     <descriptor-mapping sourceid="title" targetid="title" />
9     <descriptor-mapping sourceid="subject" targetid="subject" />
10    <descriptor-mapping sourceid="keywords" targetid="keywords" />
11    </assettype-mapping>
12    <assettype-mapping
13     sourceid="dm_document"
14     targetid="Documentum_Document;dm_document"
15     id="dm_document" extends="dm_sysobject">
16     <descriptor-mapping
17     sourceid="contentURL" targetid="contentURL" type="dynamic" />
18     <descriptor-mapping
19     sourceid="contentAttr" targetid="file" type="static" />
20     <descriptor-mapping
21     sourceid="r_version_label" targetid="version_label" />
22     <descriptor-mapping
23     sourceid="r_full_content_size" targetid="file_size" />
24    </assettype-mapping>
25    <assettype-mapping
26     sourceid="dm_folder"
27     targetid="Documentum_Folder;dm_folder"
28     id="dm_folder" extends="dm_sysobject">
29    </assettype-mapping>
30   </mapping>
31 </mappings>
‘Documentum’ Flex Family Specifications

Table B-1 summarizes the default mapping of Documentum schema to Content Server’s “Documentum” flex family. For customized implementations, you can either re-use the flex family or create your own.

Table B-1: Documentum Default Data and Flex Family Analogs

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>Documentum Default Data</th>
<th>Maps To:</th>
<th>Flex Family Member</th>
<th>Description</th>
<th>Assets of This Type Are Created By ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Attributes (for documents):(^a)</td>
<td>Documentum Attribute</td>
<td>Flex Attribute</td>
<td>This flex asset type stores attributes for documents.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td></td>
<td>• contentURL</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• contentAttr</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• r_version_label</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• r_full_content_size</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Folder Object Type</td>
<td>dm_folder</td>
<td>Documentum Parent Definition</td>
<td>Flex Parent Definition</td>
<td>This flex asset type stores parent definition instances.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td>Document Object Type</td>
<td>dm_document</td>
<td>Documentum Child Definition</td>
<td>Flex (Child) Definition</td>
<td>This flex asset type stores child definition instances.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td>Content</td>
<td>Published Folders</td>
<td>Documentum Folder</td>
<td>Flex Parent</td>
<td>This flex asset type stores folder assets. For an example, see Figure B-1, on page 73.</td>
<td>Content Integration Platform</td>
</tr>
<tr>
<td></td>
<td>Folders of object type Folder</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Published Documents</td>
<td>Documentum Document</td>
<td>Flex (Child) Asset</td>
<td>This flex asset type stores document assets. For an example, see Figure B-2, on page 74.</td>
<td>Content Integration Platform</td>
</tr>
<tr>
<td></td>
<td>Documents of object type Document</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^a\) Attribute names are those listed in the “Types” node of the Documentum WebTop interface.

\(^b\) “Instance” means “asset.” “Instance” is used only to help differentiate metadata (instances) from published content (assets).
Appendix C

SharePoint Systems: Default Mapping Specifications

This appendix contains the following sections:

- Overview
- Default mappings.xml
- ‘SharePoint’ Flex Family Specifications
Overview

The default mapping framework in the Content Integration Platform enables publishing from SharePoint libraries on selected sites, as long as the publishable content is based on the following default SharePoint constructs: Content types Document, Folder, and Picture; and attributes File Size, contentAttr, contentURL, Date Picture Taken, Description, Keywords, Picture Height, and Picture Width.

The mapping framework supplies the following default components:

- The “SharePoint” flex family, pre-configured to match the content types and attributes listed above.
- A mappings.xml file, in which content types and attributes (listed above) are mapped to assets in the “SharePoint” flex family:
  - The Folder type is mapped to a flex parent definition asset named sp_folder.
  - The Document type is mapped to a flex definition asset named sp_document.
  - The Picture type is mapped to a flex definition asset named sp_picture.
  - Attributes are mapped to flex assets of type “SharePoint Attribute.”

The mappings are illustrated in Figure C-1, on page 79, coded in mappings.xml, on page 81, and summarized in Table C-1, on page 82.

Once the mapping is established, folders of type Folder (and their contents) can be published:

- Folders are published as flex parent assets to the “SharePoint Folder” asset type.
- Documents and pictures are published as flex assets to the “SharePoint Document” asset type.

Publishing to the SharePoint flex family is summarized below.

During publishing, the Content Integration Platform refers to the mappings.xml file to determine the types of content to publish (sp_folder, sp_document, and sp_picture). The folder that is named in the publish command is the starting point of the publication process. The folder is published as a flex parent asset of type “SharePoint Folder,” along with all the subfolders, documents, and pictures it contains.

To reproduce the folder’s structure, (subfolders, documents, and pictures), the Content Integration Platform refers to path information:

- If subfolders exist, the Content Integration Platform chains their corresponding “SharePoint Folder” assets to reproduce the hierarchy.
- Documents, treated as “SharePoint Document” assets, are placed under their respective “SharePoint Folder” parent assets.
- Pictures, treated as “SharePoint Document” assets, are placed under their respective “SharePoint Folder” parent assets.
Figure C-1: Mapping the Folder Type to Content Server’s “SharePoint” Flex Family

Published Content:
- Folder: Gems
- Folder: Minerals

Content of type Folder published as Assets of type “SharePoint Folder”

Folder Content Type:
- Folder maps to <AssetType>;<ParentDefinitionInstance>

Mapping:
```
<assettype-mapping
    sourceid="Folder" targetid="SharePoint_Folder;sp_folder"
    id="sp_folder">
</assettype-mapping>
```

See “Default mappings.xml,” on page 81 for the code in its context. See also page 82 for flex family specifications.
Figure C-2: Mapping the Document Type to Content Server’s “SharePoint” Flex Family

SharePoint

Content of type Document published as Assets of type “SharePoint Document”

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Emerald</td>
</tr>
<tr>
<td></td>
<td>Sapphire</td>
</tr>
</tbody>
</table>

Document: Sapphire
Document: Emerald

Content Type: Document <AssetType>;<ChildDefinitionInstance>

Mapping:

```xml
<assettype-mapping
  sourceid="Document" targetid="SharePoint_Document;sp_document"
  id="sp_document" extends="sp_document">
</assettype-mapping>
```

See “Default mappings.xml,” on page 81 for the code in its context (including attributes that are associated with the Document content type). See also page 82 for flex family specifications.

Note

The Picture content type is mapped similarly to the Document content type:

- `sourceid` takes the value Picture
- `targetid` takes the value `SharePoint_Document;sp_picture`

See “Default mappings.xml,” on page 81 for the code (line 13).
Default mappings.xml

The default mappings.xml file, located in the Content Integration Agent, maps SharePoint schema to Content Server’s “SharePoint” flex family. The mapping is uniquely identified by the ID in line 3. Parameters for the <assettype-mapping> and <descriptor-mapping> tags are defined in “SharePoint to CS Mapping: Tags and Parameters,” on page 52.

```
1 <mappings>
2  <!-- MS SharePoint to CS -->
3  <mapping id="sharepoint2cs">
4    <assettype-mapping sourceid="Item" targetid="" id="sp_item">
5      <descriptor-mapping sourceid="Title" targetid="title" />
6    </assettype-mapping>
7    <assettype-mapping sourceid="Document" targetid="SharePoint_Document;sp_document" id="sp_document" extends="sp_item">
8      <descriptor-mapping sourceid="File Size" targetid="file_size" />
9      <descriptor-mapping sourceid="contentAttr" targetid="file" type="static" />
10     <descriptor-mapping sourceid="contentURL" targetid="contentURL" type="dynamic" />
11    </assettype-mapping>
12    <assettype-mapping sourceid="Folder" targetid="SharePoint_Folder;sp_folder" id="sp_folder" extends="sp_item" />
13    <assettype-mapping sourceid="Picture" targetid="SharePoint_Document;sp_picture" id="sp_picture" extends="sp_document">
14      <descriptor-mapping sourceid="Date Picture Taken" targetid="date_taken" />
15      <descriptor-mapping sourceid="Description" targetid="description" />
16      <descriptor-mapping sourceid="Keywords" targetid="keywords" />
17      <descriptor-mapping sourceid="Picture Height" targetid="picture_height" />
18      <descriptor-mapping sourceid="Picture Width" targetid="picture_width" />
19    </assettype-mapping>
20  </mapping>
21 </mappings>
```
‘SharePoint’ Flex Family Specifications

Table C-1 summarizes the default mapping of SharePoint schema to Content Server’s “SharePoint” flex family. For customized implementations, you can either re-use the flex family or create your own.

Table C-1: SharePoint Default Data and Flex Family Analogs

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>SharePoint Default Data</th>
<th>Maps To:</th>
<th>Assets of This Type Are Created By ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attributes</td>
<td>SharePoint Attribute</td>
<td>Flex Asset Type</td>
<td>Flex Family Member</td>
</tr>
<tr>
<td>For documents:</td>
<td></td>
<td>SharePoint Attribute</td>
<td>Flex Attribute</td>
</tr>
<tr>
<td>Date, Size, etc.</td>
<td></td>
<td>Flex Parent Definition</td>
<td>Flex Parent</td>
</tr>
<tr>
<td>For pictures:</td>
<td></td>
<td>Flex (Child) Definition</td>
<td>Flex (Child)</td>
</tr>
<tr>
<td>Date, Size, etc.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a. “Instance” means “asset.” “Instance” is used only to help differentiate metadata (instances) from published content (assets).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

a. “Instance” means “asset.” “Instance” is used only to help differentiate metadata (instances) from published content (assets).
Appendix D

TeamUp Systems: Default Mapping Specifications

This appendix contains the following sections:

- Overview
- Default mappings.xml
- ‘TeamUp’ Flex Family Specifications
Overview

The default mapping framework in Content Integration Platform enables publishing from TeamUp repositories, as long as the publishable content is based on the following default TeamUp constructs: Metadata types\textsuperscript{1} Folder and Document, and properties keywords, description, contentAttr, contentURL.

The mapping framework supplies the following default components:

- The “TeamUp” flex family, pre-configured to match the metadata type and properties listed above.

- A \texttt{mappings.xml} file, in which the metadata type and properties (listed above) are mapped to assets in the “TeamUp” flex family:
  - The Folder type is mapped to a flex parent definition asset named \texttt{tu_folder}.
  - The Document type is mapped to a flex definition asset named \texttt{tu_document}.
  - Attributes are mapped to flex assets of type “TeamUp Attribute.”

  The mappings are illustrated in Figure D-1, on page 85, coded in \texttt{mappings.xml}, on page 87, and summarized in Table D-1, on page 88.

Once the mapping is established, folders of type Folder (and their contents) can be published:

- Folders are published as flex parent assets to the “TeamUp Folder” asset type.
- Documents are published as flex assets to the “TeamUp Document” asset type.

Publishing to the TeamUp flex family is summarized below.

During publishing, the Content Integration Platform refers to the \texttt{mappings.xml} file to determine the types of content to publish (\texttt{tu_folder} and \texttt{tu_document}). The folder that is named in the \texttt{publish} command is the starting point of the publication process. The folder is published as a flex parent asset of type “TeamUp Folder,” along with all the subfolders and documents it contains.

To reproduce the folder’s structure, (subfolders and documents), the Content Integration Platform refers to path information:

- If subfolders exist, the Content Integration Platform chains their corresponding “TeamUp Folder” assets to reproduce the hierarchy.
- Documents, treated as “TeamUp Document” assets, are placed under their respective “TeamUp Folder” parent assets.

---

\textsuperscript{1} Metadata assigned to a top folder is inherited by its subfolders and documents.
Figure D-1: Mapping the Folder Type to Content Server’s “TeamUp” Flex Family

**TeamUp**

Content of type **Folder** published as **Assets of type “TeamUp Folder”**

**Published Content:**
- Folder: Minerals
  - Abundant
  - Gems
- Folder: Gems
- Folder: Minerals

**Folder Content Type:**
- Folder maps to `<AssetType>;<ParentDefinitionInstance>`

**Mapping:**
```
<assettype-mapping
  sourceid="Folder" targetid="TeamUp_Folder; tu_folder"
  id="tu_folder">
</assettype-mapping>
```

See “Default mappings.xml,” on page 87 for the code in its context. See also page 88 for flex family specifications.
Figure D-2: Mapping the Document Type to Content Server’s “TeamUp” Flex Family

TeamUp

Content of type Document \(\text{published as}\) Assets of type “TeamUp Document”

Content Server

Gems

- Emerald
- Sapphire

Document: Sapphire
Asset: Sapphire.png

Document: Emerald
Asset: Emerald.png

Published Content:

Document Content Type:

| <AssetType>;<ChildDefinitionInstance> |

Mapping:

```xml
<assettype-mapping
sourceid="Document" targetid="TeamUp_Document;tu_document"
id="tu_document">
</assettype-mapping>
```

See “Default mappings.xml,” on page 87 for the code in its context (including attributes that are associated with the Document metadata type). See also page 88 for flex family specifications.
Default mappings.xml

The default mappings.xml file, located in the Content Integration Agent, maps TeamUp schema to Content Server’s “TeamUp” flex family. The mapping is uniquely identified by the ID in line 2. Parameters for the <assettype-mapping> and <descriptor-mapping> tags are defined in “TeamUp to CS Mapping: Tags and Parameters,” on page 55.

1   <!-- TeamUp to CS -->
2   <mapping id="teamup2cs" MissingAssetTypeAction="pass"
            MissingDescriptorAction="skip">
3     <assettype-mapping
               sourceid="Folder:http://ns.infostoria.com/wikis/
                        RepositorySchema/topFolder_cipID/RepositoryItem"
               targetid="TeamUp_Folder;tu_folder"
               id="tu_folder">
4       <descriptor-mapping
               sourceid="keywords"
               targetid="tags" />
5       <descriptor-mapping
               sourceid="description"
               targetid="notes" />
6     </assettype-mapping>
7     <assettype-mapping
               sourceid="Document:http://ns.infostoria.com/wikis/
                        RepositorySchema/topFolder_cipID/RepositoryItem"
               targetid="TeamUp_Document;tu_document"
               id="tu_document">
8       <descriptor-mapping
               sourceid="keywords"
               targetid="tags" />
9       <descriptor-mapping
               sourceid="description"
               targetid="notes" />
10      <descriptor-mapping
               sourceid="contentAttr"
               targetid="file" type="static" />
11      <descriptor-mapping
               sourceid="contentURL"
               targetid="contentURL" type="dynamic" />
12     </assettype-mapping>
13   </mapping>
‘TeamUp’ Flex Family Specifications

Table D-1 summarizes the default mapping of TeamUp schema to Content Server’s “TeamUp” flex family. For customized implementations, you can either re-use the flex family or create your own.

Table D-1: TeamUp Default Data and Flex Family Analogs

<table>
<thead>
<tr>
<th>Type of Data</th>
<th>TeamUp Default Data</th>
<th>Maps To:</th>
<th>Flex Family Member</th>
<th>Description</th>
<th>Assets of This Type Are Created By ...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Schema</td>
<td>Property</td>
<td>TeamUp Attribute</td>
<td>Flex Attribute</td>
<td>This flex asset type stores attributes for documents.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td></td>
<td>Properties (for documents):</td>
<td>Stores attribute instances.(a)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• keywords</td>
<td>• tags</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• description</td>
<td>• notes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• contentAttr</td>
<td>• file</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• contentURL</td>
<td>• contentURL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata Type(^b)</td>
<td>Folder</td>
<td>TeamUp Parent Definition</td>
<td>Flex Parent Definition</td>
<td>This flex asset type stores parent definition instances.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td></td>
<td>Stores parent definition instance: tu_folder</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Metadata Type(^b)</td>
<td>Document</td>
<td>TeamUp Child Definition</td>
<td>Flex (Child) Definition</td>
<td>This flex asset type stores child definition instances.</td>
<td>CS Administrator</td>
</tr>
<tr>
<td></td>
<td>Stores child definition instance: tu_document</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Content</td>
<td>Published Folders</td>
<td>TeamUp Folder</td>
<td>Flex Parent</td>
<td>This flex asset type stores folder assets. For an example, see Figure D-1, on page 85.</td>
<td>Content Integration Platform</td>
</tr>
<tr>
<td></td>
<td>Folders of metadata type Folder</td>
<td>Stores flex parent assets (of type tu_folder and any other type)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Published Documents</td>
<td>Documents of metadata type Document</td>
<td>TeamUp Document</td>
<td>Flex (Child) Asset</td>
<td>This flex asset type stores document assets. For an example, see Figure D-2, on page 86.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Stores flex (child) assets (of type tu_document and any other type)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(a\) “Instance” means “asset.” “Instance” is used only to help differentiate metadata (instances) from published content (assets).

\(b\) Set of all metadata properties used for TeamUp top folders and inherited by subfolders and documents.