FatWire

# Content Integration Platform for EMC Documentum

Version 2.0

## Administrator's Guide

**Revision Date:** Mar. 7, 2011

FatWire® SOFTWARE

*FatWire Content Integration Platform for EMC Documentum Administrator’s Guide*
Revision Date: Mar. 7, 2011
Product Version 2.0

**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

# About This Guide

This guide contains procedures for installing, configuring, and using the FatWire Content Integration Platform to publish from EMC Documentum to FatWire Content Server and archive FatWire Content Server websites to EMC Documentum.

## Audience

Installation engineers must have experience configuring Content Server and its supporting enterprise-level software. CIP administrators must be expert users of FatWire Content Server and EMC Documentum. They must be thoroughly familiar with Content Server's flex asset model, the process of creating and configuring content management (CM) sites, and implementing RealTime publishing methods. CIP administrators must also have expertise creating Documentum object types and implementing workflows.

## Naming Conventions

- **CIP**. FatWire Content Integration Platform
- **Content Server**. "FatWire Content Server."
- **CS**. FatWire Content Server.
- **Content Integration Agent**. Also called "Integration Agent."
- **Content Server Agent Services**. Also called "Agent Services."

## Terms in This Guide

**archiving**. Storing Content Server websites to Documentum. See also *FatWire-to-Documentum integration*.

**assets**. Content Server assets, both basic and flex.

**CS DataStore.** CIP name for the hierarchical folder structure that stores Content Server websites to be archived on Documentum. The CS DataStore is created and exported during a RealTime publishing session that is enabled for archiving.

**FatWire to Documentum integration.** Also called *archiving*. The process of storing Content Server websites to Documentum. The stored websites can be assigned compliance and retention policies.

**items**. Generic term for Documentum objects and Content Server assets.

**mappings.xml.** The metadata map that enables Content Integration Platform to interpret Documentum objects as Content Server assets and vice versa.

**monitored workspace.** A published or archived workspace. The workspace is monitored by the CIP synchronization engine, which automatically replicates updated content based on previously replicated metadata (defined in `mappings.xml`).

**objects**. Documentum objects, such as cabinets, folders, and documents.

**`publish` command.** CIP command used to initialize the source and target workspaces. The `publish` command replicates the source system's specified workspace to the target system and initializes the synchronization engine, which then starts monitoring the source workspace. The `publish` command is used in both models, publishing and archiving.

**publishing.** Replicating Documentum objects to Content Server. Publishing does not produce a website. See also *archiving*.

**synchronization.** The process of keeping content on source and target systems in agreement. The synchronization interval can be configured. *See also* `publish` command.

**workspace.** An object that stores data to be published or archived. In the publishing model, the source workspace is a Documentum cabinet containing the folder that will be published; the target workspace is a Content Server flex family (`Documentum` by default). In the archival model, the source workspace is a CS DataStore; the target workspace is a Documentum folder.

Chapter 1

# Installing FatWire Content Integration Platform

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

This chapter contains the following sections:

## Installation Overview

FatWire Content Integration Platform enables bidirectional communication between FatWire Content Server and EMC Documentum:

- Publishing Documentum objects to FatWire Content Server
- Archiving FatWire Content Server websites on EMC Documentum

CIP supports publication and archiving by replicating the source system's metadata to the target system, thus providing the schema that stores the source system's content on the target system. CIP also provides a command-line interface for publishing and archiving via the CIP `publish` command. The synchronization engine subsequently monitors the source system's published workspaces for changes to content and automatically replicates the modified items to the target system.

You will install CIP components to support the operations of your choice. Installing Content Integration Agent is a requirement:

- To support publishing and archiving, you must install Content Integration Agent, which manages metadata and the associated content. Integration Agent includes:
  - `mappings.xml`, which maps the source system's metadata to the target system.
  - `cipcommander`, a command-line application that is used to run the CIP `publish` command. The same command publishes to Content Server (via Agent Services) and archives to Documentum.
  - synchronization engine, which monitors the source's published workspaces in order to keep the source and target in agreement.
  - `catalog.xml`, which stores information about published and archived items and serves as a configuration file for tuning the synchronization process.
- To enable publishing, you will install a programmatic publishing interface and database tables to store published objects, including their object type definitions:
  - The programmatic interface, named "Content Server Agent Services," receives Documentum data from Integration Agent and stores the data to Content Server's database tables. The data consists of metadata in CS-compliant format and objects associated with the metadata.
  - Database tables for storing published objects are provided in the form of the `Documentum` flex family, a data model that must be installed on the Content Server management system and enabled on one of its content management sites. By default, the `Documentum` flex family also contains definitions of the Documentum object types `dm_folder` and `dm_document`. Both object types are mapped in the default `mappings.xml` file.
- To enable archiving, you will install the `fw_asset` and `fw_document` object types on Documentum to provide the schema for storing Content Server assets as Documentum objects. The object types are mapped in `mappings.xml`.

## Prerequisites

- Download the *CIP Supported Platform Document* and release notes, available on our e-docs site, at `http://e-docs.fatwire.com`. The site is password-protected. Accounts with FatWire Technical Support can be opened from the home page.
  - Microsoft Visual C++ 2008 Redistributable (x86) can be downloaded from `http://www.microsoft.com`
  - OpenSSL can be downloaded from `http://www.openssl.org`
- To support publishing operations, create or select a CM site on the staging system for the `Documentum` flex family you will be installing.
- To support archiving operations:
  - The Content Server staging and delivery systems must be running the Web Experience Management (WEM) Framework.
  - If your delivery system is running FatWire Community Server and you wish to archive site visitors' comments and reviews, you will have to RealTime publish them from the delivery system to a separate Content Server system. The target Content Server must be running the WEM Framework.
- At the end of the installation process, you have the option to configure event notification. If you choose this option, you must configure event notification before publishing or archiving.

## Packaging

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

| File | Description |
|---|---|
| `cipagent-`*`vNo`*`.msi` (for Windows)<br>`cipagent-`*`vNo`*`.rpm.bin` (for Linux) | For publishing and archiving. These files install Content Integration Agent (`cipcommander` and service) and configuration files that control the Integration Agent's process.<br>This file must be installed on a machine that runs a supported operating system and can access both the source and target systems |
| `csagentservices.war` | For publishing. This file installs Content Server Agent Services, including property files for setting the detail of log files and regulating access to Content Server's database.<br>This file must be deployed on a system other than delivery. The system must have access to the Content Server `Shared` directory. |
| `cs_documentum_schema.zip` | For publishing. This file installs the "Documentum" flex family on FatWire Content Server, on the CM site that you specify. |
| `documentum_cs_schema.dar` | For archiving. This file installs schema on Documentum to support the content of the CS DataStore. |

# Installing FatWire Content Integration Platform

To ensure a smooth installation process, read the steps below to gain an understanding of the installation process and note the information you will need to provide.

## Step I. Install Content Integration Agent

Complete the following steps for all CIP integrations:

1. If you are using Windows, install Microsoft Visual C++ 2008 Redistributable Package (x86) on the machine that will host Content Integration Agent.
2. Run the `cipagent` file on a machine with a supported operating system and the ability to access the source and target systems.
   - **Windows:**

     Run `cipagent-2.0.0-127.msi` and follow the prompts.

     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-2.0.0-128.el5.i386.rpm.bin
     ```

     The following directories are installed:

     ```
     usr
       local
         bin
           cipagent -exe
           cipcommander
         lib
           cipagent
             ..all libraries...
     share
       cipagent
         conf
         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 Content Integration Agent requires in order to connect to the source system and FatWire Content Server. You will edit this file to provide Content Integration Agent with system location and user information.

   a. Open `catalog.xml` in a text editor.

   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. 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 context-related paths. Typically you need to modify only the host name (the default value is `http://localhost:9090`).
      - **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. Save `catalog.xml`.

5. Restart the Content Integration Agent executable:

   - **For Windows:** Restart the FatWire Content Integration Agent service.

- **For Linux:** Type as root user: **`/sbin/service restart cipagent`**

> **Note**
>
> The Content 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 Content 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 multiple network interfaces are 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 <port>`** to all commands that start `CIPCommander`.

6. In `facilities.xml`, enable the java facility section. Specify the path to Java on your system.
7. Continue to the next step, "A. Installing Content Server Agent Services."

## Step II. Install CIP Publishing Components

If you wish to install only archival capability, skip to "Step III. Install Schema to Support Archiving to Documentum," on page 14. Otherwise, complete all the steps in this section to install publishing capability.

### A. Installing Content Server Agent Services

> **Note**
>
> Content Server Agent Services can be installed on any Content Server system other than delivery. We recommend a content management (staging) system.

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.
4. Continue to the next step, "B. Installing Documentum Schema on Content Server."

## B. Installing Documentum Schema on Content Server

In this step, you will import `cs_documentum_schema.zip` into Content Server to support publication of Documentum objects to Content Server

### To install the Documentum schema

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

**Note**

Before using CatalogMover, 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
3. Log in to Content Server's Advanced interface as a general administrator (**fwadmin / xceladmin**, by default) and continue as follows:
   a. Enable the Documentum flex family on the content management site you have chosen or created, as explained in "Prerequisites," on page 9.

b. For quick access to published content, create a tree tab (named **Documentum**, for example).

For instructions on enabling flex families, creating sites, and creating tree tabs, see the *Content Server Administrator's Guide*.

4. Continue to the next step, "Step III. Install Schema to Support Archiving to Documentum."

## Step III. Install Schema to Support Archiving to Documentum

Install `documentum_cs_schema.dar` using Documentum Composer (see the *Documentum Composer User Guide* on how to install and use Documentum Composer).

```
CREATE TYPE fw_asset (
fw_id char(255),
fw_name char(64),
fw_createdby char(64),
fw_createddate time,
fw_description char(128),
fw_publist char(255),
fw_status char(2),
fw_subtype char(32),
fw_updatedby char(64),
fw_updateddate time,
fw_publisheddate time
) WITH SUPERTYPE dm_document

CREATE TYPE fw_document (
fw_publisheddate time,
fw_name char(64)
) WITH SUPERTYPE dm_document
```

## Step IV. Back Up the Default mappings.xml File

The `mappings.xml` file is located on the server that hosts Integration Agent.

# Next Step

Having installed CIP, you can proceed with one of the following steps:

- **Optional**. You can choose to configure event notification workflows to keep CIP administrators informed of actions that occur on the target system when changes are made to the source system's monitored workspaces (i.e., published or archived workspaces).

**Note**

Event notification workflows must be configured before any Documentum objects are published or any Content Server assets are archived. For background information and instructions on configuring event notification workflows, see Chapter 2, "Configuring Event Notification."

- To publish from Documentum to Content Server, skip to Chapter 3, "Publishing to FatWire Content Server" for background information and instructions.
- To archive Content Server assets to Documentum, skip to Chapter 4, "FatWire to Documentum Content Integration" for background information and instructions.

Chapter 2

# Configuring Event Notification

## Overview of Event Notification

CIP administrators typically require confirmation that source and target systems are synchronized. Event notification keeps them up to date. Event-related notices are delivered through a simple workflow process. Workflow processes can be enabled globally or selectively, for specific events and for specific types of items.

**Note**

If you choose to configure CIP for event notification, you must do so before objects are published to Content Server or assets are archived to Documentum. For instructions, see:

- "Configuring Event Notification Workflows for Content Server"
- "Configuring Event Notification Workflows for Documentum," on page 21

## Configuring Event Notification Workflows for Content Server

In the publishing model, event notification is the process of informing CIP administrators of actions that occur on Content Server when changes are made to monitored workspaces on Documentum. A Documentum workspace can be a cabinet or a folder.

Monitoring of a workspace begins once its metadata and associated objects are published to Content Server via the **`cipcommander publish`** command. The synchronization engine starts listening to the workspace for changes to content and automatically replicates them to Content Server, where they are treated as events. When event notification is configured, Content Server sends a message to CIP administrators confirming that the target workspace has been synchronized to the monitored workspace. Failed events also trigger event notices if their corresponding workflows are configured. Table 1 lists the default workflows.

Default workflows have the following properties:

- Default workflows consist of one state and two steps. When an event occurs, only the first step of the corresponding workflow is taken. If the option "Assign from list of participants" is chosen, all members with the selected roles are assigned the first task.
- All supported events trigger notices to users with role `CIPAdmin`. Tasks may or may not be assigned, depending on the event:
    - A task is assigned for the following events: creation, deletion failure, modification, and modification failure. The task is simply a way of notifying the users of events or their failure on Content Server. The task can be removed; there is no obligation to take a step.

- For deletion events, no task is assigned (the asset no longer exists).

**Note**

Although publishing-related workflows can be created from scratch, it is typically more convenient to use workflows that are packaged with CIP. If you wish to create your own workflows, refer to the *Content Server Administrator's Guide* for instructions.

**Table 1:** Default workflows for Content Server

| Event in Content Server | Workflow Process |
|---|---|
| Asset creation | `CIPAssetCreated.` Invoked when an object is created in a monitored cabinet or folder and the counterpart asset is created in Content Server. |
| Asset deletion | `CIPAssetDeleted`. Invoked when an object is deleted from a monitored cabinet or folder and the counterpart asset is deleted from Content Server. |
| Asset deletion failure | `CIPAssetDeletionFailed.` Invoked when: • An object that was deleted from the monitored cabinet or folder is checked out on the Content Server system. • An object that was deleted from the monitored cabinet or folder has dependencies that would become unresolved on the Content Server system if the counterpart asset were to be deleted. |
| Asset modification | `CIPAssetModified`, invoked when an object in the monitored cabinet or folder is modified and the counterpart asset is created in Content Server. |
| Asset modification failure | `CIPAssetModificationFailed`, invoked when an object in the monitored cabinet or folder is modified, but its counterpart asset is checked out in Content Server. |

## Installing Default Workflows on Content Server

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 default 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:

```
http://localhost:8080/cs/ContentServer?pagename=OpenMarket/
   Xcelerate/Installation/CIPCreateWorkflows&
   username=fwadmin&password=xceladmin
```

   When the workflows are created, the following message is displayed:

   "`Workflows for Content Integration Platform were created successfully`"

## Verifying Default Workflows

When the default workflows are created, associated items are also created in Content Server.

**To verify default workflows and their associated items**

1. Log in to the Content Server Advanced interface as a general administrator (default credentials: `fwadmin` / `xceladmin`).
2. Verify that the following items have been created:
   - `CIPAdmin` role, which will be used as the management role in all CIP workflows. All users with the `CIPAdmin` role will be notified of all CIP events in the default workflows.
   - Workflow processes:
     CIP Asset Created, CIP Asset Deleted, CIP Asset Deletion Failed,
     CIP Asset Modified, and CIP Asset ModificationFailed

- 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 Default Workflows

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

**To enable a CIP workflow**

1. Open `mappings.xml` (on the Integration Agent host), go to the section `<mapping id= "documentum2cs">` and uncomment the required workflows (below) for each asset type that must be enabled for event notification:

   ```
   <param name="assetCreatedProcess">CIPAssetCreated</param>
   <param name="assetModifiedProcess">CIPAssetModified</param>
   <param name="assetDeletedProcess">CIPAssetDeleted</param>
   <param name="assetDeletionFailedProcess">CIPAssetDeletionFailed
      </param>
   <param
      name="assetModificationFailedProcess">CIPAssetModification
      Failed</param>
   ```

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

# Configuring Event Notification Workflows for Documentum

In the archival model, event notification is the process of informing CIP administrators of actions that occur in Documentum target folders when changes are made to monitored CS DataStores.

Monitoring of a CS DataStore begins when its metadata and associated files are archived to Documentum via the **`cipcommander publish`** command. The synchronization engine starts listening for changes to the content of the CS DataStore and automatically replicates them to the Documentum target folders, where they are treated as events. When event notification is configured, Documentum sends a message to CIP administrators confirming that the target folders have been synchronized to the monitored CS DataStore.

**Note**

Event notification related to the archival process requires you to create or reuse Documentum workflows. Table 2 lists supported events.

**Table 2:** Event Notification Workflows for the Archival Process

| Event in Documentum | Workflow |
|---|---|
| Object creation | Invoked when a new asset is created in the CS DataStore and its counterpart object is created in Documentum. |
| Object modification | Invoked when an asset is modified in the CSDataStore and and its counterpart object is created in Documentum. |

#### To enable notification workflows for archival events

1. Create your own workflow processes in Documentum or re-use workflow processes. Define the packages as necessary.
2. Open `mappings.xml` and go to `<mapping id= “csds2documentum”>`.
   a. Uncomment the required workflow processes and packages (below) for each asset type that must be enabled for event notification:
   ```
   <param name="objectCreatedProcess">ProcessName</param>
   <param name="objectModifiedProcess">ProcessName</param>
   <param name="objectCreatedPackage">PackageName</param>
   <param name="objectModifiedPackage">PackageName</param>
   ```
   b. Replace *`ProcessName`* with the name of the workflow process that should be automatically started.
   c. *`PackageName`* is an optional parameter. If *`PackageName`* is not specified, an asset will be placed into the first available package.

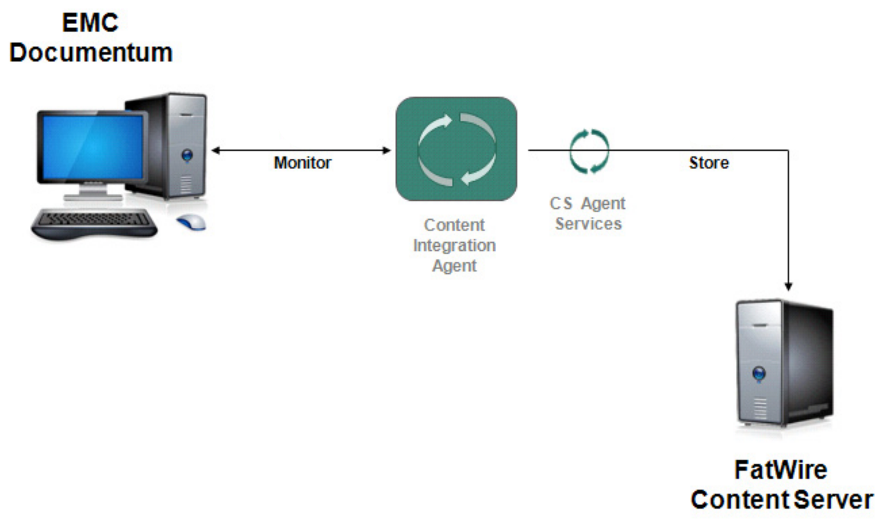
Chapter 3

# Publishing to FatWire Content Server

## Overview of Publishing

Publishing from Documentum to FatWire Content Server requires the CIP components shown in Figure 1. The components are Content Integration Agent and Content Server Agent Services.

**Figure 1: System Architecture for Publishing to FatWire Content Server**



## System Architecture and Process Flow

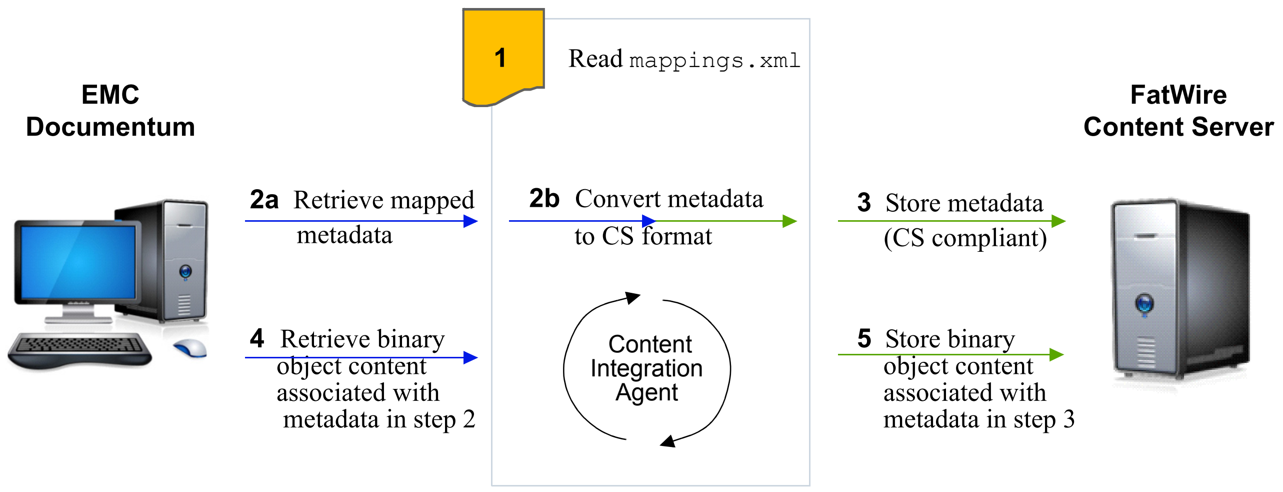
Content Integration Agent is used to synchronize the source and target workspaces via the `publish` command, the synchronization engine, and the `mappings.xml` file, which provides the metadata map. Content Server Agent Services exposes the web interface used by Integration Agent to perform the synchronization process. Following a publishing session, the synchronization process runs automatically. Details of the implementation are described below.

### Initial Synchronization

When the `publish` command is issued, the synchronization engine initializes the source and target by replicating metadata and associated content as outlined below (and in Figure 2, on page 25):

1. The synchronization engine refers to the source workspace (i.e., cabinet or folder) that is named in the `publish` command,
   a. reads the workspace’s objects to retrieve their mapped metadata,
   b. converts the metadata to CS-compliant format, using the `mappings.xml` file
2. stores the CS-compliant metadata to Content Server (via Agent Services),
3. retrieves from Documentum the content of binary objects associated with the metadata, and
4. stores the objects (via Agent Services) to the target flex family.

**Figure 2: Publishing Process**


1
Read mappings.xml
EMC Documentum
FatWire Content Server
2a Retrieve mapped metadata
2b Convert metadata to CS format
3 Store metadata (CS compliant)
4 Retrieve binary object content associated with metadata in step 2
Content Integration Agent
5 Store binary object content associated with metadata in step 3

### Monitoring the Data Source

Following the initial synchronization, the synchronization engine monitors the published cabinet or folder and automatically replicates changes to Content Server (via Agent Services). For example, when an object based on the published metadata is created, modified, or deleted in the monitored cabinet or folder, the synchronization engine replicates the new or modified object, or the object's deletion to Content Server. If metadata schema is modified, the workspace must be republished.

### Tuning the Integration

Content Integration Agent contains a configuration file named `catalog.xml`, which stores information about the publishing session. Various parameters in the file can be tuned once the synchronization process is initialized. When objects are "unpublished," their information is deleted from `catalog.xml`.

## Mapping Framework

The CIP mapping framework determines the success of the publishing and synchronization processes. Documentum objects can be published to Content Server as long as their metadata is mapped. The basic mapping framework for publishing involves two configurable components: the `Documentum` flex family and the `mappings.xml` file.

## Target Flex Family

The `Documentum` flex family, provided with CIP, stores published objects and their object type definitions as Content Server assets. Appendix A provides flex family specifications.

For simplicity, we recommend using the default flex family. Should you need to create your own flex family, refer to the *Content Server Administrator's Guide* for instructions and Appendix A as a model of the flex family.

## mappings.xml

The default `mappings.xml` file contains a `documentum2cs` section, which specifies the mappings listed below:

- The `dm_folder` type maps to `Documentum_Folder;dm_folder` in the `Documentum` flex family, where
  - `Documentum_Folder` is a flex parent asset type that stores folder assets.
  - `dm_folder` is a parent definition (of type `Documentum Parent Definition`) that defines the folder type.
- The `dm_document` type maps to `Documentum_Document;dm_document` in the `Documentum` flex family, where
  - `Documentum_Document` is a flex asset type that stores document assets.
  - `dm_document` is a child definition (of type `Documentum Child Definition`) that defines the document type.
- Attributes are mapped in the `<descriptor-mapping .../>` tags. Attributes are named as listed under the “Types” node of the Documentum WebTop interface:
  - `title`
  - `subject`
  - `keywords`
  - `r_version_label`
  - `r_full_content_size`

## Mapping, Publishing, and Synchronization

When publishing, bear in mind the following mapping specifications:

- Documentum objects that are based on default metadata can be published to Content Server without your having to modify either the default `mappings.xml` file or the `Documentum` flex family. Running the `publish` command replicates the specified workspace and its subfolders to the `Documentum` flex family as *flex parents*; documents are replicated as *flex assets*. The published workspace is then monitored by the synchronization engine.
- Publishing objects based on custom metadata (such as a new document type) requires you to first update at least the flex family with the new metadata. The `mappings.xml` file may or may not require updates, depending on the nature of the metadata. Details of custom mappings can be found in “Publishing via Customized Mappings,” on page 29.

## Publishing Procedures

- Publishing via the Default Mapping
- Publishing via Customized Mappings

### Publishing via the Default Mapping

In this section, you will publish cabinets and folders to Content Server using the default `mappings.xml` file and `Documentum` flex family.

**To publish**

1. Start Integration Agent.

**Note**

If you changed the port in step 5 on page 11 (starting Integration Agent), make sure that the new port is set in `facilities.xml`, and add **`-p <port>`** to the publish command in step 3, below (which starts `CIPCommander`).

2. Run the `CIPCommander` executable (located in the `bin` folder of the system where Integration Agent is installed).
3. Publish objects of the types that are specified in the default `mappings.xml` file (for definitions of publishing parameters, see Table 3, on page 28):

```
cipcommander
  publish <source_providerid> <target_providerid>
  -source_repname <cabinet_name>
  -source_path <path_in_cabinet>
  -target_repname <CS_content_management_site>
  -mapping <mapping_id>
  -replic_mode <full | new | updated>
  -bulk_resynch_interval <seconds>
```

**Examples:**

- To publish the `Images` cabinet to the "CIPDemo" content management site:

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

- To publish the `/Sample/Trees` folder in the `Images` cabinet to the "CIPDemo" content management site:

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

4. When the publishing session ends, the synchronization engine starts monitoring the published object.
   - Verify that modifications and deletions are replicated to Content Server (for example, modify the replicated objects, add folders and documents to the monitored object, and delete documents).
   - Objects can also be "unpublished." For information, see "Unpublishing," on page 30.
   - To optimize the synchronization process, see "Tuning the Synchronization Process," on page 30.

**Table 3:** Publishing Parameters

| Publishing Parameter | Required | Value |
|---|---|---|
| `<source_providerid>` | R | Provider ID for Documentum:<br>`d7a96a63-e78c-407c-8d7f-e84988806e49` |
| `<target_providerid>` | R | Provider ID for Content Server:<br>`70b1e307-26a1-499c-9295-cf0b6bd01342` |
| `-source_repname` | R | `<cabinet_name>`: Name of the cabinet containing the objects to be published. Enter the name exactly as it appears in the URL. |
| `-source_path` | | `<path_in_cabinet>`: Path to the object you want to publish.<br>• `/`<br>(to publish the cabinet specified by `<source_repname>`)<br>• `/<folder>/<folder>/... /<folder>`<br>(to publish the last folder in the path) |
| `-target_repname` | R | Name of the content management site (on FatWire Content Server) on which the target flex family is enabled. Enter the site's display name, exactly as it appears on the Admin tab in the Content Server Advanced interface. |
| `-mapping` | R | `<mapping_id>`: Value of the `mapping id` in `mappings.xml`. The value is `documentum2cs`. |
| `-replic_mode` | | `full \| new \| updated`<br>• `full` means that a full replication will be performed (by default), i.e., newly created items, updated items, and deletions.<br>• `new` means that only newly created items will be replicated (updates and deletions will not be replicated).<br>• `updated` means that only new and updated items will be replicated (deletions will not be replicated). |
| `-bulk_resynch_interval` | | `<seconds>`: Number of seconds between two successive synchronization events.<br>**Default value:** `600`<br>For optimal performance, set the synchronization interval to a value that agrees with the frequency of updates to the monitored folders. For more information, see "Tuning the Synchronization Process," on page 30. |

## Publishing via Customized Mappings

If the objects you plan to publish are based on unmapped metadata, you must first map the object types.

**To publish via customized mappings**

1. Depending on which type of metadata you have created, update the relevant mapping components as shown below. (We suggest reusing the `Documentum` flex family. The `mappings.xml` file is located on the Content Integration Agent host).

| New Metadata | Update | Guidelines |
|---|---|---|
| New folder type | `Documentum` flex family | Create a parent definition for the new folder type. (The default parent definition is `dm_folder`.) |
| | `mappings.xml` | Map the new folder type (`source id`). The `target id` takes the value `Documentum_Parent;<parent definition>` (where `Documentum_Parent` defines the storage table for folder assets). |
| New document type | `Documentum` flex family | Create a child definition for the new document type. (The default child definition is `dm_document`.) |
| | `mappings.xml` | Map the new document type (`source id`). The `target id` takes the value `Documentum_Child;<child definition>` (where `Documentum_Child` defines the storage table for document assets). |
| New document attribute | `Documentum` flex family | Add the new attribute to the Documentum flex family. |
| | Updating `mappings.xml` is conditional | Mapping the new attribute is required only if its name differs on the source and target. **Note:** Incorrect mapping of attributes does not stop the publication process, but it does produce a warning message and an entry in the log file. |

2. If you create flex filters, add the corresponding `jar` files to both the Content Server and the Content Server Agent Services applications.
3. Publish the objects. For instructions, see "Publishing Procedures," on page 27.

# Maintaining the Integrated Systems

- Tuning the Synchronization Process
- Unpublishing

## Tuning the Synchronization Process

When a cabinet or folder is published, `catalog.xml` is updated with data points from the `publish` command. The data points identify the Documentum and Content Server systems (in the `<workspace>` tags) and specify replication settings (in the `<replication>` tag).

Following a publishing session, the synchronization engine monitors the published cabinet (folder), using `catalog.xml`. The `BulkResynchInterval` and `ReplicMode` parameters can be reset as shown in Table 3, on page 28. The `catalog.xml` file is located in the `conf` folder on the Integration Agent server.

### Sample catalog.xml

```
<workspace id="41ce3f11-0411-46cb-b974-429162249462">
  <provider-ref refid="d7a96a63-e78c-407c-8d7f-e84988806e49" />
  <init-params>
    <param name="repname">Documents</param>
    <param name="path">/Images</param>
    <param name="repid">0c000001800045fe</param>
    <param name="itemid">0b0000018002c58e</param>
  </init-params>
</workspace>
<workspace id="6af59904-c6a3-4588-af7b-76f1422d6c10">
  <provider-ref refid="70b1e307-26a1-499c-9295-cf0b6bd01342" />
  <init-params>
    <param name="repname">FirstSiteII</param>
    <param name="repid">68ef906a-6c59-406a-84c2-b73b098cdb93</param>
  </init-params>
</workspace>
<replication>
  <link id="332e73c8-e977-4ba4-85c2-d7636281e192">
    <source-ref refid="41ce3f11-0411-46cb-b974-429162249462" />
    <target-ref refid="6af59904-c6a3-4588-af7b-76f1422d6c10" />
    <mapping-ref refid="documentum2cs" />
    <init-params>
      <param name="BulkResynchInterval">600</param>
      <param name="ReplicMode">full</param>
      <param name="IncrementalSyncDelay">10</param>
    </init-params>
  </link>
</replication>
```

## Unpublishing

You can unpublish objects from `catalog.xml` and Content Server by executing the **`cipcommander unpublish`** command. 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 30). The **`-delete`** parameter removes the same entries from Content Server's database.

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

```
cipcommander unpublish <parameters>
```

**Table 4:** Unpublish Parameters

| Unpublish Parameter | Description |
|---|---|
| `-all` | Clears `catalog.xml` of **all** publication entries. |
| `-linkid` | Clears `catalog.xml` of **selected** publication entries.<br>**`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 30).<br>For example, to unpublish an object with `linkid` `332e73c8-e977-4ba4-85c2-d7636281e192` from `catalog.xml`, run the following command:<br>**`cipcommander unpublish linkid`** `332e73c8-e977-4ba4-85c2-d7636281e192` |
| `-delete` | Removes from Content Server's database the same objects that you are unpublishing from `catalog.xml`.<br>**Legal values:** `<true \| false>`<br>**Default value:** `true` |

Chapter 4

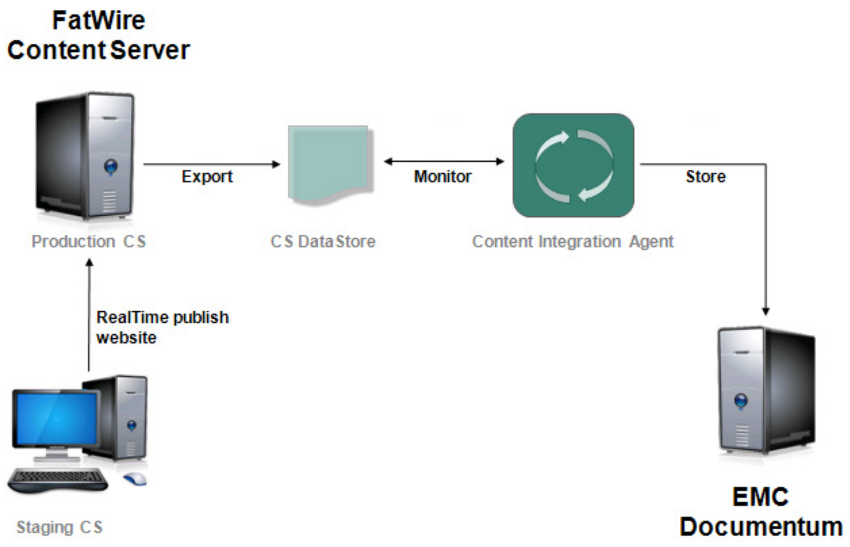
# FatWire to Documentum Content Integration

## Overview of the Integration Process

Archiving FatWire Content Server websites to EMC Documentum requires the CIP components shown in Figure 3. Users who are familiar with the process of publishing Documentum objects to Content Server will recognize archiving to be a similar process. The main differences are:

- In the archival process, Integration Agent treats the CS DataStore, rather than Content Server, as its source of data. The CS DataStore contains websites in folders and files that map to Documentum folders of type `dm_folder` and files of type `fw_document`. The CS DataStore is created when assets are published to a RealTime destination and archiving is enabled.
- Integration Agent archives to Documentum directly. It does not require the Agent Services component to store data on the target system.

**Figure 3: System Architecture for Archiving**



## System Architecture and Process Flow

Content Integration Agent synchronizes the CS DataStore and target Documentum folder via the `publish` command, the synchronization engine, and the `mappings.xml` file, which provides the metadata map. Following the initial archival session, the synchronization process runs automatically. Details of the implementation are described below.

### Initial Synchronization

Issuing the `publish` command invokes the synchronization engine to archive the CS DataStore to Documentum and thereby initialize the synchronization process:

1. The synchronization engine refers to the CS DataStore that is specified in the `publish` command,

   **a.** reads the files in the CS DataStore to retrieve their metadata,
   **b.** converts the metadata to a Documentum-compliant format, using `mappings.xml` and

**2.** stores the files to the target Documentum folder.

### Monitoring the CS DataSource

Following the initialization process, the synchronization engine monitors the archived CS DataStore and automatically replicates changes to the Documentum target folder. When an asset based on the archived metadata is created or modified in the monitored CS DataStore (during a RealTime publishing process), the synchronization engine replicates the new or modified asset to the target folder on Documentum. If metadata is modified, the `mappings.xml` file must be reconfigured and the CS DataStore republished.

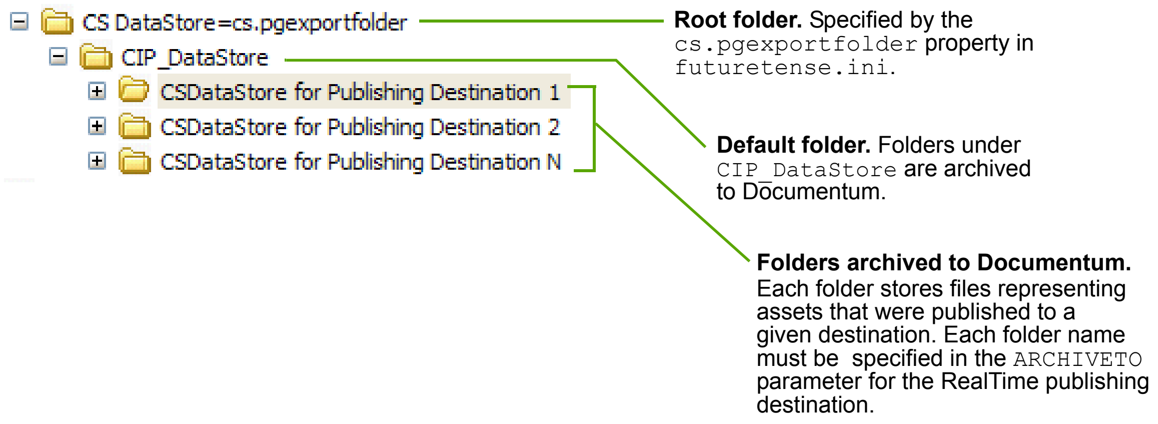
### Tuning the Integration

Content Integration Agent contains a configuration file named `catalog.xml`, which stores information about the archival session and allows tuning of the synchronization interval.

## CS DataStore

The CS DataStore contains websites in folders and files that map to Documentum folders of type `dm_folder` and files of type `fw_document`. A CS DataStore is created when assets are published to a RealTime destination with archiving enabled. The export path is:

```
<CS DataStore = cs.pgexportfolder>/CIP_DataStore/<CS DataStore for
   Publishing Destination>
```


CS DataStore=cs.pgexportfolder
CIP_DataStore
CSDataStore for Publishing Destination 1
CSDataStore for Publishing Destination 2
CSDataStore for Publishing Destination N

**Root folder.** Specified by the `cs.pgexportfolder` property in `futuretense.ini`.

**Default folder.** Folders under `CIP_DataStore` are archived to Documentum.

**Folders archived to Documentum.** Each folder stores files representing assets that were published to a given destination. Each folder name must be specified in the `ARCHIVETO` parameter for the RealTime publishing destination.

Figure 4 illustrates the archival of a sample CS DataStore.

**Figure 4:** Sample CS DataStore Archived to Documentum

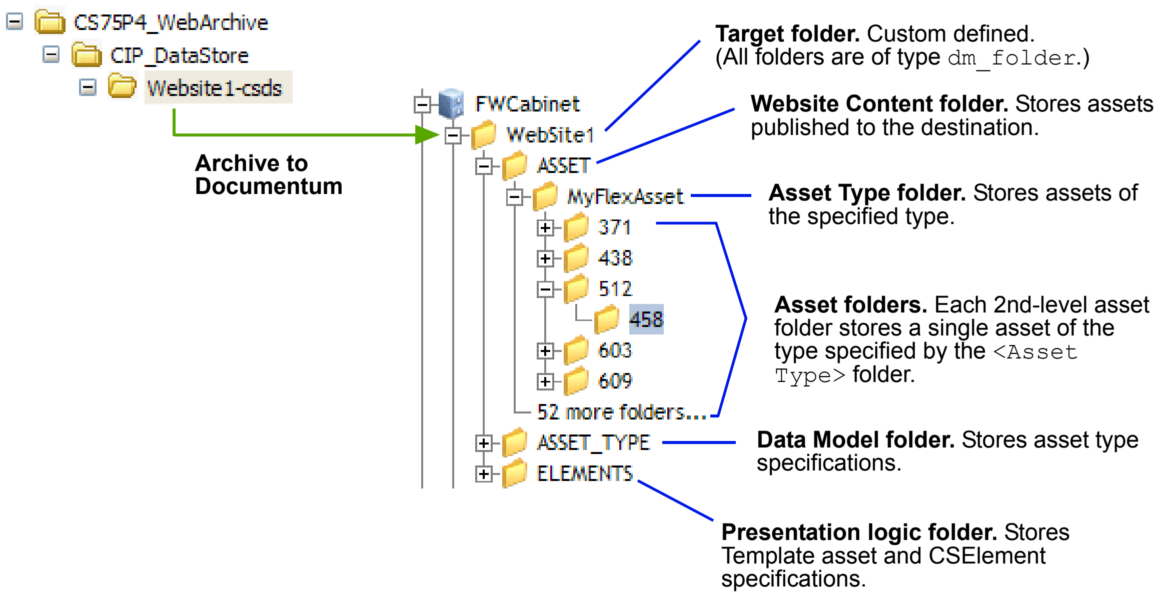
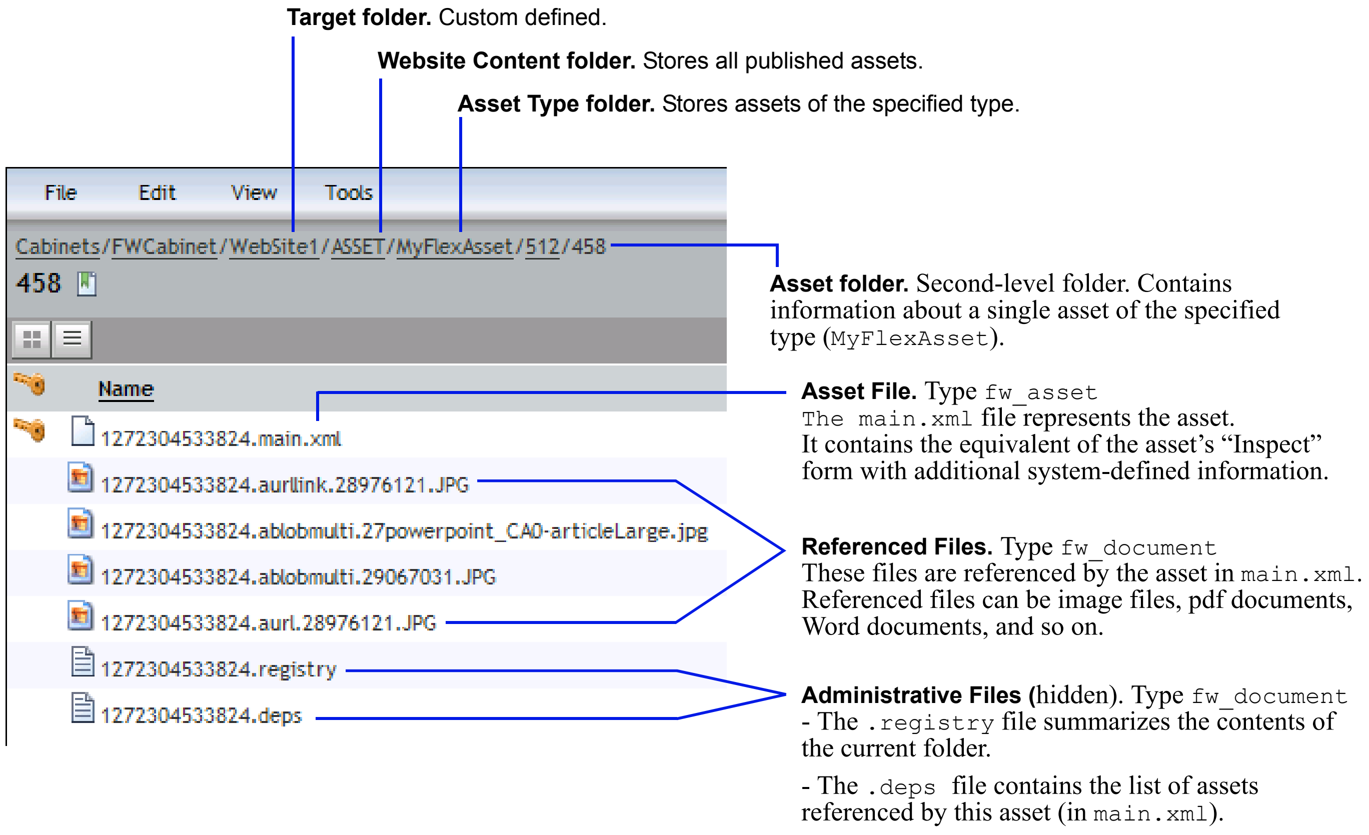

CS75P4_WebArchive
CIP_DataStore
Website1-csds
Archive to Documentum
FWCabinet
WebSite1
ASSET
MyFlexAsset
371
438
512
458
603
609
52 more folders...
ASSET_TYPE
ELEMENTS
Target folder. Custom defined. (All folders are of type dm_folder.)
Website Content folder. Stores assets published to the destination.
Asset Type folder. Stores assets of the specified type.
Asset folders. Each 2nd-level asset folder stores a single asset of the type specified by the <Asset Type> folder.
Data Model folder. Stores asset type specifications.
Presentation logic folder. Stores Template asset and CSElement specifications.

Figure 5 shows the contents of a typical asset folder (the same folder, named **458**, is also shown in Figure 4). **Note that the administrative files shown in Figure 5 are hidden files. They are shown here only for illustration purposes.**

**Figure 5:** Sample Asset Folder


Target folder. Custom defined.
Website Content folder. Stores all published assets.
Asset Type folder. Stores assets of the specified type.
File
Edit
View
Tools
Cabinets/FWCabinet/WebSite1/ASSET/MyFlexAsset/512/458
458
Name
1272304533824.main.xml
1272304533824.aurllink.28976121.JPG
1272304533824.ablobmulti.27powerpoint_CA0-articleLarge.jpg
1272304533824.ablobmulti.29067031.JPG
1272304533824.aurl.28976121.JPG
1272304533824.registry
1272304533824.deps
Asset folder. Second-level folder. Contains information about a single asset of the specified type (MyFlexAsset).
Asset File. Type fw_asset The main.xml file represents the asset. It contains the equivalent of the asset's "Inspect" form with additional system-defined information.
Referenced Files. Type fw_document These files are referenced by the asset in main.xml. Referenced files can be image files, pdf documents, Word documents, and so on.
Administrative Files (hidden). Type fw_document - The .registry file summarizes the contents of the current folder. - The .deps file contains the list of assets referenced by this asset (in main.xml).

## Mapping Framework

The CIP mapping framework determines the success of the archival and synchronization processes. A CS DataStore can be archived to Documentum as long as its metadata is mapped. The basic mapping framework involves two configurable components: object types in the Documentum workspace and the `mappings.xml` file.

### Object Types in the Documentum Workspace

Default Documentum target types are `fw_asset` and `fw_documentum`, which could be modified or replaced with custom types.

### mappings.xml

The default `mappings.xml` file contains a `documentum2cs` section, which specifies the mappings listed below:

- `csds_Folder` is the data type for all folders in the CS DataStore. Folders of type `csds_Folder` map to Documentum folders of type `dm_folder`:

```
<assettype-mapping
    sourceid="csds_Folder" targetid="dm_folder"
    id="Folder" extends="Item" />
```

- `csds_Document` is the data type for all files (except `main.xml`) in the CS DataStore. Files of type `csds_Document` map to Documentum documents of type `fw_document`.

```
<assettype-mapping
    sourceid="csds_Document" targetid="fw_document"
    id="Document" extends="Item" />
```

- `csds_Asset` is the data type of the `main.xml` file, which defines the asset (see Figure 5, on page 36). Files of type `csds_Asset` map to Documentum documents of type `fw_asset`.

```
<assettype-mapping
    sourceid="csds_Asset" targetid="fw_asset"
    id="Asset" extends="Document">
```

- Attributes of the `csds_Asset` document type map to Documentum attributes as shown. All Content Server attributes are system-defined.

| source id | target id | source id | target id |
|---|---|---|---|
| `id` | `fw_id` | `publist` | `fw_publist` |
| `name` | `fw_name` | `status` | `fw_status` |
| `createdby` | `fw_createdby` | `subtype` | `fw_subtype` |
| `createddate` | `fw_createddate` | `updatedby` | `fw_updatedby` |
| `description` | `fw_description` | `updateddate` | `fw_updateddate` |

- The `datemodified` attribute is a special attribute that stores the last publication date. The  date is taken from the last modification time of the corresponding asset file in the CS DataStore. The `datemodified` attribute maps to the Documentum attribute `fw_publisheddate`.

### Mappings, Publishing, and Synchronization

When archiving, bear in mind the following mapping specifications:

- **Default mapping.** Any CS DataStore can be archived to Documentum without your having to modify the default `mappings.xml` file. Running the `publish` command archives the CS DataStore to the target Documentum folder. The CS DataStore is then monitored by the synchronization engine. When RealTime publishing sessions update the CS DataStore, the new assets and modified assets are automatically replicated to the target Documentum folder by the synchronization engine.
- **Custom mapping.** You can map selected asset types and definitions to your own Documentum object type. Instructions are available in "Step 3. Add metadata to mappings.xml," on page 39.

## FatWire to Documentum Content Integration Procedures

### Step 1. Prepare the Documentum System to Store the Website

1. Create a cabinet for Content Server websites (`FWCabinet`, for example).
2. Create a folder for the given website (`WebSite1`, for example).
3. If you configured event notification, the associated workflow processes can be stored anywhere on the Documentum system.

### Step 2. Configure the Path to the CS DataStore

In this step, you will enable Content Server's RealTime publishing system to export your selected site in a CS DataStore along the following path (see also "CS DataStore," on page 35):

```
<CS DataStore=cs.pgexportfolder>/CIP_DataStore/<CSDataStore for
   Publishing Destination>
```

where:

- `<cs.pgexportfolder>` defines the root directory of the CS DataStore. (The `cs.pgexportfolder` property is located in the `futuretense.ini` file on the Content Server delivery system.)
- `CIP_DataStore` is a default subdirectory of `<cs.pgexportfolder>`. Its subfolders will be archived on Documentum.
- `<CSDataStore for Publishing Destination>` is a subfolder that holds the assets of the published site.

**To configure the path to the CS DataStore**

1. Configure the root directory of the CS DataStore by setting the `cs.pgexportfolder` property in the `futuretense.ini` file of the delivery system.
2. Configure the RealTime publishing process to support archiving.
   a. Configure publishing as shown in the *FatWire Content Server Administrator's Guide*. If RealTime publishing is already configured, start with "Creating a RealTime Destination Definition on the Source System."
   b. In the "Add New Destination" form, set the "More Arguments" field as follows:

   ```
   ARCHIVETO=<CSDataStore for Publishing Destination>
   ```

   **Note**

   If you are publishing a small number of assets (less than thousands) and wish to store their files to the same folder in the CS DataStore, specify `USEHASHDIRS=false` to disable hash folders.

3. Complete the remaining steps up to and including mirroring site configuration data to the destination database.

## Step 3. Add metadata to mappings.xml

To add custom mappings to `mappings.xml`, include the following information, depending on whether you are mapping a flex or basic asset type:

For flex asset types, `sourceid` takes the form `<asset type>;<Definition>`. For basic asset types, `sourceid` takes the form `<asset type>`. The `targetid` specifies the corresponding Documentum object type.

For example:

- To map all flex assets of type `Content_C` with asset definition `FSII Article` to the `fw_content` object type, add the following line to `mappings.xml`:

  ```
  <assettype-mapping sourceid="Content_C;FSII Article"
     targetid="fw_content" id="Content">
  </assettype-mapping>
  ```

- To map all basic assets of type `FW_Article` to the `fw_article` object type, add the following line to `mappings.xml`:

  ```
  <assettype-mapping sourceid="FW_Article"
     targetid="fw_article" id="Article">
  </assettype-mapping>
  ```

## Step 4. RealTime Publish the Site

Publish the content management site and verify that the CS DataStore was created in the specified path (in "Step 2. Configure the Path to the CS DataStore").

## Step 5. Archive the CS DataStore on Documentum

In this step, you will run the CIP `publish` command to archive the `<CS DataStore for Publishing Destination>` and initialize the synchronization process.

**Note**

If you changed the port in step 5 on page 11 (starting Integration Agent), make sure that the new port is set in `facilities.xml`, and add **`-p <port>`** to the command in step 2, below (which starts `CIPCommander`).

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

```
cipcommander
   publish <source_providerid> <target_providerid>
   -source_repid <path to data store>
   -target_repname <cabinet name>
   -target_path <path within the cabinet>
   -mapping <mapping_id>
   -bulk_resynch_interval <seconds>
   -handlerset csdatastore
   -create false
   -replic_mode updated
```

**For example:**

```
cipcommander
  publish 7833d862-4f8b-4285-84f2-731d5af81865 d7a96a63-
  e78c-407c-8d7f-e84988806e49
  -source_repid c:\temp\csdatastore
  -target_repname Archive
  -target_path /fatwire
  -mapping csds2documentum
  -bulk_resynch_interval 60
  -handlerset csdatastore
  -create false
  -replic_mode updated
```

**Table 5:** Publishing Parameters

| Publishing Parameter | Value |
|---|---|
| `<source_providerid>` | provider ID for the Content Server system:<br>`7833d862-4f8b-4285-84f2-731d5af81865` |

**Table 5:** Publishing Parameters

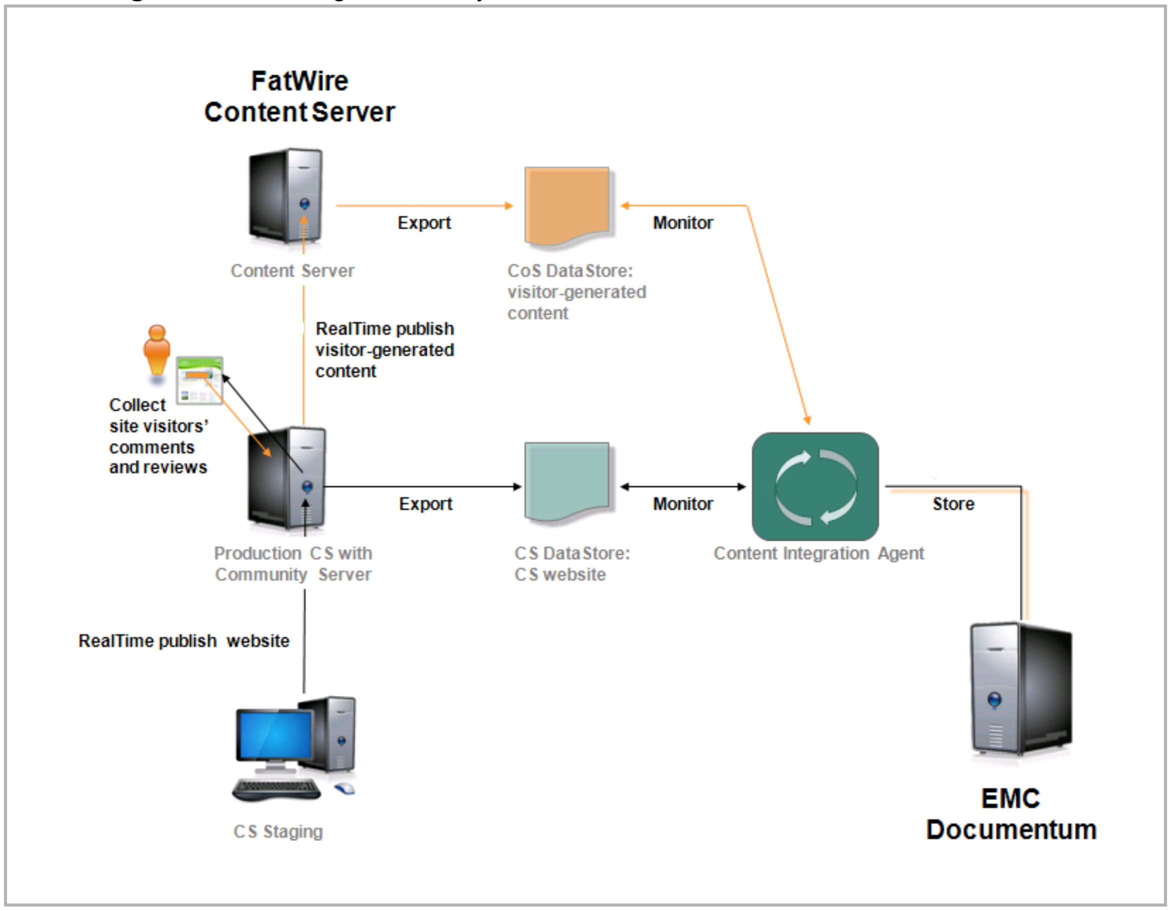
| Publishing Parameter | Value |
|---|---|
| `<target_providerid>` | provider ID for the Documentum system: `d7a96a63-e78c-407c-8d7f-e84988806e49` |
| `-source_repid` | `<path to data store>`: Path to the `<CS DataStore=cs.pgexportfolder>` folder on the file system. |
| `-target_repname` | `<cabinet name>`: Name of the Documentum cabinet to which `<CSDataStore for Publishing Destination>` will be archived. |
| `-target_path` | `<path within the cabinet>`: Path to the Documentum folder in the cabinet specified by `target_repname`. To publish to the cabinet itself, skip this parameter. |
| `-mapping` | `<mapping_id>`: mapping identifier from the `mappings.xml` file. **Default value**: `csds2documentum` |
| `-bulk_resynch_interval` | `<seconds>`: Number of seconds between two successive synchronization events. For optimal performance, set this value to a number that correlates with the frequency of RealTime publishing sessions. **Default value:** `600` |
| `-handlerset` | References the handlerset element from `handlers.xml`. **Allowed value:** `csdatastore` |
| `-create` | Specifies whether to create target repository. **Allowed value:** `false` |
| `-replic_mode` | Specifies which types of changes will be replicated. **Allowed value:** `updated` (only new and updated items will be replicated; deletions will not be replicated). |

3. Verify on Documentum the directory structure of the archived website. For background information, see "CS DataStore," on page 35.

## Step 6. Archive Visitor-Generated Content

If your Content Server delivery system runs FatWire Community Server and you wish to archive its visitor-generated content (comments and reviews), publish the content from the delivery system to a RealTime destination (see Figure 6, on page 42). Procedures for archiving visitor-generated content are identical to those for archiving websites.

**Figure 6:** Archiving Community Server's Visitor-Generated Content


FatWire Content Server
Content Server
Export
CoS Data Store: visitor-generated content
Monitor
RealTime publish visitor-generated content
Collect site visitors' comments and reviews
Production CS with Community Server
Export
CS Data Store: CS website
Monitor
Content Integration Agent
Store
RealTime publish website
CS Staging
EMC Documentum

## Testing Synchronization

After the `publish` command executes, the session ends and the synchronization engine starts monitoring the archived CS DataStore. Verify that modifications are replicated to Documentum:

1. Create and modify assets on the published content management site.
2. Republish the site to export the same `<CS DataStore for Publishing Destination>` folder.
3. Look for updates in the target Documentum folder.

## Tuning the Synchronization Process

When the `publish` command executes, the CS DataStore is archived and `catalog.xml` is updated with data points from the `publish` command. The data points identify the CS DataStore and Documentum system (in the `<workspace>` tags) and specify replication settings for the CS DataStore (in the `<replication>` tag).

Following the archival session, the synchronization engine starts monitoring the published CS DataStore. The synchronization interval can be reset in `catalog.xml`. See `BulkResynchInterval` in Table 5, on page 40. The `catalog.xml` file is located in the `conf` folder.

### Sample catalog.xml

```
<workspace id="776a0536-1af8-4e10-9b55-ecb9cfd715b8">
  <provider-ref refid="7833d862-4f8b-4285-84f2-731d5af81865" />
  <init-params>
    <param name="repid">C:\cs\export\CIP_datastore\DCTM</param>
    <param name="repname"></param>
  </init-params>
</workspace>
<workspace id="ae679aa2-572c-492a-b553-b7a866b60a2f">
  <provider-ref refid="d7a96a63-e78c-407c-8d7f-e84988806e49" />
  <init-params>
    <param name="repname">Archive</param>
    <param name="path">/FirstSiteII</param>
    <param name="repid">0c0000018002bd87</param>
    <param name="itemid">0b0000018002bd91</param>
  </init-params>
</workspace>
<replication>
  <link id="b5be4bc4-a8dc-4928-b3df-e0ac2851f4e4">
    <source-ref refid="776a0536-1af8-4e10-9b55-ecb9cfd715b8" />
    <target-ref refid="ae679aa2-572c-492a-b553-b7a866b60a2f" />
    <mapping-ref refid="csds2documentum" />
    <handlerset-ref refid="csdatastore" />
    <init-params>
      <param name="BulkResynchInterval">3</param>
      <param name="ReplicMode">updated</param>
      <param name="IncrementalSyncDelay">10</param>
    </init-params>
  </link>
</replication>
```

Appendix A

# Default Mapping Specifications for Publishing to Content Server

This appendix contains the following sections:

- Mapping Framework
- 'Documentum' Flex Family Specifications

## Mapping Framework

The default mapping framework for the publishing model supplies the following components (also listed in Table A-1):

- The “Documentum” flex family, pre-configured to match the object types and attributes listed above.
- A `mappings.xml` file, in which the 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.”

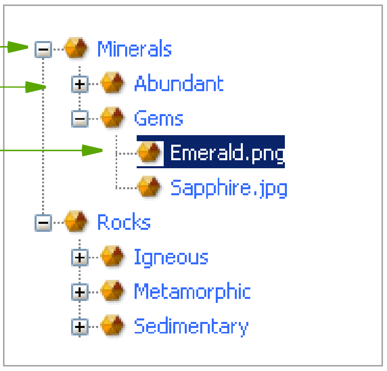
When the `publish` command is issued:

- 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.

During publishing, Content Integration Agent refers to the `mappings.xml` file to determine the types of objects to publish. 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), Content Integration Agent refers to path information. If subfolders exist, Content Integration Agent chains the counterpart “Documentum Folder” assets to reproduce the hierarchy.

Documents are treated as flex assets of type “Documentum Document.” They are placed under their respective “Documentum Folder” parents.



## 'Documentum' Flex Family Specifications

Table A-1 summarizes the mapping of default object types in Documentum to asset types and assets in Content Server's `Documentum` flex family. In customized implementations, you can either re-use the flex family or create your own.

**Table A-1:** Documentum Default Data and Flex Family Analogs

| **Documentum** | **Content Server: "Documentum" Flex Family** | |
|---|---|---|
| **Object Type** | **Flex Asset Type** | **Assets** |
| **Document Attribute**[a]<br>• **`title`**<br>• **`subject`**<br>• **`keywords`**<br>• `r_version_label`<br>• `r_full_content_size` | **Documentum Attribute**<br>Stores document attributes. | • `title`<br>• `subject`<br>• `keywords`<br>• `version label`<br>• `file size` |
| **Folder**<br>`dm_folder` (default) | **Documentum Parent Definition**<br>Stores folder type definitions. | **Parent definitions**<br>**`dm_folder`** (default) |
| | **`Documentum_Folder`**<br>Flex Parent asset type;<br>stores folder assets. | Documentum folders |
| **Document**<br>`dm_document` (default) | **Documentum Child Definition**<br>Stores document type definitions. | **Document definitions**<br>**`dm_document` (default)** |
| | **Documentum_Document**<br>Flex (Child) asset type;<br>stores document assets. | Documentum documents |
| **default mappings.xml:**<br>**<assettype mapping>** | | |
| **sourceid** | **targetid** | |
| `dm_folder` | `Documentum_Folder;<dm_folder>` | |
| **`dm_document`** | `Documentum_Document;<dm_document>` | |

a. Attribute names are those listed in the "Types" node of the Documentum WebTop interface.