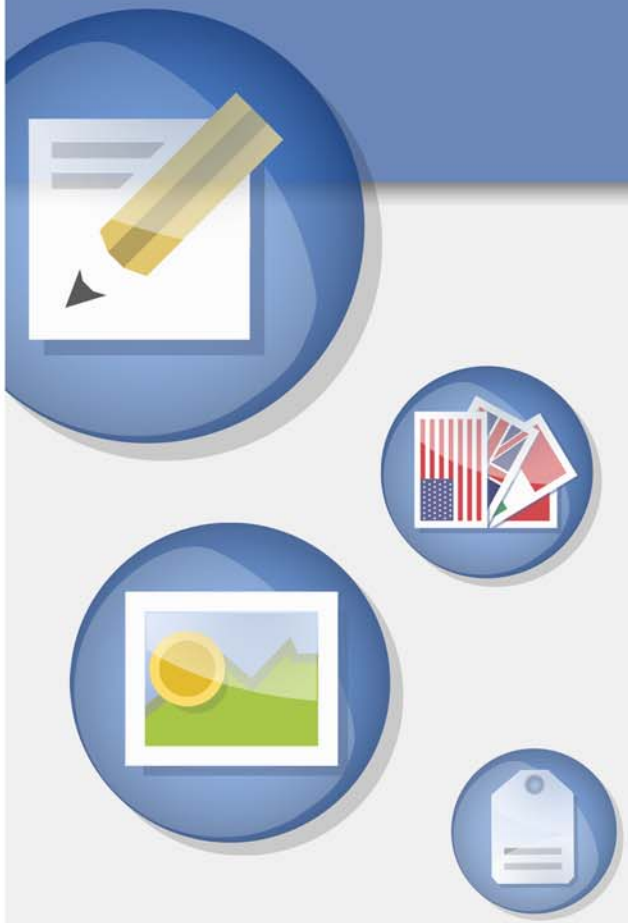
FatWire | Content Server 7

Version 7.6 Patch 2



Guide to Content Server Developer Tools

Document Publication Date: Jan. 31, 2012

FatWire® SOFTWARE

FatWire Content Server: Guide to Content Server Developer Tools
Document Publication Date: Jan. 31, 2012
Product Version: 7.6 Patch 2

FatWire Headquarters
FatWire Corporation
330 Old Country Road
Suite 303
Mineola, NY 11501

Table of

Contents

Appendices

About This Guide

This guide provides information about developing FatWire Content Server sites using FatWire Content Server Developer Tools, which enable working in a distributed environment.

Who Should Use This Guide

This guide is intended for developers who are familiar with FatWire Content Server, its data models, and the process of writing templates as well as other code.

Related Documents

See the following documents in the FatWire documentation set:

- *FatWire Content Server Developer's Guide*
- *FatWire Content Server Administrator's Guide*
- *FatWire Content Server Advanced User's Guide*
- *FatWire Content Server Javadoc*

Conventions

The following text conventions are used in this guide:

- **Boldface** type indicates graphical user interface elements that you select.
- *Italic* type indicates book titles, emphasis, or variables for which you supply particular values.
- `Monospace` type indicates file names, URLs, sample code, or text that appears on the screen.
- **`Monospace bold`** type indicates a command.

Third-Party Libraries

FatWire Content Server and its applications include third-party libraries. For additional information, see *FatWire Content Server 7.6 Patch 2: Third-Party Licenses*.

Chapter 1

About FatWire Content Server Developer Tools

This chapter provides an overview of the FatWire Content Server Developer Tools (CSDT).

This chapter contains the following topics:

Introduction

This guide provides information and instructions about developing FatWire Content Server sites using FatWire Content Server Developer Tools (CSDT). CSDT enables developers to work in a distributed environment using tools such as the Eclipse Integrated Development Environment (IDE) and version control system (VCS) integration. CSDT does not interfere or integrate with other development models. Using CSDT, a development team can manage FatWire Content Server resources and exchange those resources with other members of the team.

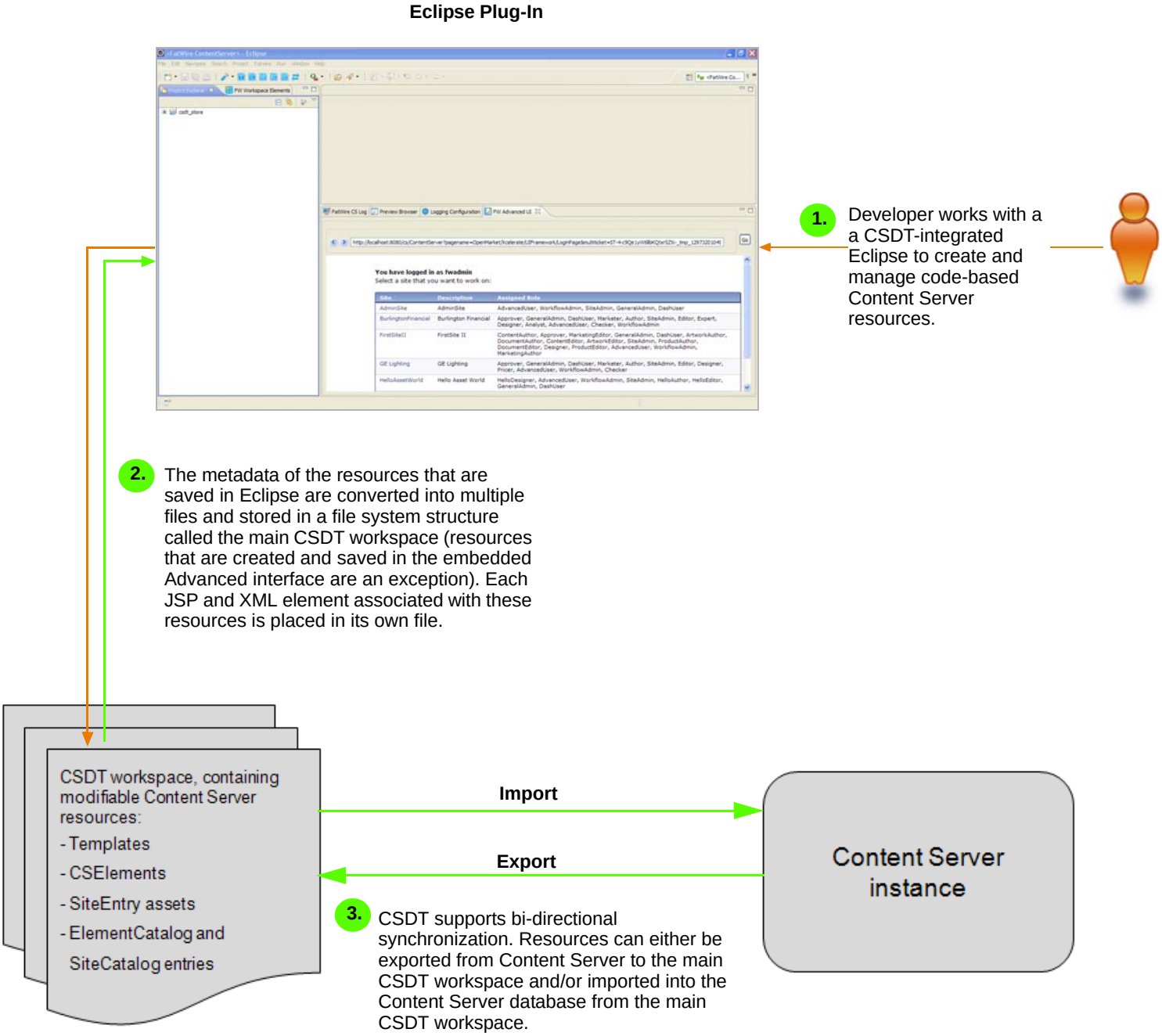
CSDT Architecture

On any computer running Content Server 7.6 with the FatWire Web Experience Management (WEM) Framework installed, CSDT can be used to integrate Content Server with the Eclipse IDE as shown in Figure 1, on page 11 and thus create a personal and flexible developer's environment:

- The developer interacts with CSDT (and therefore Content Server), primarily through Eclipse, which upon integration provides a rich set of Content Server-specific tools for managing assets and other types of Content Server resources.
- CSDT enables synchronization of resource development in Eclipse with resource development in Content Server, and vice versa.

Although simple in concept and design, CSDT has many important implications. For example, IDE-managed resources are stored as files in a file system, giving developers the option to integrate with a version control system of their choice. At the same time, the files are automatically converted to Content Server's native asset representation and imported into Content Server's database. Synchronization can be performed in two directions, enabling developers to work either in Eclipse or directly in Content Server. More information about CSDT capabilities can be found in the rest of this chapter.

Figure 1: CSDT Process Flow



IDE Integration

Using the Eclipse integration, developers can:

- Create, edit, and delete CSElement, Template, and SiteEntry assets, as well as SiteCatalog and ElementCatalog entries
- Develop JSP elements with standard Eclipse features such as tag completion, syntax highlighting, debugging, and so on
- Export and import assets, asset types, flex families, sites, roles, tree tabs, and start menu items
- Preview Content Server pages within the Eclipse IDE using an embedded preview browser
- View the Content Server log file in a dynamically refreshing panel
- Leverage existing Eclipse capabilities for integration with version control systems, given the file system representation of Content Server resources

Note

When integrated with CSDT, Eclipse provides an embedded Content Server Advanced interface, used to manage all types of Content Server resources. **The embedded Advanced interface is not covered in this guide**. Content Server resources created in the Advanced interface are not stored in a file system structure; they are stored directly in the Content Server database. For information about using the Advanced interface, see the *Content Server Developer's Guide*, *Administrator's Guide*, and *Advanced User's Guide*.

The CSDT Workspace

Content Server resources that are managed in Eclipse are stored as files in a file system structure called the *main CSDT workspace*. This enables resources to be easily managed and optionally exchanged with other Content Server instances. The main CSDT workspace is the only workspace accessible from Eclipse.

Note

Advanced developers can use the command-line tool to create any number of custom CSDT workspaces for a Content Server instance. Custom workspaces are not accessible from Eclipse. Creating a custom workspace is optional, and in most distributed environments the only necessary workspace is the main CSDT workspace. For more information, see Chapter 6, "Workspaces."

In this guide, any mention of the CSDT workspace refers to the main CSDT workspace. Custom CSDT workspaces are explicitly identified.

Synchronization

CSDT enables you to synchronize resource development in the Eclipse IDE with resource development in Content Server, ensuring that the CSDT workspace and Content Server database are populated with the same content. Manual synchronization is bi-directional, meaning you can import resources into Content Server and export resources to the CSDT workspace.

Any resource developed in the Eclipse IDE is stored as a single file or multiple interrelated files in the CSDT workspace. When you import a resource into Content Server, CSDT converts the resource to native Content Server format (database representation) and stores the resource in Content Server's database. When you export resources that are developed directly in Content Server to the Eclipse IDE, those resources are converted into files by CSDT and stored in the CSDT workspace.

Note

Automatic synchronization occurs when Content Server resources are edited, created, or deleted in Eclipse. All changes are automatically synchronized with the Eclipse-integrated Content Server instance and stored in the native database structure. Synchronization to Content Server is automatic only when the Eclipse-integrated Content Server instance is running.

JSP Management

CSDT exposes JSPs at a well-known location in the CSDT workspace. This enables developers to write and debug JSPs by working directly with the files. This way managing (creating, editing, debugging) Content Server JSPs in Eclipse is the same as working with any other JSP files. CSDT automatically synchronizes the files stored in the CSDT workspace with Content Server. This synchronization also includes transparent flushing of page and resultset caches in Content Server.

In addition, CSDT manages modifications made to all other files by automatically synchronizing those changes to Content Server. However, if you modify Content Server resources without using Eclipse or modify resources directly in Content Server, manual synchronization is required.

Command-line Tool

CSDT provides a command-line utility which can be used for automation, deployment, and certain development activities. The command-line tool is an export/import feature intended for large-scale resource movement. However, unlike the Eclipse integration which enables you to work only with Content Server resources that are exported to the main CSDT workspace, the command-line tool enables you to work with Content Server resources stored in any workspace.

Using a Version Control System

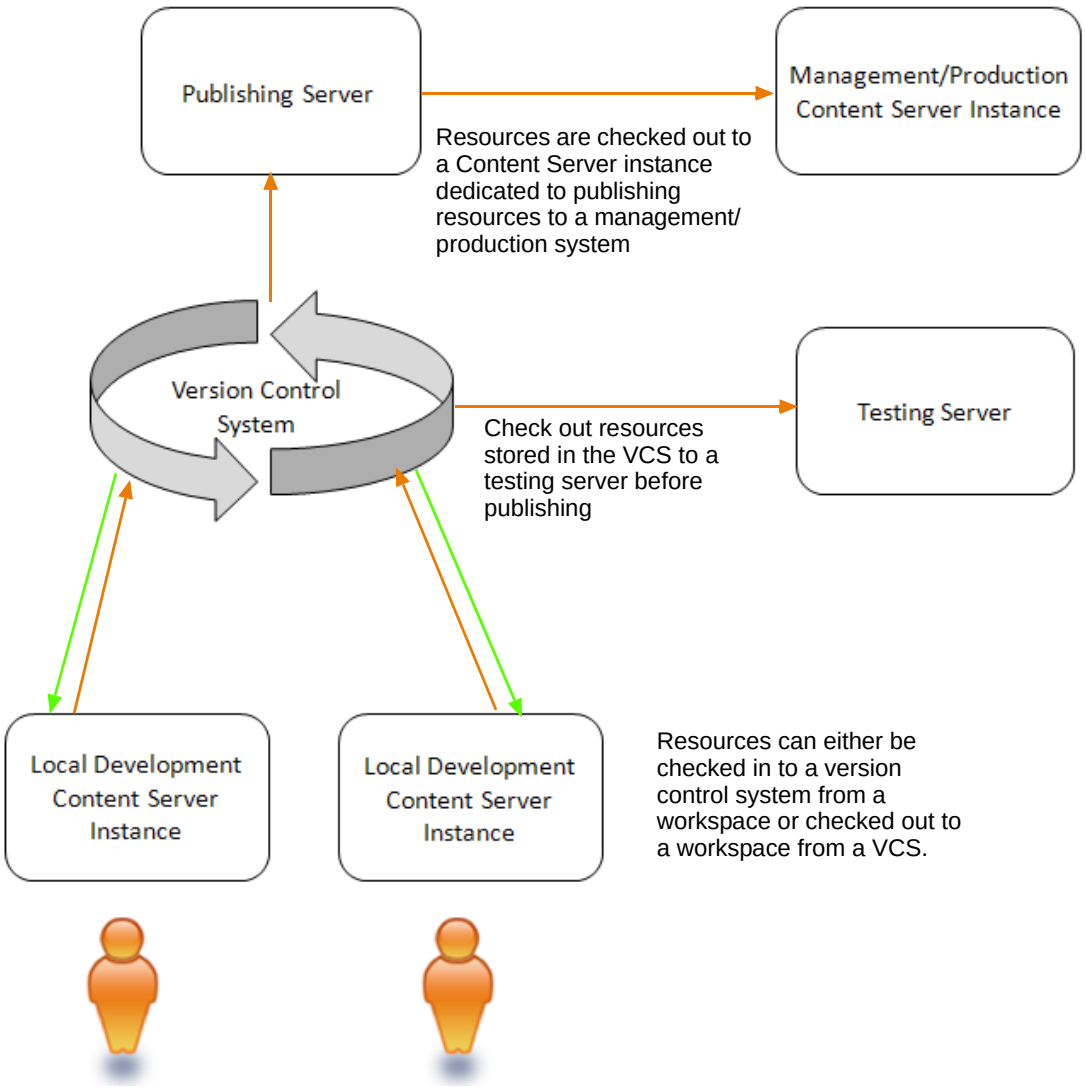
CSDT supports the exchange of resources between Content Server instances, this can be accomplished with the implementation of a version control system. While CSDT does not provide any tools for integrating with version control systems, the file system structure in which CSDT stores resources in workspaces supports the implementation of version

control integration. A workspace's file system structure enables the resources to be tracked by a version control system.

Checking in resources from your workspace to a version control system enables you to exchange those resources with other developers. You can also update your workspace with the resources checked in to the version control system by other developers. Using a version control system you can check out resources to any target system, including testing servers, Management or Production Content Server systems, or another developer's Content Server instance.

Figure 2 illustrates an example of using CSDT with a version control system. This example uses a dedicated Content Server instance to publish resources to a Management/ Production Content Server instance. Therefore, the Approval/Publishing feature provided by Content Server can be used to publish resources that were checked out from the version control system. This example is the recommended way to use a VCS with CSDT, but it is not required.

Figure 2: Using CSDT with a version control system



Next Steps

The rest of this guide provides information about using CSDT to manage Content Server resources in a distributed development environment. The next chapter provides instructions for installing CSDT and integrating a Content Server instance with the Eclipse IDE. The next chapter also provides information to help you get started creating and managing resources in an Eclipse-integrated Content Server instance. For information and instructions, see Chapter 2, "Quick Start."

Chapter 2

Quick Start

This chapter contains instructions for setting up CSDT and integrating a Content Server instance with the Eclipse IDE. This chapter also provides a brief overview for managing Content Server resources in Eclipse.

This chapter contains the following sections:

Prerequisites

Before setting up Content Server Developer Tools, ensure the following requirements are met:

- CSDT requires a fully functional, licensed Content Server 7.6 instance. The Content Server instance must have the Web Experience Management (WEM) Framework installed.
- After you have integrated Eclipse with Content Server, you must log in to Content Server with general administrator credentials (for example, `fwadmin/xceladmin`). This user must be a part of the `RestAdmin` group.
- To use the command-line tool feature, you must have an advanced knowledge of CSDT. Information and instructions about running and using the command-line tool are provided in Chapter 7, "Command-Line Tool."

Setting Up Content Server Developer Tools

1. Install Eclipse 3.6 Helios J2EE edition on the computer Content Server 7.6 is installed. You can download Eclipse from the following URL:

   `http://eclipsesource.com/en/downloads/eclipse-helios-download/`
2. Unzip `csdt.zip` located in the rollup installer (`Rollup/csdt`). Open the `csdt-eclipse` folder and save the `com.fatwire.csdt.eclipsecsdt_1.0.0.jar` file to the `plugins` folder under your Eclipse installation.
3. Start your local Content Server.
4. Start Eclipse (`eclipse.exe`) and configure its settings according to your preferences.
5. Open the "FatWire Content Server" perspective:

   In the Eclipse menu bar, select **Window > Open Perspective > Other ... > FatWire ContentServer**.
6. Integrate Content Server with the Eclipse IDE:
   - If you are setting up CSDT for the first time, the configuration screen is automatically displayed.
   - If you have used CSDT before and wish to integrate a different Content Server instance, navigate to the Eclipse menu bar and select **FatWire > Configure**.

   In the configuration screen, fill in the following fields with the information for your Content Server instance:

   a. In the "Content Server Installation Directory" field, click **Browse** to select the directory containing the `futuretense.ini` file for the Content Server instance.

   b. In the "Username" field, enter the user name of a general administrator. This user must be a member of the `RestAdmin` group.

   c. In the "Password" field, enter the password for the user name you entered in step b.

   d. In the "Project name" field, enter a name for the project on which you will be working.

e. In the "Content Server Log File" field, enter the location of your log file (for example, `ContentServer/7.6.0/futuretense.txt`)

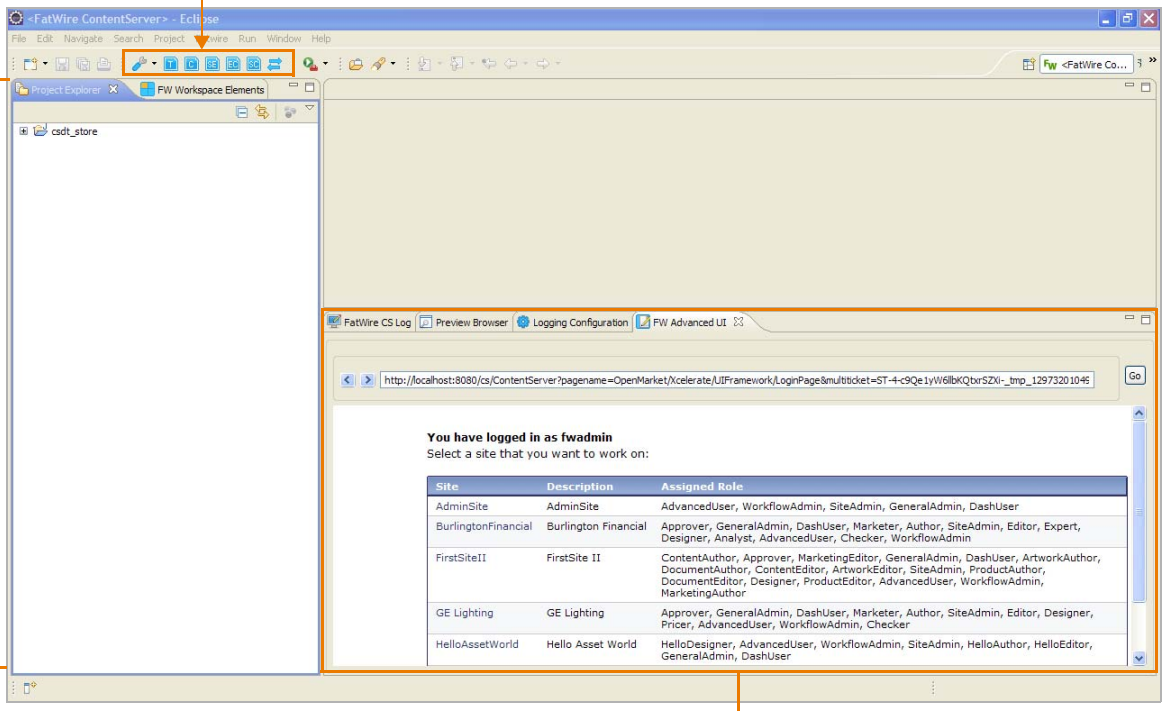
f. Click **OK**.

The "FatWire Content Server" perspective opens. If you are accessing the Content Server-integrated Eclipse for the first time, the FatWire Content Server perspective looks as follows:

FatWire Toolbar, provides:

- Shortcut to the FatWire Content Server configuration screen
- Shortcuts for creating SiteEntry, CSElement, and Template assets
- Shortcuts for creating SiteCatalog and ElementCatalog entries
- Synchronization tool

Project Explorer and **FW Workspace Elements** views



Bottom panel views: FatWire CS Log, **Preview Browser**, **Logging Configuration**, and **Site Editor**.

If the FatWire perspective is rendered as shown above, then you have successfully installed and configured the CSDT plug-in.

Note

The panels containing the Eclipse and CSDT views are interchangeable. To move a view to a different panel, click the view's tab and drag it to the desired panel.

7. (Optional) To associate the *Content Server Tag Reference* and *Javadoc* with Eclipse:
   a. Download the `TagReference.zip` and `javadoc.zip` from the e-docs site, at the following URL:

      `http://support.fatwire.com`

   b. Create a folder named “developerdocs” inside your Content Server installation directory.
   c. Extract the `TagReference.zip` and `javadoc.zip` inside the “developerdocs” folder.
   d. In Eclipse, open the “FW Developer Reference” view. In both the “Tag Reference” and “Javadoc” tabs, click **Home**. The *Content Server Tag Reference* and *Javadoc* are displayed in their respective tabs.
8. If you upgraded your Content Server system to version 7.6, and wish to use CSDT to work with resources created prior to this release (existing resources), see “Using CSDT with Existing Resources,” on page 45.
9. To quickly get started with managing Content Server resources in the FatWire Content Server perspective, continue to the next section. Start with step 4.

Managing Content Server Resources in Eclipse

This section takes you through the Eclipse FatWire Content Server perspective by summarizing the steps you would take to create, edit, and otherwise manage code-based Content Server resources:

- SiteEntry assets
- CSElement assets
- Template assets
- ElementCatalog Entries
- SiteCatalog Entries

To manage Content Server resources in Eclipse

1. Start your local Content Server.
2. Start Eclipse.
3. Open the “FatWire Content Server” perspective:

   In the Eclipse menu bar, select **Window > Open Perspective > Other ... > FatWire ContentServer**.
4. To create resources, do the following:
   - To create a SiteEntry asset, click the  icon and fill in the forms.
   - To create a CSElement asset, click the  icon and fill in the forms.
   - To create a Template asset, click the  icon and fill in the forms.
   - To create an ElementCatalog entry, click the  icon and fill in the forms.
   - To create a SiteCatalog entry, click the  icon and fill in the forms.

   For field definitions, see “Creating Template, CSElement, and SiteEntry Assets” in the *Content Server Developer’s Guide*.

5. To manage the resources you create, edit, delete, or share with other sites use the "FW Workspace Elements" view. Right-click the resource and select the desired option. For information about the available options, see "'FW Workspace Elements'," on page 27.
6. To display CSDT views in panels, do the following:
   a. Select **Window > Show View > Other...**.
   b. In the "Show View" dialog box, select the desired view (located under the **FatWire Content Server** folder):
      - **FW Advanced UI** displays the embedded Advanced interface.
      - **FatWire CS Log** displays the log file for Content Server. This view is only available if you have specified the location of the Content Server log file in the configuration screen (for instructions, see step 6 on page 18).
      - **FW Developer Reference** displays the *Content Server Tag Reference* and *Javadoc*, only if you have associated the *Tag Reference* and *Javadoc* with your current Content Server instance (for instructions, see step 7 on page 20).
      - **FW Workspace Elements** provides access to code-related resources. This view displays the resources in a tree and groups each resource according to its site affiliation.
      - **Logging Configuration** displays a dynamically updating view of the log4j configuration. In this view you can set the log levels of each Content Server logger.
      - **Preview Browser** displays an embedded preview browser.

   For more information about the CSDT views, see "CSDT Views," on page 27.
7. Synchronize Content Server resources by selecting the  icon. The synchronization tool enables you to either export data from Content Server to the CSDT workspace or import data to Content Server from your CSDT workspace.
   - For a quick overview of using the synchronization tool, see "Data Synchronization (Export/Import) Tool," on page 31.
   - For detailed information about synchronizing resources, see Chapter 5, "Synchronization and Data Exchange."

Chapter 3

Content Server Features in Eclipse

This chapter contains information about Content Server features that are provided in Eclipse when it is integrated with CSDT.

FW Content Server Perspective

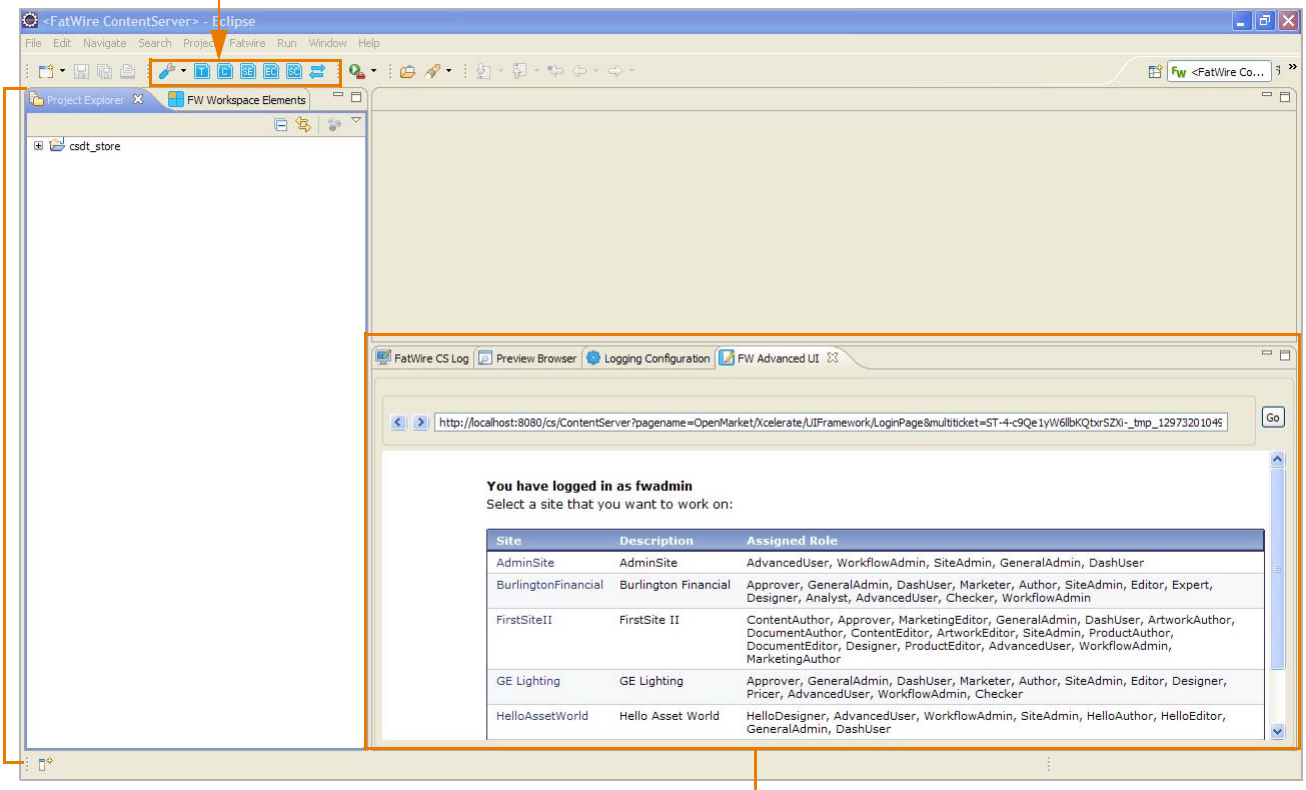
All CSDT functionality in Eclipse is grouped under the FatWire Content Server perspective (**Window** > **Open Perspective** > **Other...** > **FatWire ContentServer**).

Figure 3: FatWire Content Server perspective:
Eclipse > Window > Open Perspective > Other... > FatWire ContentServer

FatWire Toolbar, provides:

- Shortcut to the FatWire Content Server configuration screen
- Shortcuts for creating SiteEntry, CSElement, and Template assets
- Shortcuts for creating SiteCatalog and ElementCatalog entries
- Synchronization tool

Left panel containing the **Project Explorer** and **FW Workspace Elements** views



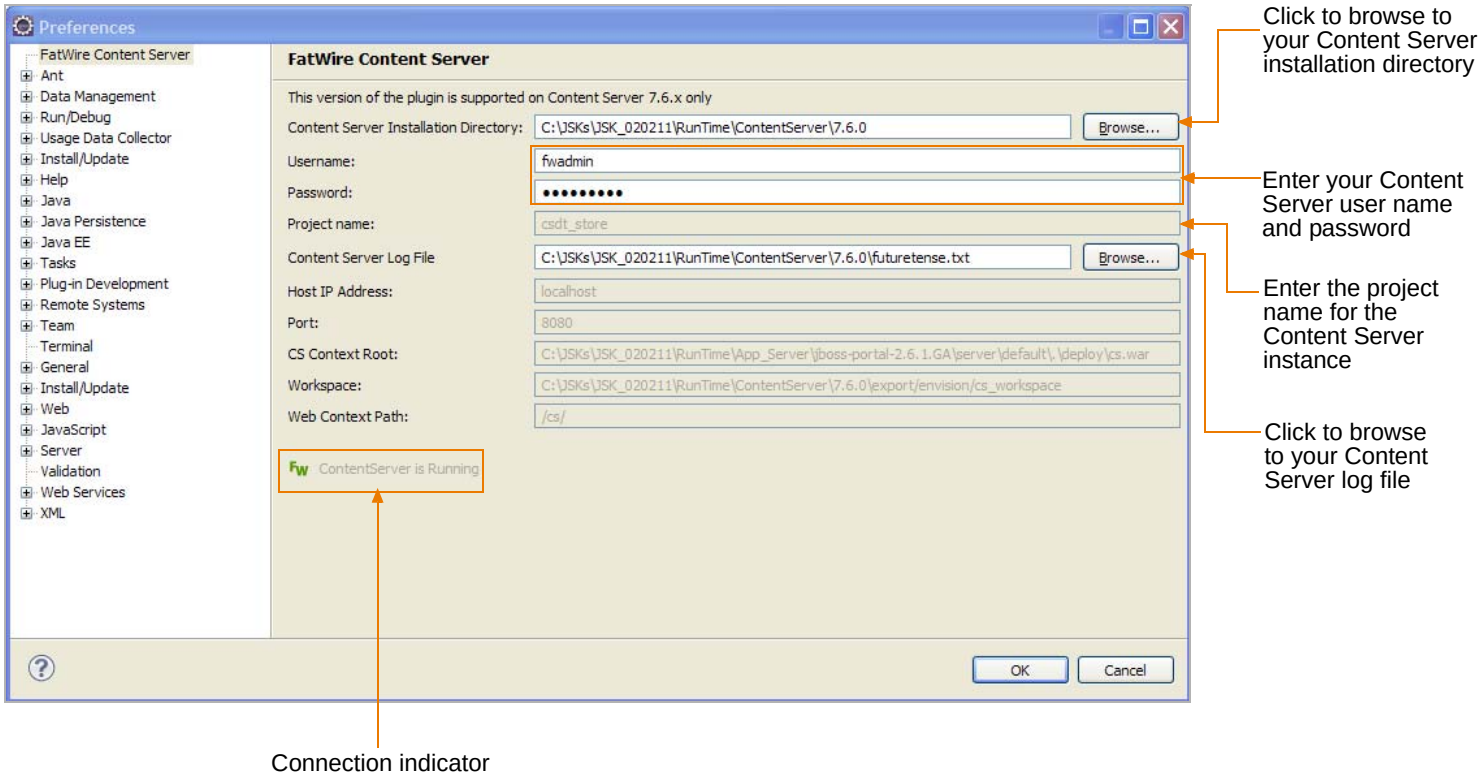
Bottom panel views: **FatWire CS Log**, **Preview Browser**, **Logging Configuration**, and **Site Editor**.

The FatWire perspective contains the following:

- Configuration Screen
- Project and Workspace in Eclipse
- CSDT Views
- Data Synchronization (Export/Import) Tool

Configuration Screen

The configuration screen opens automatically on first access of Content Server-integrated Eclipse. On subsequent access the configuration screen can be opened by selecting the **Configuration** button on the FatWire Toolbar. This screen enables you to specify the Content Server instance with which you wish to work.



The configuration screen requires the path to the Content Server installation directory, a Content Server user that is part of the `RestAdmin` group, a project name for this Content Server instance, and the path to your Content Server log file. After you fill in all the required information, CSDT determines a number of other parameters for your Content Server instance and displays them for your information in read-only fields. In addition, the connection indicator will show whether CSDT is able to connect to the specified Content Server instance.

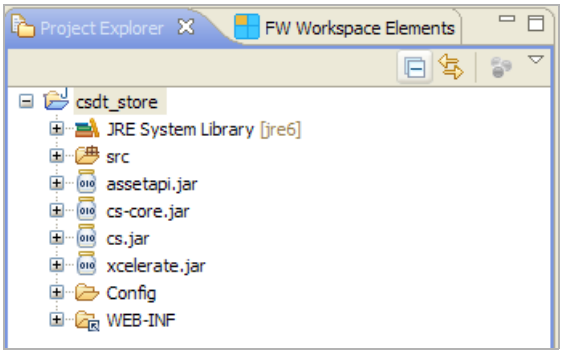
Project and Workspace in Eclipse

Each Content Server instance that is accessed through Eclipse is assigned an Eclipse project. The Eclipse project's folder is displayed in the "Project Explorer" view. The project is needed for tracking CSDT workspace items. Only one project is created by default for each Content Server instance and only one Content Server instance can be serviced by a project.

Note

The main purpose of the project is to facilitate information tracking and process Eclipse events. Projects are managed by the CSDT plug-in. **Do not open, close, or modify the project.**

Each CSDT Eclipse project includes the following elements:



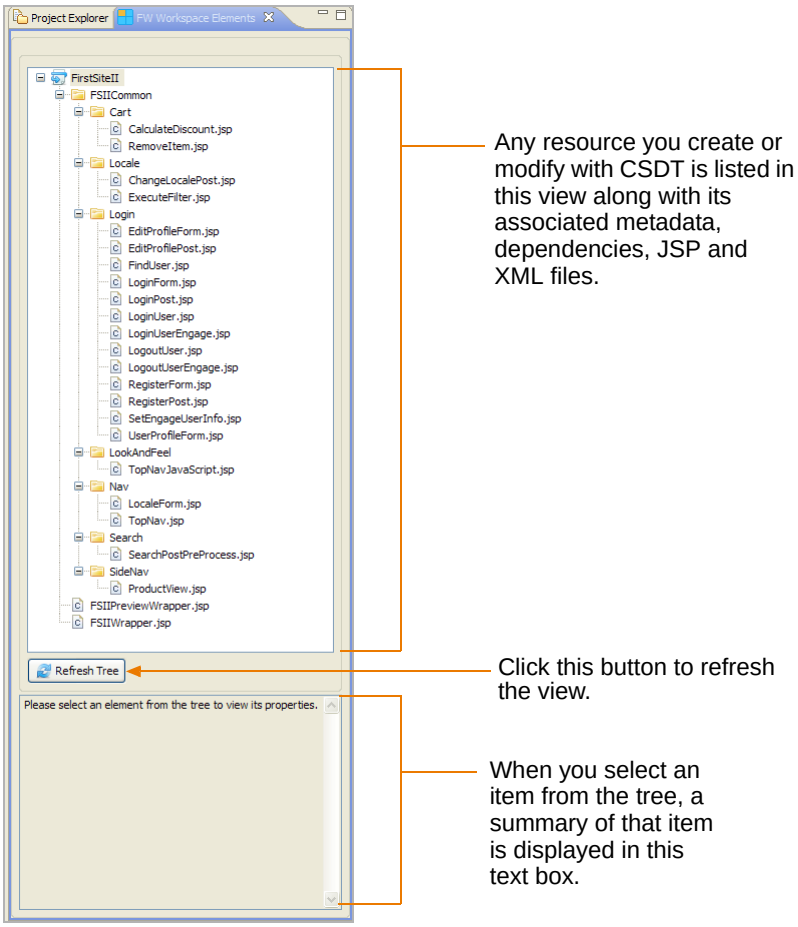
- `src` – The CSDT workspace folder for the current Content Server instance. This folder contains all the files for the resources stored in the CSDT workspace. The resources in this folder can be checked in to a version control system.
- `Config` – Links to common configuration files belonging to the current Content Server instance.
- `WEB-INF` – Links to the current Content Server instance's `WEB-INF` folder.

CSDT Views

- 'FW Workspace Elements'
- 'FatWire CS Log' View
- 'Preview Browser' View
- 'Advanced UI' View
- 'Logging Configuration' View
- 'FW Developer Reference' View
- Wizards

'FW Workspace Elements'

This view provides access to code-related resources. The resources are grouped according to their site affiliation. If you select a resource, a quick summary of that resource is shown in the text box at the bottom of the view.



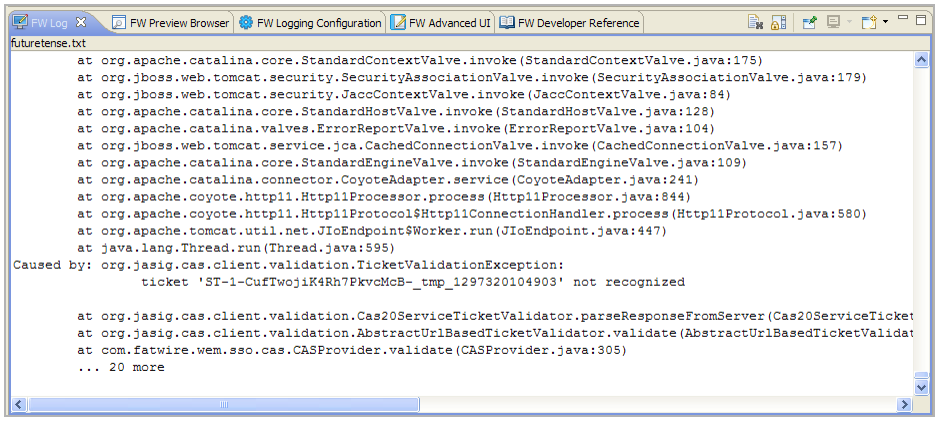
Right-click a resource in the tree to view the available management options. The options that are displayed to you depend on the resource you select:

- **Show Metadata** – Shortcut to the `.main.xml` file, which contains the metadata of the selected item.

- **Site Entry –** View the resource's site entry, share the site entry with other sites, create a new site entry, and delete a site entry.
- **Share** – Manage the sites with which this resource is associated.
- **Properties** – Manage properties of this resource, such as cache criteria and default arguments.
- **Delete** – Delete this resource.

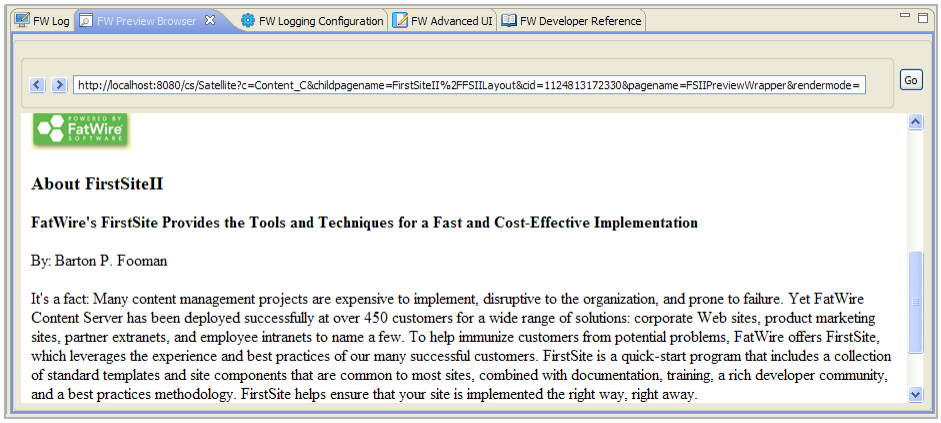
'FatWire CS Log' View

This view shows a dynamically updating record of the Content Server log file. This view can be used to monitor the behavior of your IDE-integrated Content Server instance.



'Preview Browser' View

This view provides a quick way to preview pages. To preview a web page with this view, enter the name of the page to the URL in the address bar and press **Enter** or click **Go**. To refresh the current page, use the **Ctrl + r** keyboard shortcut or click **Go**.



‘Advanced UI’ View

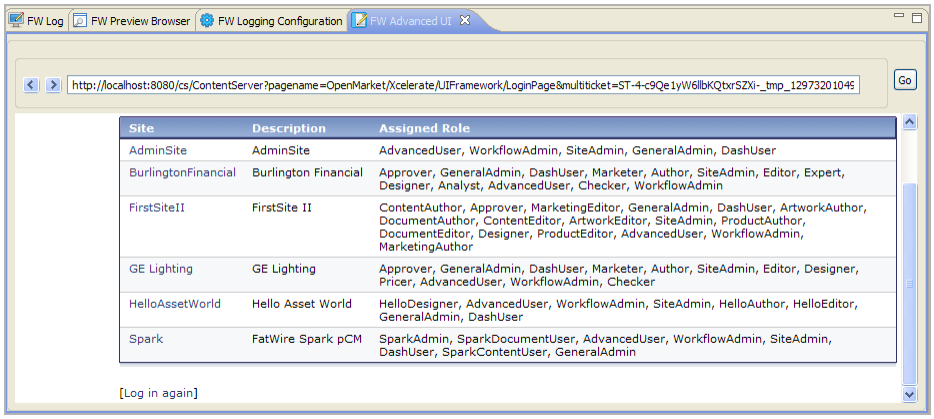
This view displays the Content Server Advanced interface in an embedded browser. This is equivalent to using the Advanced interface in a standalone browser.

Note

The Advanced interface utilizes a Java applet to display the left pane. For information about running applets in browser views, see the Eclipse FAQ, located at the following URL:

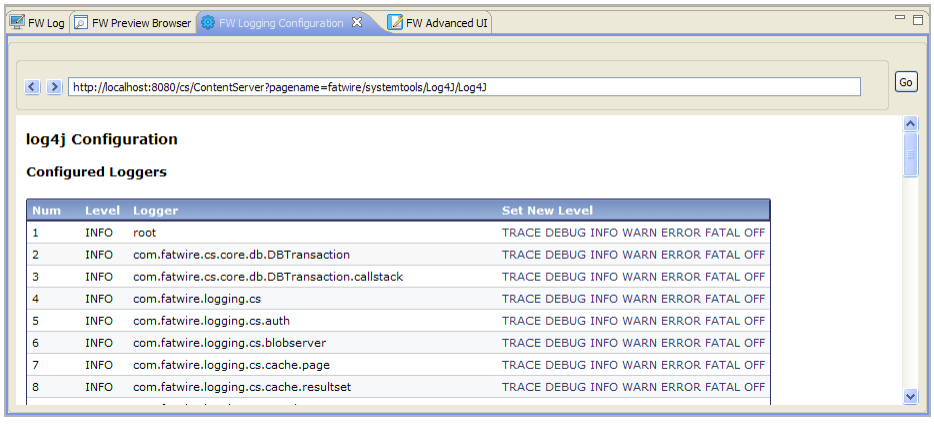
```
http://www.eclipse.org/swt/faq.php#browserapplets
```

If you are unable to run the applet, use the applet-free Advanced interface mode or use a standalone browser to work in the Advanced interface.



‘Logging Configuration’ View

If you have migrated from Content Server’s existing logging system to Apache log4j, this view displays a dynamically updating log4j configuration screen. The log4j configuration screen enables you to view current loggers, change logger levels, add new loggers, and search logs.

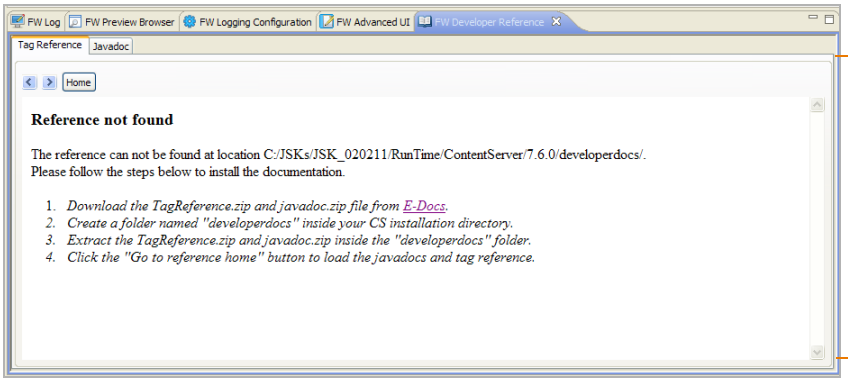


For information about using the log4j configuration screen, see the “System Tools” chapter in the *Content Server Administrator’s Guide*.

‘FW Developer Reference’ View

This view contains two tabs, one of which displays the *Content Server Tag Reference* and the other displays the *Javadoc*. This information is only displayed if you have associated the *Content Server Tag Reference* and *Javadoc* with Eclipse. Otherwise, the view displays instructions for associating the *Tag Reference* and *Javadoc* with Eclipse. For instructions, you can also refer to step 7 on page 20 in the “Setting Up Content Server Developer Tools” section.

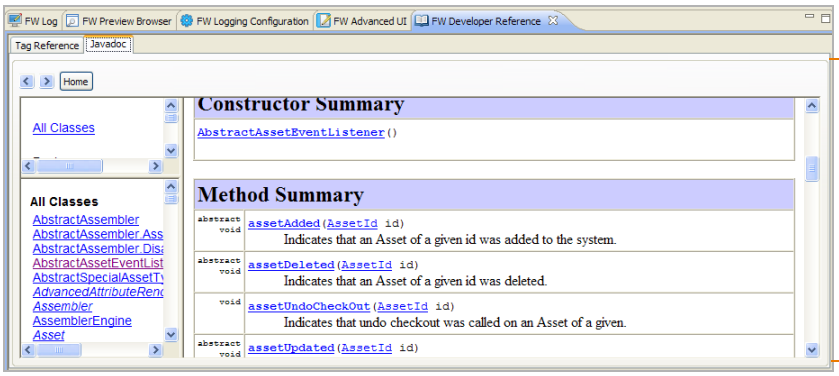
If the Tag Reference and Javadoc are not associated with Eclipse, the tabs display the following:



Instructions for downloading and installing the Tag Reference and Javadoc

If the Tag Reference and Javadoc are associated with Eclipse, the tabs display the following:

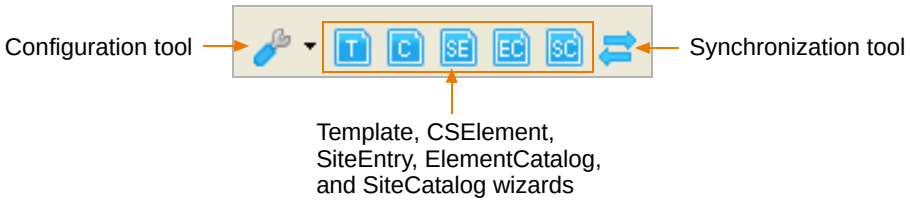
Tag Reference tab



Javadoc tab

Wizards

Wizards can be invoked from either the FatWire menu or the FatWire Toolbar, and enable you to create code-based Content Server resources. The available wizards are: SiteEntry, CSElement, Template, ElementCatalog, SiteCatalog, the configuration tool, and the synchronization tool.



Data Synchronization (Export/Import) Tool

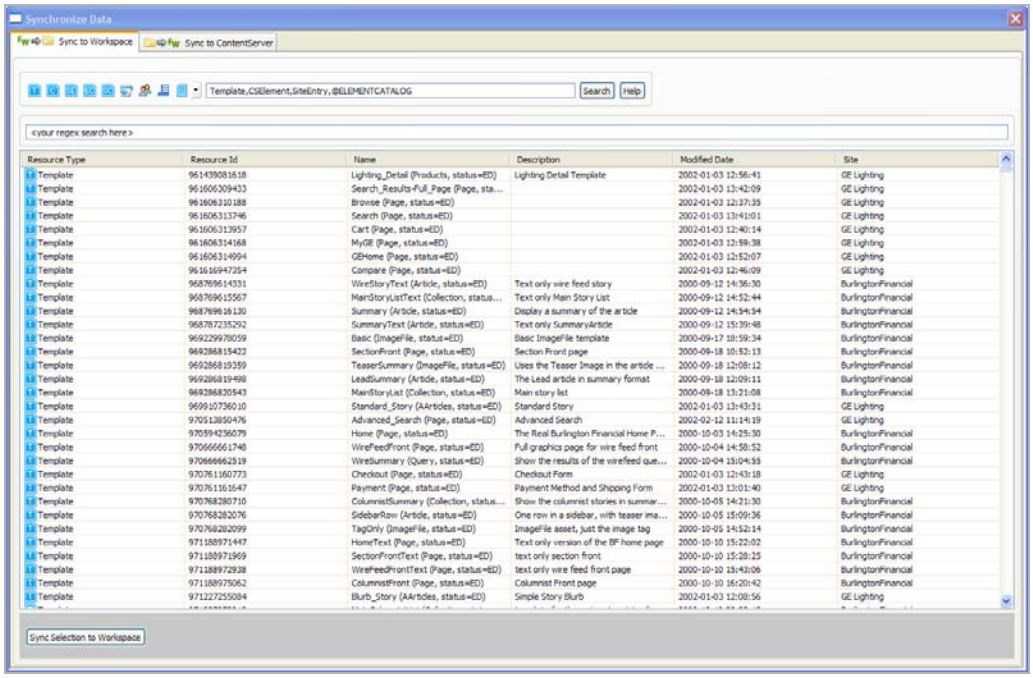
The synchronization tool provides you with two tabs:

- Sync to Workspace (Export from Content Server)
- Sync to Content Server (Import into Content Server)

Sync to Workspace (Export from Content Server)

Sync to Workspace is used to export data from the IDE-integrated Content Server to your CSDT workspace. In the process, CSDT serializes selected resources (transforms database representations into files) and copies the serialized representation to the CSDT workspace. You can then modify the resources in Eclipse.

Figure 4: synchronization icon > Sync to Workspace tab



To export items from Content Server to your workspace

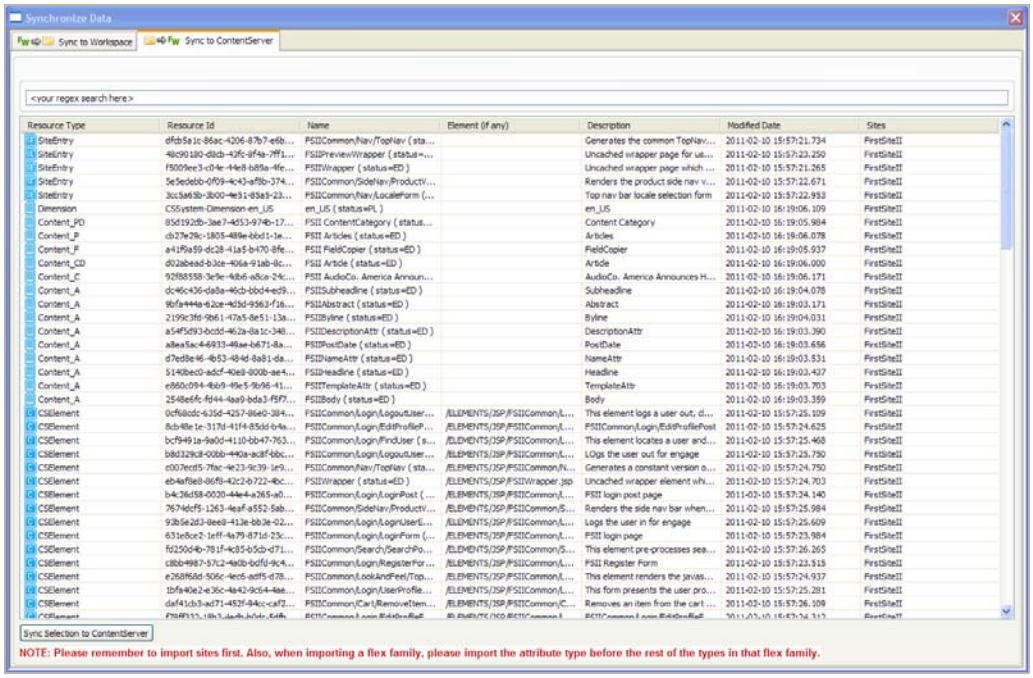
1. Select the items you wish to export. (To narrow down the list of items, go to the "regex" search bar and enter the name of the asset type you are searching for. To search for multiple asset types, enter a comma-separated list.)
2. Click **Sync Selection to Workspace**.

   The assets you exported to your CSDT workspace are now listed in the "FW Workspace Elements" tree tab.

Sync to Content Server (Import into Content Server)

Sync to Content Server is used to import resources from your CSDT workspace into the IDE-integrated Content Server. In the process, CSDT transforms the selected resource to its native Content Server representation and copies it to the Content Server database.

Figure 5:  synchronization icon > Sync to ContentServer tab



To import items into Content Server from the workspace

1. Select the items you wish to import. (To narrow down the list of items, go to the "regex" search bar and enter the name of the asset type you are searching for. To search for multiple asset types, enter a comma-separated list.)
2. Click **Sync Selection to ContentServer**.

Next Steps

The rest of this guide provides information about using the Content Server features, provided by CSDT, in the Eclipse IDE. Proceed to the next chapter (Chapter 4, “Developing JSPs”) for information about JSP development in Eclipse.

Chapter 4

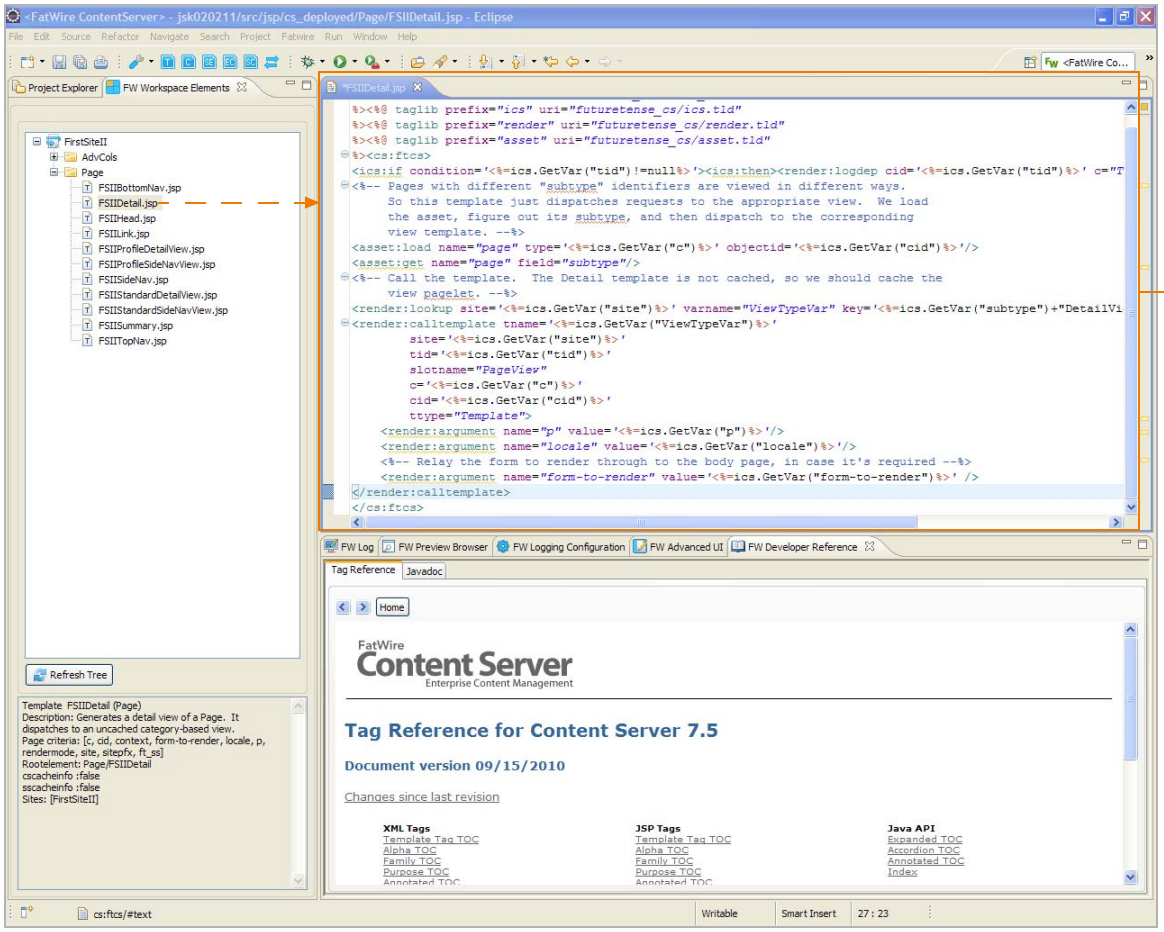
Developing JSPs

This chapter contains information about developing Content Server JSPs with CSDT. This chapter contains the following sections:

JSP Development with CSDT

CSDT supports the development of Content Server JSPs using the native Eclipse JSP editor. The Eclipse JSP editor includes support for Content Server tag and Java API completion, syntax highlighting, and debugging. Figure 6 shows an example of a Content Server JSP in the Eclipse editor.

Figure 6: Eclipse JSP editor



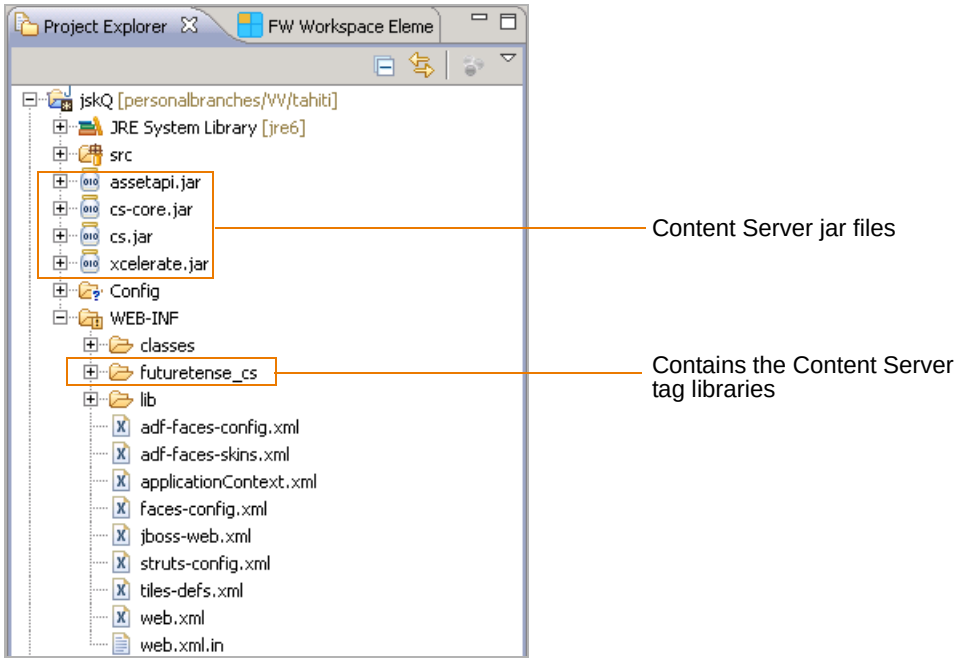
Content Server JSPs can include page caching, resultset caching, and associated metadata such as Template assets, CSElement assets, or ElementCatalog entries. The metadata of a JSP enables Content Server to track and manage it. CSDT handles a JSP's underlying Content Server processes transparently, including tracking the JSP and its corresponding metadata. When you save a JSP in Eclipse, CSDT automatically synchronizes those changes with the current Content Server instance. Any metadata associated with the JSP is also synchronized with Content Server automatically. This enables you to view the changes in Content Server as soon as you save the JSP in Eclipse.

For example, if the JSP is associated with a Template asset, CSDT saves the Template asset with the updated JSP.

Tag and Java API Completion

Eclipse provides tag and Java API completion features. Eclipse uses the tag libraries and jar files belonging to the current Content Server instance to provide the appropriate code completion for Content Server related tags and Java APIs. The Content Server tag libraries and jar files are automatically linked to your Eclipse project, and contained within the Eclipse project folder (located in the “Project Explorer” view):

- The tag libraries are contained in the `futuretense_cs` folder under the `WEB-INF` folder.
- The jar files are contained under the main Eclipse project folder.



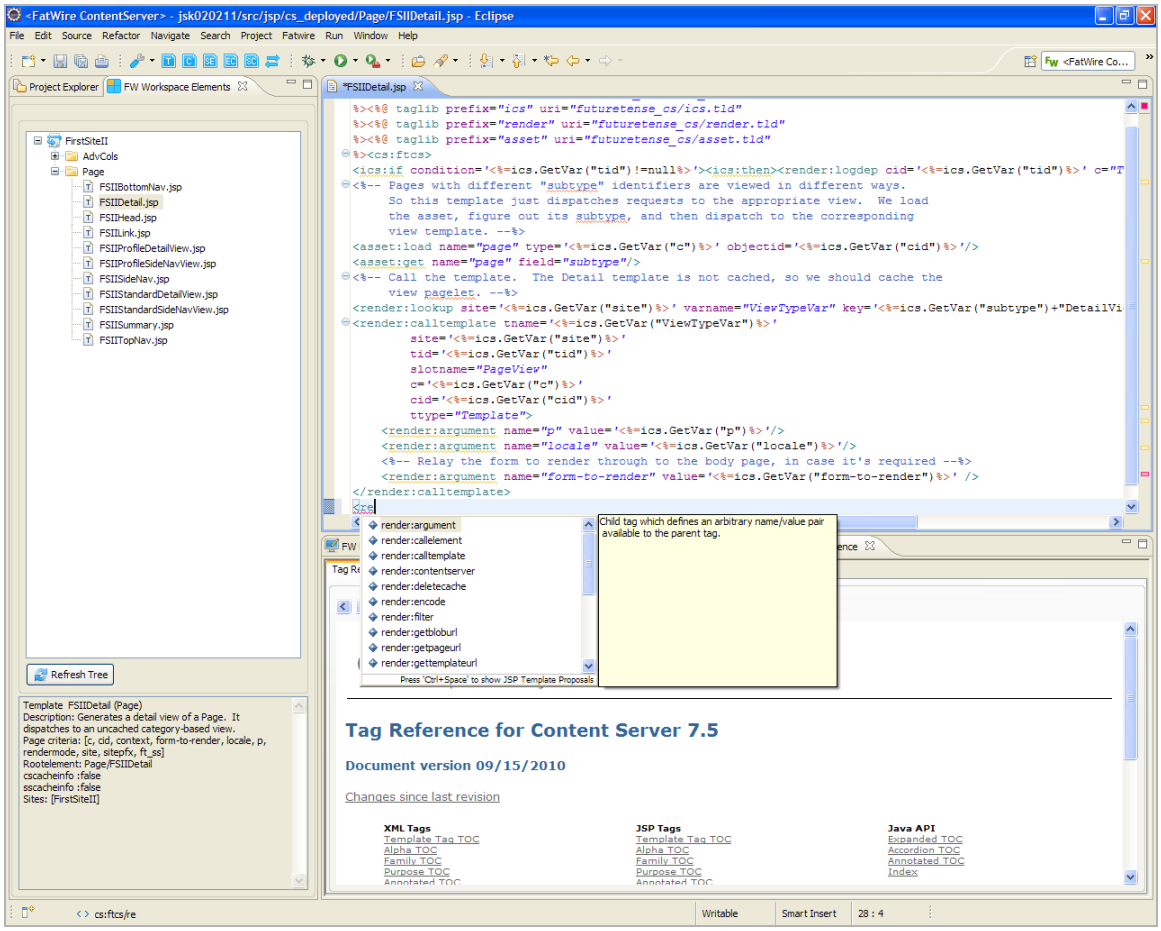
Note

When you use the tag and Java API completion feature, keep in mind the following:

- Make sure you follow strict JSP coding standards. This way your code can be deployed on any application server.
- Eclipse code completion displays all public Java methods contained within the Content Server jars, only use the APIs that are in the FatWire documentation. Using undocumented functionality is risky and unsupported.

Associating the *Content Server Javadoc* and *Tag Reference* with Eclipse enables the tag and Java API completion features to display information about each tag and piece of Java code you use when managing a Content Server JSP. For example, when you are working with a Content Server JSP and you begin to type the name of a tag, a window opens listing code completion suggestions. If you associated the *Tag Reference* and *Javadoc* with Eclipse, a second pop-up window is displayed containing information about each suggestion (see Figure 7).

Figure 7: Tag and Java API completion feature



In addition to adding functionality to the tag and Java code completion features, the *Javadoc* and *Tag Reference* are both made accessible in the "FW Developer Reference" view. For more information, see "'FW Developer Reference' View," on page 30.

Debugging

To debug Java and JSP code in CSDT, you must first attach the debugger to the JVM process that runs Content Server. It is recommended to do so with remote debugging. To attach the Content Server JVM, follow the instructions provided by Eclipse at the following URL:

`http://www.ibm.com/developerworks/library/os-ecbug/`

Once the JVM is attached to the debugger, you can set breakpoints in your JSP and Java code, view variables, and so on.

Chapter 5

Synchronization and Data Exchange

This chapter provides information about the export/import features supported by CSDT. This chapter also provides information about exchanging resources between Content Server instances, and the CSDT mappings processes.

This chapter contains the following topics:

CSDT Synchronization

Synchronization is the bi-directional flow of resources between a Content Server instance and its associated workspace. Using CSDT, you can perform the following synchronization operations:

- Export/import assets with built-in dependency resolution and ID mapping.
- Export/import asset types, such as flex families and AssetMaker asset types.
- Export/import site definitions, roles, start menu items, and tree tabs.
- Export/import SiteCatalog and ElementCatalog entries.
- (Command-line tool operation) Perform site re-mapping. For example, creating reusable modules which can be imported into any Content Server site.

Exporting or importing all resources of a given site enables you to track the entire site in a version control system. Advanced developers can use the command-line tool to re-map the resources of one site to another by creating reusable modules (custom workspaces).

Synchronization Scenarios

Depending on the scenario, resources are synchronized either automatically or manually.

Resources between Content Server and Eclipse are automatically synchronized when the following actions are performed in Eclipse:

- Code-based resources (Templates, CSElements, SiteEntries, ElementCatalog entries, and SiteCatalog entries) are created with the CSDT wizards in Eclipse.
- Code-based resources (Templates, CSElements, and ElementCatalog entries) stored in the CSDT workspace are edited in Eclipse. This includes edits to JSP files, XML files, metadata, and other files associated with the resource.

  For example, if you edit a resource's associated JSP file in the Eclipse editor, CSDT automatically synchronizes the changes into the Content Server instance. Using the Eclipse editor, advanced developers can also edit metadata files (`.main.xml`) of flex definitions and CSDT will automatically synchronize the changes into Content Server. However, we recommend using the Advanced interface to modify flex definitions.

In certain cases, resources must be manually synchronized using either the Synchronization tool in the Eclipse IDE or (for advanced developers) the command-line tool. Manual synchronization is required when:

- The Eclipse editor is not used to edit resources stored in the CSDT workspace. For example, when resources are copied to the CSDT workspace from a shared network file system or a version control system.
- Content Server resources are modified in the Advanced or Dash interface.

Note

The Eclipse IDE provides an embedded Advanced interface. However, Eclipse does not detect the changes that are made using this interface. Therefore, working in the embedded Advanced interface is the same as working in a standalone browser running the Advanced interface.

- Content Server is not running while you are creating or editing resources in the Eclipse IDE. Once Content Server is restarted, you must manually synchronize the resources you created or edited.

Using the command-line tool to synchronize resources is mainly for deployment purposes, such as nightly builds that are deployed to test servers. For example, an advanced developer can embed a synchronization command into a script for an automated deployment procedure. For information about running and using the command-line tool, see Chapter 7, "Command-Line Tool."

Dependency Resolution

Content Server resources often depend on other resources. For example, a flex asset requires an associated flex definition to exist before it can be created. In turn, the flex definition depends on a set of attributes and possibly other resources. Therefore, all flex constructs require that the flex family exist on the system. To import a flex asset into an empty Content Server system, you must first create a flex family to which the flex asset will be associated. Then, create the following:

1. Create the flex attributes. For example, name, address, age, and so on.
2. Create the desired flex parent definitions.
3. Create flex definitions.
4. Create the desired flex parents.
5. Create flex assets.

When you export a flex asset, CSDT performs all dependency resolutions for that asset and automatically exports all of its dependencies. Therefore, you only need to select the desired resource (such as the desired flex asset) and CSDT computes all of the asset's dependencies.

Note

CSDT does not resolve a resource's dependency on site definitions. This enables you to choose whether you want to export or import an entire site, a subset of sites, or completely ignore site definitions (for example, if you are using the command-line tool to create a reusable module that can by imported into any site). For a detailed example of creating a reusable module, see Appendix B, "Using the Command-line Tool to Create Reusable Modules."

Data Exchange and Mappings

CSDT uses ID and site mapping processes to enable developers to exchange resources between Content Server instances. This section contains the following topics:

ID Mapping

Each resource created in Content Server is assigned a unique local identifier. A resource's local identifier is unique only to the Content Server instance on which it was created. Since multiple Content Server instances will be used to create resources, it is possible for two different resources, on separate Content Server instances, to have the same local identifier.

To uniquely identify resources, CSDT assigns each resource a globally unique identifier (`fw_uid`), which is unique across all Content Server instances. In addition, when you import a resource into a Content Server instance, CSDT assigns a new local identifier to that resource on that instance. If the resource references other assets (such as associations, asset pointers, and flex definitions), a new local identifier is generated for each of those assets. On subsequent imports to that Content Server instance, the resources are assigned the same local identifier. CSDT maintains the resources' `fw_uid` values across all Content Server instances. If the resource and its referenced assets are imported back into their original Content Server instance, CSDT re-maps their local identifiers back to their original value.

Note

Certain Content Server resources, such as Template assets, flex attributes, and tree tabs have unique name constraints. To avoid name conflicts, make sure each resource is uniquely named across all Content Server instances.

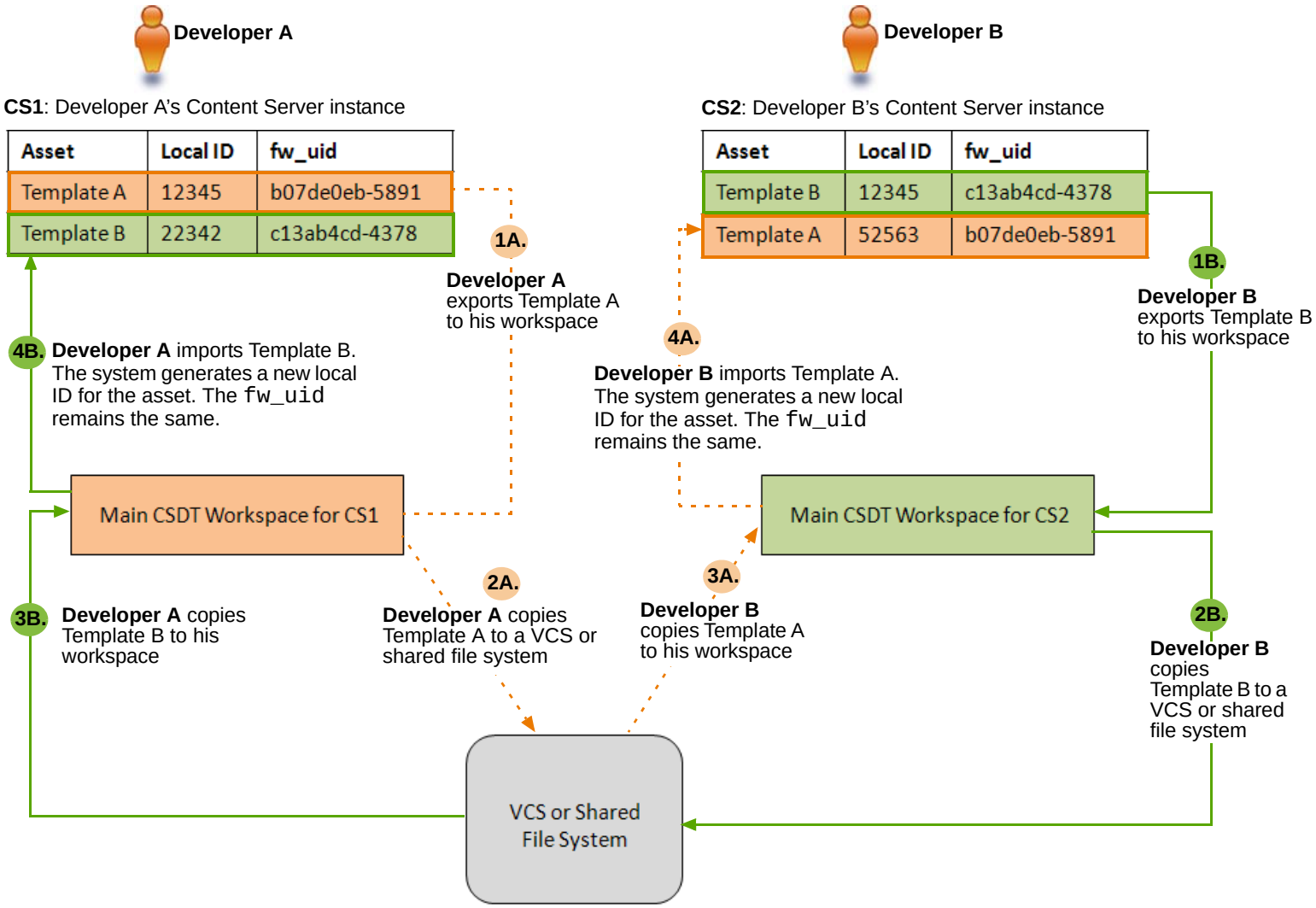
For example, (as shown in Figure 8, on page 43) Developer A is working with a Content Server instance named CS1 and Developer B is working with a Content Server instance named CS2. Both developers created a completely different Template asset. Developer A created Template A and Developer B created Template B. The two Template assets have different `fw_uid` values and different names. However, since local identifiers are randomly assigned, both Template assets, by chance, have been assigned the same local identifier (`12345`). Developers A and B want to exchange Template assets between each other's Content Server instances. Developer A wants to import Template B into the CS1 instance, and Developer B wants to import Template A into the CS2 instance.

Figure 8 illustrates the steps both developers take to exchange Template assets between their Content Server instances. Both Template assets' local identifiers are re-mapped when imported into the other developer's Content Server instance. When Template A is imported into the CS2 instance, the system assigns it the local identifier `52563`. When Template B is imported into the CS1 instance, the system assigns it the local identifier `22342`. In each case, the `fw_uid` values for both Template assets remain the same.

Note

To exchange resources between Content Server instances, the developers in our examples use a VCS or shared file system. For information about using a VCS, see Chapter 8, "Notes for Integrating with Version Control Systems."

Figure 8: Exchanging two different assets with the same local identifier between two Content Server instances

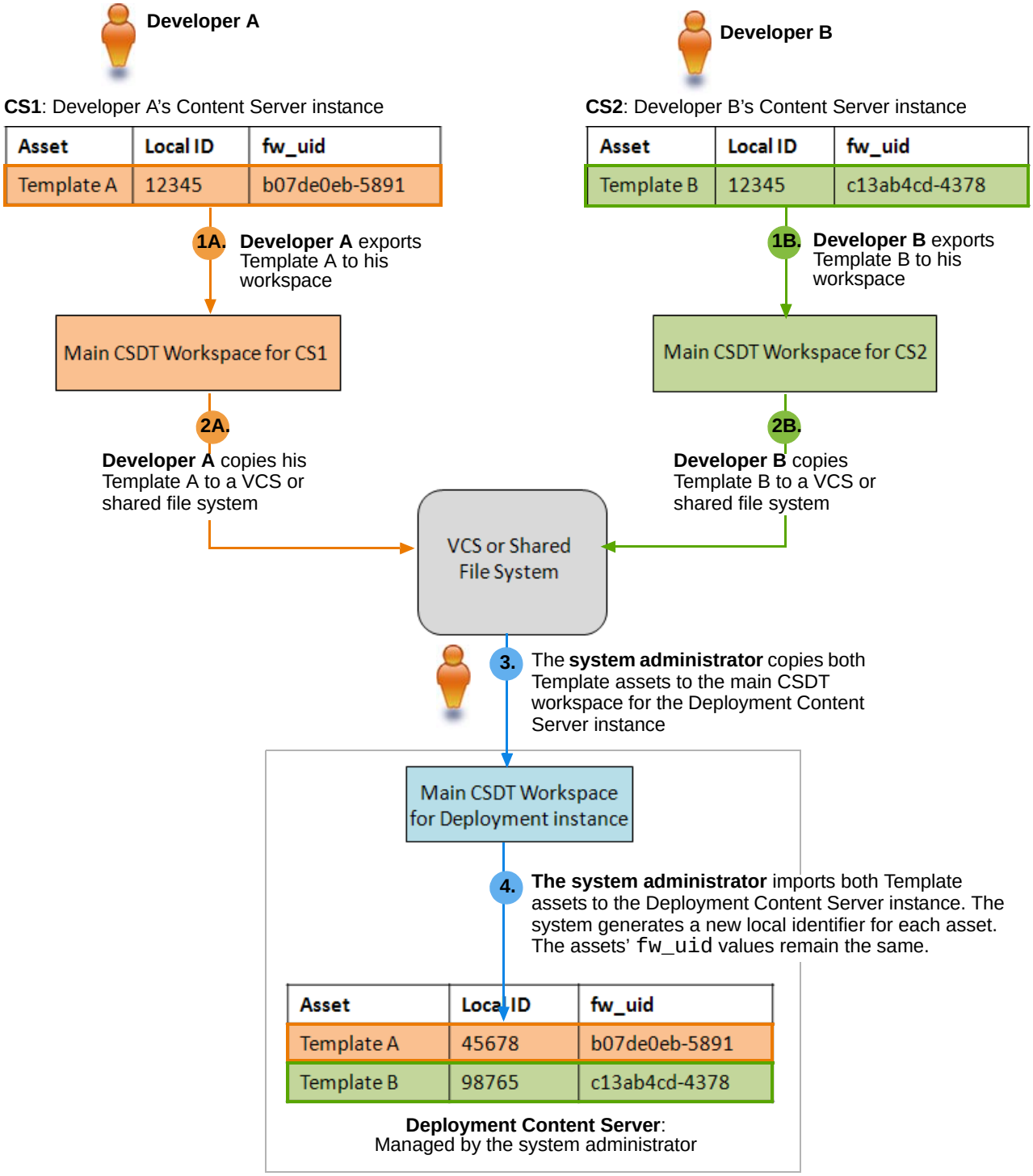


In Figure 9, on page 44, Developer A wants to deploy Template A to the Deployment Content Server instance (managed by the system administrator) and Developer B wants to deploy Template B to the same instance. Both Template assets have the same local identifier (`12345`).

Developers A and B each export their Template to the main CSDT workspace for their Content Server instance. They then copy their Templates to a VCS or shared file system. From here, the system administrator copies both Template assets to the Deployment Content Server's main CSDT workspace. The system administrator then imports the two

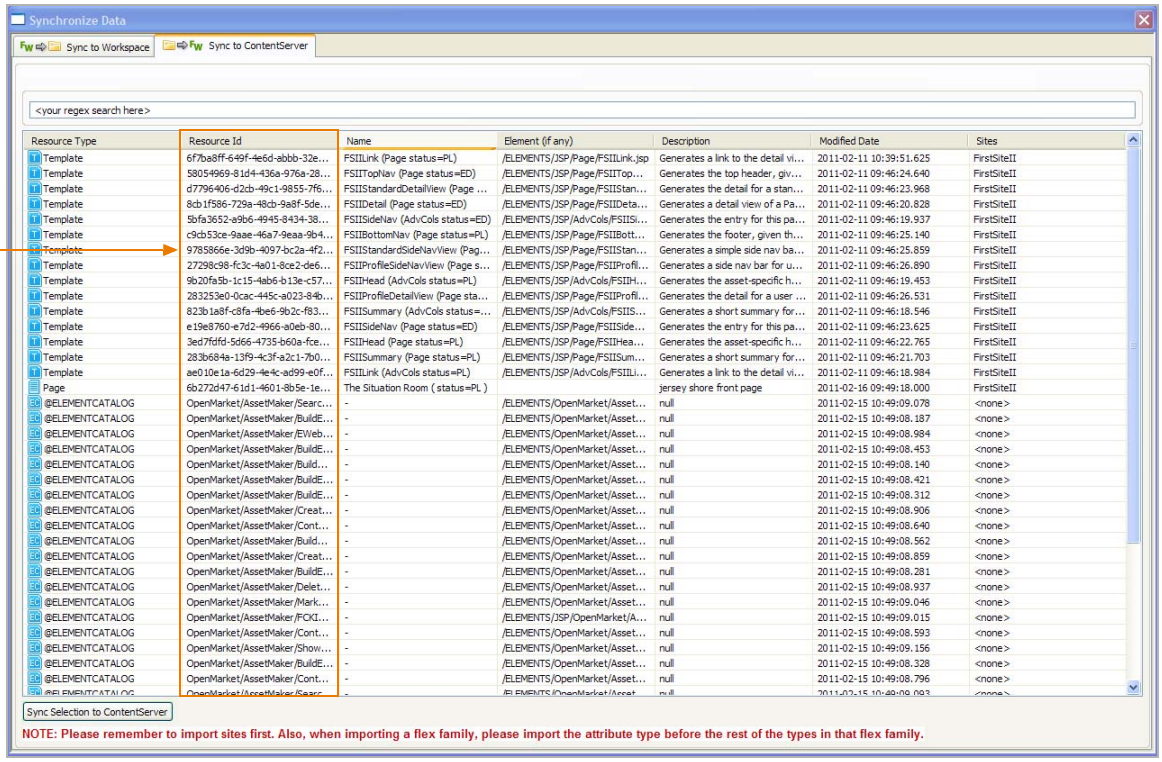
Template assets from the workspace to the Deployment Content Server. Upon import, the system assigns both Templates a new local identifier. Template A is assigned the local identifier of `45678` and Template B is assigned the local identifier of `98765`. The assets' `fw_uid` values remain the same.

Figure 9: Deploying two different assets with the same local identifier to a third Content Server instance



When a resource is exported to a workspace, it is identified only by its `fw_uid`. ElementCatalog and SiteCatalog entries are not assigned an `fw_uid` because these entries are uniquely identified by element name.

The **Resource ID** column lists each Content Server resource by its `fw_uid` (with the exception of ElementCatalog and SiteCatalog entries).



Overriding a Resource's fw_uid

When a resource is created, a UUID value is automatically generated as its globally unique identifier and stored in an asset attribute named `fw_uid`. Advanced developers can use the Asset API to override the default `fw_uid` scheme with their own by modifying the `fw_uid` attribute. For information about using the asset API, see the *Content Server Developer's Guide* and the *Content Server Javadoc*.

> **Note**
>
> We recommend using the default Content Server `fw_uid` scheme. If you override a resource's default `fw_uid` value, you must make sure the value is unique across all Content Server instances. Once you set a resource's `fw_uid` attribute, **do not** change the value.

Using CSDT with Existing Resources

If you upgraded your Content Server system to version 7.6, and wish to use CSDT to work with resources created prior to this release (existing resources), those resources must be assigned an `fw_uid` value that is unique across all Content Server instances. Content Server versions prior to 7.6 provided the `fw_uid` column to all assets and a number of other resource types. However, the value of the `fw_uid` column is `CSSystem:[type]:id`. In Content Server 7.6, a resource's `fw_uid` is generated as a UUID value.

CSDT can map resources with either type of `fw_uid` value, as long as the resource's `fw_uid` value is globally unique. Therefore, you can continue to use an existing resource's current `fw_uid` value (in the format of `CSSystem:[type]:id`).

When you are using CSDT to work with existing resources, do one of the following (or both):

- We recommend continuing to use the existing resource's current `fw_uid` value (`CSSystem:[type]:id`). However, you must ensure that no other Content Server instance has generated the same `fw_uid` value for a different resource. For example, if you have a Content Server development instance and you published resources to a management instance, the `fw_uid` values of the published resources remain the same on both instances. Therefore, synchronizing resources between these two instances using CSDT will not result in ID conflicts.
- If you have existing resources that were created on separate Content Server instances with the same `fw_uid` values, those resources must be assigned a new, unique `fw_uid` value. To avoid ID conflicts, you can either remove the current `fw_uid` value and allow CSDT to generate a new UUID value when you export the resource from a Content Server instance, or you can assign your own unique identifier to the resource. For instructions, see "Overriding a Resource's fw_uid," on page 45.

Note

If you assign a resource a new `fw_uid`, make sure to assign the new `fw_uid` value to every instance of that resource. For example, if you published the resource to another Content Server instance before modifying its `fw_uid` value, make sure you assign the same `fw_uid` to both copies of that resource.

Site Mappings

Most Content Server resources, such as assets, are associated with at least one site. When a resource is exported from a Content Server instance to a workspace, it stores a complete (canonical) list of sites with which it is associated in its `.main.xml` file. The resource's canonical list remains the same on every Content Server instance, unless you add a new site affiliation, remove a current one, or (if you are an advanced developer) override the resource's natural site mapping using the command-line tool.

Natural Site Mappings

By default, CSDT maps resources to their associated sites by referencing the canonical list stored in a resource's `.main.xml` file. If any of the sites referenced in this list exist on the Content Server instance to which the resource is imported, CSDT maps the resource to those sites. If none of the sites referenced in the resource's canonical list exist on the Content Server instance, the import fails.

For example, Developer A installs two sites – News and Sports. On a separate Content Server instance, Developer B also installs two sites – News and Weather. Both developers import the same Template asset into their Content Server instances. This Template asset is associated with both the Sports and Weather sites (both sites are referenced in the asset's canonical list). Upon import, CSDT references the Template asset's canonical list and then maps the asset to the Sports site on Developer A's environment and the Weather site on Developer B's environment.

When Developers A and B share the changes they made to the Template asset with each other, CSDT maps the asset to the appropriate sites on both Content Server instances. The canonical list enables CSDT to recognize the sites with which the Template asset is associated, even when the asset is exported into an instance where some of those sites are not installed.

Overriding Natural Site Mappings With the Command-line Tool

Advanced developers can use the command-line tool to import a resource into sites that are not referenced in its canonical list. The command-line tool enables you to create reusable modules, which are workspaces containing resources that can be imported into any site.

For example, a developer creates a blogging solution within the FirstSiteII sample site. This solution includes resources such as a flex family, assets, and Templates. The developer wants the resources to be imported into various sites, including sites that do not exist yet. Since he is an advanced developer, he uses the command-line tool to export the desired resources to an empty workspace, and then archives the content of this workspace (using a `.zip` or `.tar` format). Using the command-line tool, other developers can then customize the site mappings of the resources contained in this module and manually specify the sites into which the module will be imported.

For more information about using the command-line tool, see Chapter 7, “Command-Line Tool.” For a detailed scenario of creating a reusable module, see Appendix B, “Using the Command-line Tool to Create Reusable Modules.”

Chapter 6

Workspaces

This chapter contains information about how CSDT stores resources exported from an integrated Content Server instance.

This chapter contains the following topics:

Introduction

A workspace is a disk-based repository of serialized Content Server data which represent resources from either the workspace's Content Server instance or another instance's workspace. Workspaces can store any type of Content Server resource including assets, flex families, sites, and so on. Each workspace is associated with one Content Server instance.

By default, Eclipse provides each Content Server instance with a main CSDT workspace (located in the Eclipse project folder) which is used for continuous development when working in the Eclipse IDE. Custom workspaces can be created by advanced developers using the CSDT command-line tool. Custom workspaces can be used for special projects such as creating modules. (For more information about creating custom workspaces, see Chapter 7, "Command-Line Tool.")

With the use of a version control system (such as Subversion) or a shared file system, resources stored on one workspace can be exchanged with other workspaces. Any resource exported from a Content Server instance into the associated workspace can be copied to another Content Server instance's workspace. This makes the resource available for import into the second workspace's associated Content Server instance. For more information about sharing resources between different workspaces, see Chapter 8, "Notes for Integrating with Version Control Systems."

Workspace Structure

Workspaces are created under the `export/envision` folder inside the Content Server installation directory. The main CSDT workspace is located under the `export/envision/cs_workspace` folder. The main CSDT workspace is the only visible workspace in the Eclipse project folder.

All workspaces have the same structure. Each resource contained in a workspace is stored as a single file or several interrelated files. The main file for each resource ends in `.main.xml` and contains resource-specific metadata. This main file also contains links to other files associated with the resource (such as an attached document, a JSP file, or a blob). This enables each resource to be fully self-contained, as long as all of a resource's associated files are stored in the workspace. Otherwise, the resource is incomplete.

If a resource has multiple files, those files are listed in the bottom section of the `.main.xml` file as `storable0`, `storable1`, and so on. The associated files of any given resource have similar names. This way, all of a resource's associated files appear together, except ElementCatalog entries which are stored separately to preserve their original root path.

The location of a resource's files in the workspace depends on the type of resource. The workspace is divided into the following sections:

- `src/_metadata` – The metadata section of a given resource which contains assets, asset types, sites, roles, and so on. In addition, legacy XML code is stored under the `ELEMENTS/` subfolder.
- `src/jsp/cs_deployed` – This section stores a resource's JSP file under its proper path.

Since workspaces have a highly consistent structure, resources from one workspace can be copied to another. As with all file system copy operations, ensure you are not overwriting files that have the same name.

Asset Storage Structure

Assets are stored under folders named `src/_metadata/ASSET/`*`asset type`*. Under this structure there is a two-level hash-based hierarchy, which contains asset data. The name of the asset file is based on the asset name and its `fw_uid` value. If the asset includes attached documents or blobs, the file name is based on the asset name, attribute name, `fw_uid` value, and the name of the document or blob (if any).

For example, a Document_C asset named *FSII IES_Manual.pdf* contains an attached document called *IES_MDPlayer_Manual.pdf*. Therefore, this asset is stored as two separate files:

- The first is the `.main.xml` file, which contains the asset's metadata and links to the files associated with the asset:

  ```
  .src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_Manual
     .pdf(aa0b47b5-f558-49d4-a6ac2ee012d1b75).main.xml
  ```

- The second is the actual document, which is a PDF file in this example:

  ```
  .src/_metadata/ASSET/Document_C/8/0/FSII IES_MDPlayer_Manual
     .pdf.FSIIDocumentFile(aa0b47b5-f558-49d4-8a6a-c2ee012d1b75)
     .IES_MDPlayer_Manual.pdf
  ```

Note

Since all file names of the asset are based on the asset's name, renaming the asset also renames the file. If you are tracking the asset in VCS, delete the file with the old name.

Code-Based Resource Storage Structure

Templates, CSElements, and ElementCatalog entries are stored under the storage path required by their code elements. The JSP files associated with code-based resources are stored in the workspace under `src/jsp/cs_deployed` and the XML elements are stored under `src/_metadata/ELEMENTS`. The metadata files of code-based resources are stored under the same name as the resource's JSP with the appended `.main.xml` extension. Therefore, the code-based resource's metadata, JSP, and XML files are grouped together in the workspace.

Attribute Editor Storage Structure

Attribute editors are tracked as assets, but also have implicit references to a set of ElementCatalog entries. An attribute editor's ElementCatalog entries are tracked independently.

For example, the TextArea editor uses the `OpenMarket/Gator/AttributeTypes/TEXTAREA` ElementCatalog entry, which is registered as a dependency. CSDT maintains the following files for the TextArea editor:

- The `.main.xml` file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea(e64f983d-9c7c-
     489baedb-476d56f8121e).main.xml
  ```

- The `urlxml` metadata file:

  ```
  src/_metadata/ASSET/AttrTypes/9/10/TextArea.urlxml(e64f983d-
     9c7c-489b-aedb-476d56f8121e).1095346398911.txt
  ```

- The ElementCatalog entry, tracked as an independent resource:
  - The `.main.xml` file of the ElementCatalog entry:

    ```
    src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/
       TEXTAREA.xml.main.xml
    ```

  - The attribute editor's element code:

    ```
    src/_metadata/ELEMENTS/OpenMarket/Gator/AttributeTypes/
       TEXTAREA.xml
    ```

Asset Type Storage Structure

Asset types have a main metadata part and a set of elements. For example, the following is the structure of a Page asset type:

- The main metadata of the page is stored in the `.main.xml` file:

  ```
  src/_metadata/Asset_Type/Page(b8d8ae9-14cc-4554-b80e-
     0c22e39a3ec8).main.xml
  ```

- The associated elements are tracked independently (each element has its own `.main.xml` file):

  ```
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     SearchForm.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     CheckDelete.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     ContentForm.xml.main.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     ContentDetails.xml.main.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     LoadSiteTree.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     IndexReplace.xml.main.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     LoadTree.xml
  src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
     IndexAdd.xml.main.xml
  ```

```
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   SearchForm.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   IndexReplace.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PreviewPage.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   LoadTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PreUpdate.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   Tile.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   SimpleSearch.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   SimpleSearch.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   ContentForm.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   AppendSelectDetailsSE.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   LoadSiteTree.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   AppendSelectDetails.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   ManageSchVars.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PreviewPage.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   CheckDelete.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   ManageSchVars.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PreUpdate.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   AppendSelectDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   IndexCreateVerity.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   ContentDetails.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PostUpdate.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   IndexAdd.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   IndexCreateVerity.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   Tile.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   AppendSelectDetailsSE.xml.main.xml
src/_metadata/ELEMENTS/OpenMarket/Xcelerate/AssetType/Page/
   PostUpdate.xml.main.xml
```

Chapter 7

Command-Line Tool

This chapter is for advanced developers and provides information about running and using the command-line tool.

This chapter contains the following sections:

Introduction

The CSDT command-line tool can be used for deployment and other resource movement activities. Unlike the Eclipse integration, which enables you to work only with the CSDT workspace, the command-line tool enables you to work with any workspace. The command-line tool also provides import and export features which are not available when working in the Eclipse IDE. For example, developers can create reusable modules, which are workspaces containing resources that can be imported into any site.

Running and Using the Command-Line Tool

To run the command-line tool:

1. Unzip `csdt.zip`, which is located in the rollup installer (`Rollup/csdt`). Open the `csdt-client` folder and place the `csdt-client.jar` file in the classpath. Make sure you have met all the requirements listed in the *Supported Platform Document*.

   Note

   The *Supported Platform Document* is available on our e-docs site, at `http://support.fatwire.com`. The site is password protected. Accounts can be opened from the home page.

2. Run the command-line tool (`cmd`) and type the following command:

   ```
   java com.fatwire.csdt.client.main.CSDT  [ContentServer url]
      username= username   password= password
      cmd=export|import|listcs|listds    [options]
   ```

   Replace the placeholder parameters with the information about your development environment and the desired command you wish to execute:

   - `ContentServer url` – The URL of your local Content Server instance, including the Content Server servlet (for example, `http://localhost:8080/cs/ContentServer`)
   - `username` and `password` – The user name and password of a Content Server general administrator. This user must be a member of the `RestAdmin` group (for example, `fwadmin`/`xceladmin`).
   - `cmd` – The command to execute. The following commands are available:
     - `export` - Export data from Content Server to a workspace
     - `import` - Import data into Content Server from a workspace
     - `listcs` - List Content Server content
     - `listds` - List workspace content
   - `options` – Specify one of the following to either import or export:
     - `resources` - Specify which resources you wish to import or export in a semicolon-separated list of resource type and resource ID. To specify multiple resources, use a comma-separated list. To specify all resources of a given type, use the `*` symbol. If you are exporting a resource (to a workspace), specify the resource's local ID. For example, use

`resources=Content_C:12345;Product_C:*` to export a specific Content_C asset and all Product_C assets.

If you are importing a resource (to a Content Server instance), specify the resource's `fw_uid`. To get the resource's `fw_uid`, use the `listds` option.

The following is a full listing of resource selectors:

- `@SITE` – Specify the desired sites
- `@ROLE` – Specify the desired roles
- `@ASSET_TYPE` – Specify the desired asset types
- `@TREETAB` – Specify the desired tree tabs
- `@STARTMENU` – Specify the desired start menu items
- `@ELEMENTCATALOG` – Specify the desired ElementCatalog entries
- `@SITECATALOG` – Specify the desired site catalog entries
- `@ALL_NONASSETS` – Use this short-hand notation to select all non-asset resources
- `@ALL_ASSETS` – Use this short-hand notation to select all available assets
- `asset type` – Specify assets of a certain type.

Note

To verify that selectors are picking up the correct resources before import or export, use `listcs` for export activities and `listds` for import activities. These commands fine-tune the selectors before execution by providing a list of the resources that will be moved.

If resources have dependencies, they are exported and imported automatically. However, dependencies are not listed using the `listcs` and `listds` commands.

- `fromSites` - Select resources from specified sites only.
- `toSites` - (Import only) Override the natural site affiliation during import with a comma-separated list of sites. Specified sites must exist on the target system.
- `modifiedSince` - (Assets only) Select only resources that have been modified since the specified date. The date format is `yyyy-mm-dd hh:MM:ss`.
- `datastore` – (Optional) Specify the workspace you wish to either export Content Server resources to or import Content Server resources from. If you do not specify a value for this parameter, the main CSDT workspace is specified by default. If you are exporting resources and specify a workspace that does not exist, the command-line tool automatically creates the workspace and exports the desired resources to it.

Example Commands

The following is a list of example commands that can be executed using the command-line tool:

- This command exports the specified Content_C assets and all Product_C assets that belong to FirstSiteII and were modified since the specified date. Since no workspace is specified, the CSDT workspace is used by default:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/
   cs/ContentServer username=bob password=password
   resources=Content_C:123432123423,11234234212,111234341234;Pr
   oduct_C:* fromSite=FirstSiteII modifiedSince=2010-08-08
   19:14:00 cmd=export
```

- This command imports the specified Content_C asset and all Product_C assets found in the workspace. Since no workspace is specified, the CSDT workspace is used by default:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/
   cs/ContentServer username=bob password=password
   resources=Content_C:aad618e9-f04e-4ee4-b902-
   076224bb6f7b;Product_C:* fromSite=FirstSiteII cmd=import
```

- This command exports all resources from the site SecondSiteII into a workspace named “TheOutput”:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/
   cs/ContentServer username=bob password=password
   resources=@ALL_ASSETS:*;@ALL_NONASSETS:*
   fromSite=SecondSiteII  datastore=TheOutput cmd=export
```

- This command imports all assets and tree tabs from the workspace named “TheInput” into the site MySite:

```
java com.fatwire.csdt.client.main.CSDT  http://localhost:8080/
   cs/ContentServer username=bob password=password
   resources=@ALL_ASSETS:*;@TREETAB:* toSites=MySite
   datastore=TheInput cmd=import
```

Creating Modules

Modules are sets of related resources exported from your Content Server instance into a given workspace. The `datastore` parameter enables you to specify the workspace you wish to either export Content Server resources to or import Content Server resources from. If you export Content Server resources to a workspace that does not exist, the command-line tool automatically creates that workspace and exports the desired resources into it.

Modules are reusable, and their content can be imported into any CM sites (even if the site is not listed in the resources’ canonical list of sites). To import a module into a CM site, you must execute an import command. In the `datastore` parameter, specify the workspace that contains the desired resources and in the `toSites` parameter, specify the site(s) to which you wish to import those resources. This imports the content of the workspace into the specified CM site(s).

Chapter 8

Notes for Integrating with Version Control Systems

This chapter provides information about storing the resources, contained in the CSDT workspace folder, in a version control system (VCS). This enables you to share the resources in your CSDT workspace with other developers.

This chapter contains the following topic:

Version Control With CSDT

Version control systems (VCS) provide you with the ability to create source code repositories. A VCS can provide advanced tools for versioning, branching, and managing source files. The file system structure in which the CSDT workspace stores Content Server resources enables those resources to be stored on any VCS and enables complete CM sites to be tracked in a VCS.

Integrating CSDT With a VCS

The CSDT workspace is located in the `src` folder of the Eclipse project. This folder can be accessed directly from the Content Server installation directory (under `export/envision/cs_workspace/src`). To copy the content of your CSDT workspace folder to a VCS, you must first determine which VCS you wish to use. Then, check-in the resources stored in the CSDT workspace to the VCS. The VCS you choose to use, determines the steps you must take to check resources in from the Eclipse IDE.

In some cases Eclipse supports the VCS you choose to use by providing a plug-in which allows you to check resources into the VCS directly from Eclipse. For example, if you use the Subversion repository to store the content of your CSDT workspace, the Eclipse IDE supports the Subclipse plug-in. Therefore, you can check resources into the Subversion directory directly from the Eclipse IDE.

The CSDT workspace stores all resources as one or more files, depending on the type of resource. If you check a resource into a VCS, you must also check-in all associated files of that resource. For example, an asset that contains attached documents (such as a PDF) is represented by a metadata file (`.main.xml`) and the associated document file(s). All associated files of the asset must be checked in to the VCS. Otherwise, the check-in fails. For a detailed description of the CSDT workspace layout and for information about how resources are mapped to workspace files, see Chapter 6, “Workspaces.”

Note

Checking data into a VCS from the CSDT workspace does not require an extensive understanding of the CSDT workspace file structure. Instead, most VCS clients detect incremental changes to the CSDT workspace folder and indicate those changes during a VCS commit operation.

Working With a CSDT-Integrated VCS

When you check Content Server resources into a VCS from your CSDT workspace, you are able to exchange those resources with other developers and track changes to those resources over time. The following is an example of a development team using a VCS to share Content Server resources:

Developer A creates a resource in Content Server and exports it to the CSDT workspace. Developer A then checks that resource into a VCS. From the VCS, Developer B can then check-out the resource to his own CSDT workspace. This developer can now modify the resource and then check the changes back into the VCS. Developer A, as well as the rest of the development team, can now see the changes made to the resource from the VCS. This enables the members of the development team to synchronize their CSDT workspaces with the most recent changes made to the resource. Additional developers can join the group by checking-out resources from the VCS into their own, respective CSDT workspaces. As the project advances, the cycle of adding and modifying resources continues.

Note

Content Server provides a revision tracking system for resources that are kept within a given Content Server instance. The Content Server revision tracking system cannot be integrated with a VCS.

Appendices

This part contains the following appendices:

- Appendix A, “Development Team Integration Use Case”
- Appendix B, “Using the Command-line Tool to Create Reusable Modules”

Appendix A

Development Team Integration Use Case

This appendix contains a development scenario involving a team of developers using CSDT to create a CM site and resources. The development team uses the synchronization tool provided by CSDT to manage and exchange resources between multiple Content Server instances. Using the command-line tool, the CM site and its resources will then be deployed as a nightly build.

The sequence of events for the scenario are as follows:

- Today – Develop a Site and Associated Resources
- Three Days Later... Deployment

Today – Develop a Site and Associated Resources

7:14 am – The New Project is Assigned

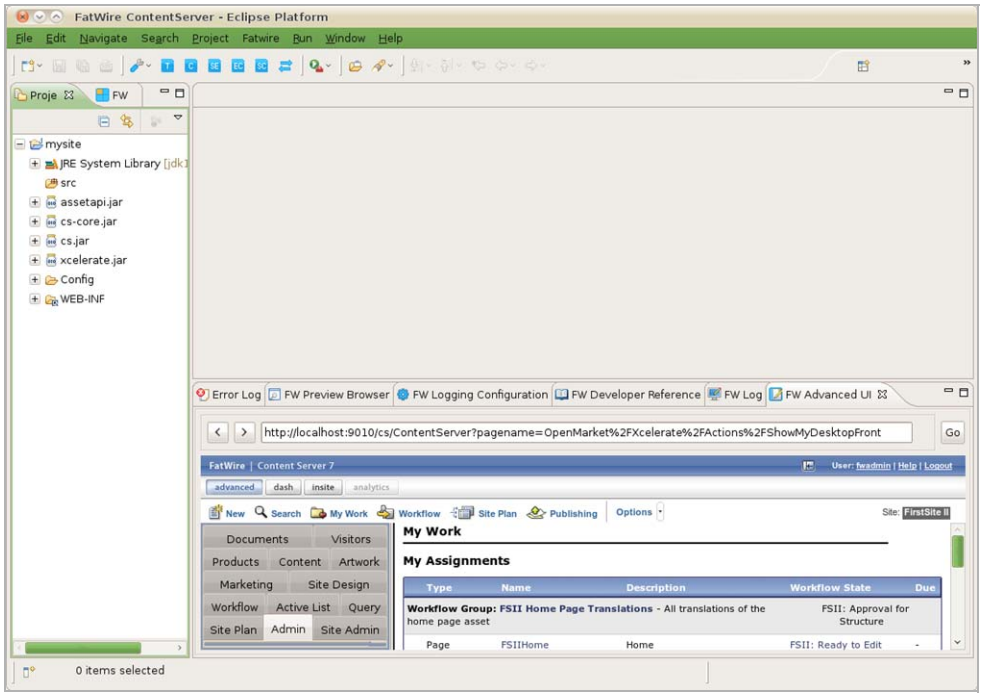
Artie the architect wakes up and finds himself appointed the leader of a new web-based project.

7:34 am – Setting Up CSDT

Artie gets some coffee and installs a Content Server instance on his laptop. He then starts the Eclipse IDE and configures CSDT.

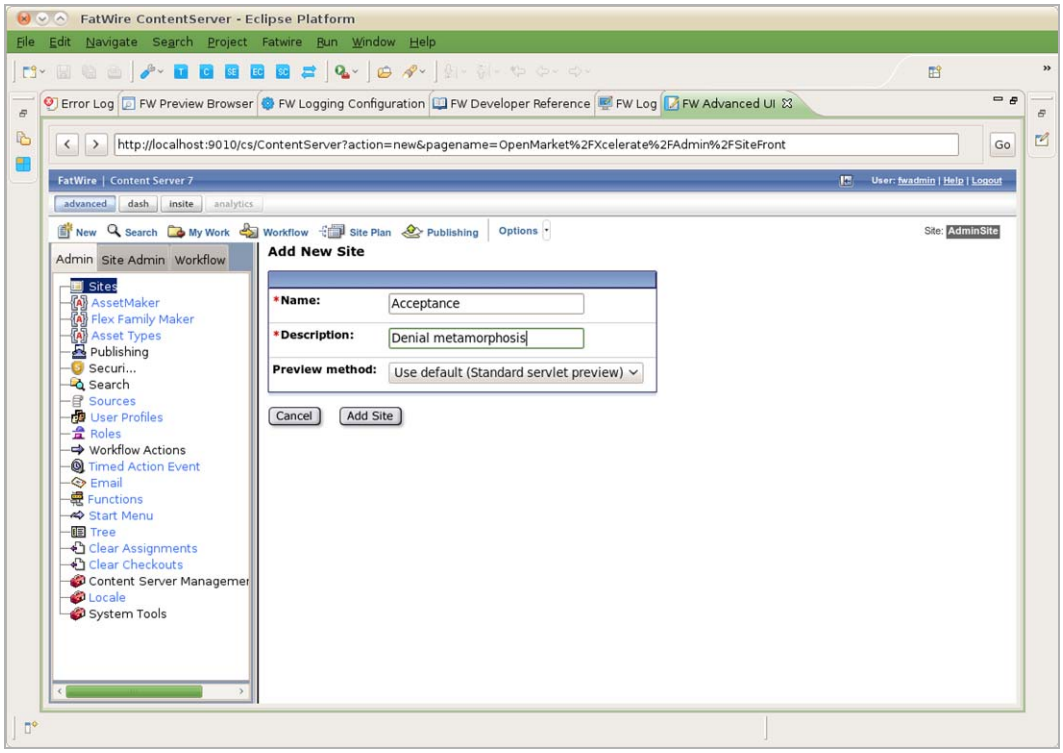
Note

To successfully integrate Eclipse with a Content Server instance, the configuration screen requires Artie to enter the user name and password of a general administrator. This user must be a member of the `RestAdmin` group.



7:45 am – Create the Site Definition

Artie creates the site definition (naming the site “Acceptance” in this scenario) by using the embedded Advanced interface view in Eclipse.

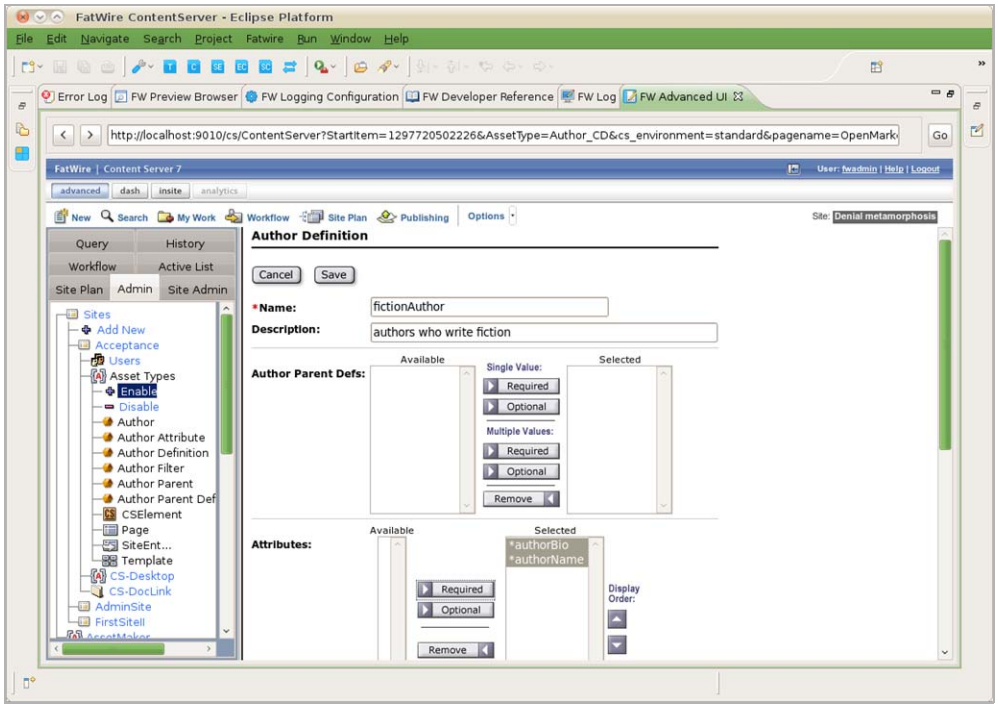


Artie could have used a separate browser window running the Advanced interface to create the site definition. However, being a huge Eclipse fan, he indulges in the fact that he can usually write complete Content Server sites without leaving Eclipse.

7:46 am – Create Resources for the Site

Artie primes the site:

- Enables asset types.
- Assigns permissions.
- Creates and enables a flex family to store information assets (author information assets in this scenario) for the site:
  - Flex Attribute: `Author_A`
  - Flex Parent Definition: `Author_PD`
  - Flex Definition: `Author_CD`
  - Flex Parent: `Author_P`
  - Flex Asset: `Author_C`
  - Flex Filter: `Author_F`
- Creates flex attributes (authorName and authorBio) and a flex definition (fictionAuthor). He then adds the attributes to the flex definition.



8:12 am – The VCS Discussion

Artie arrives at the office and meets with the rest of the development team – Sonoko (coder), Matthäus (coder), and Yogesh (system engineer). The discussion is about whether to use a version control system for the project:

Yogesh: I can set up a version control system in-house, but I would like to avoid doing extra work. Do you guys really want one?

Artie: Well, we expect this project to last several months. We could just create a shared folder on the network and synchronize all our work to it. However, we have to be careful not to overwrite each other's work. For example, if two people are working on the same Template asset, they will have to wait for each other.

Sonoko: Artie, do you remember how the last project turned out to be very intense toward the end? Waiting for other people to finish their work it so unnerving when you have all this pressure from the management. I would much rather use a version control system. Also, can we keep the repository on the web this time so I can work from Stellarbucks when I'm bored?

Matthäus: I have to agree with Sonoko. We can get SVN hosting for next to nothing. We can even get an SVN with SSL for peace of mind.

Yogesh: If I don't have time to set up an in-house SVN, I could at least get you an SVN hosting subscription.

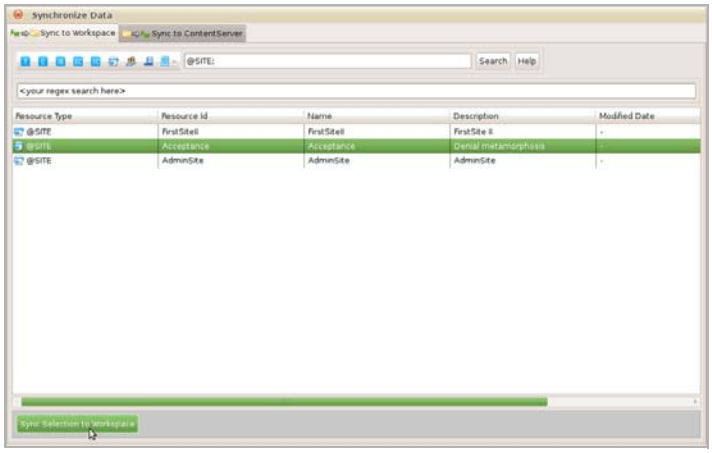
Artie: OK then, I guess we'll go with SVN. Anything else?

Artie and the rest of the development team decide to use SVN to track the resources of their site.

9:42 am – Synchronizing Workspaces With a VCS

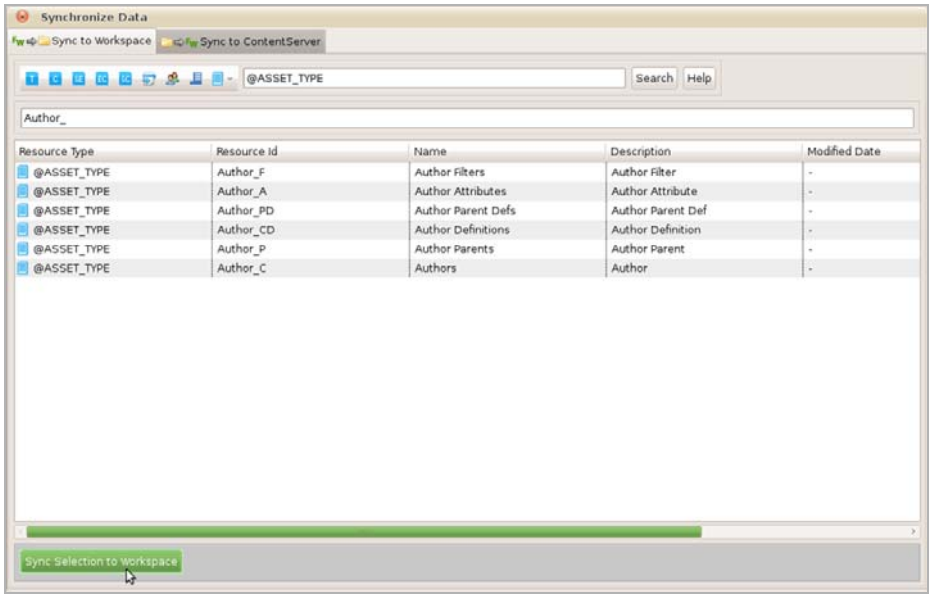
Artie and his team install the Subclipse plug-in from `http://subclipse.tigris.org/`. Now, Artie needs to check in the site and resources he created earlier:

1. Using the CSDT Synchronization screen in Eclipse, Artie accesses the "Sync to Workspace" tab and enters the `@Site` selector in the search field to retrieve a listing of all the sites on his Content Server instance.



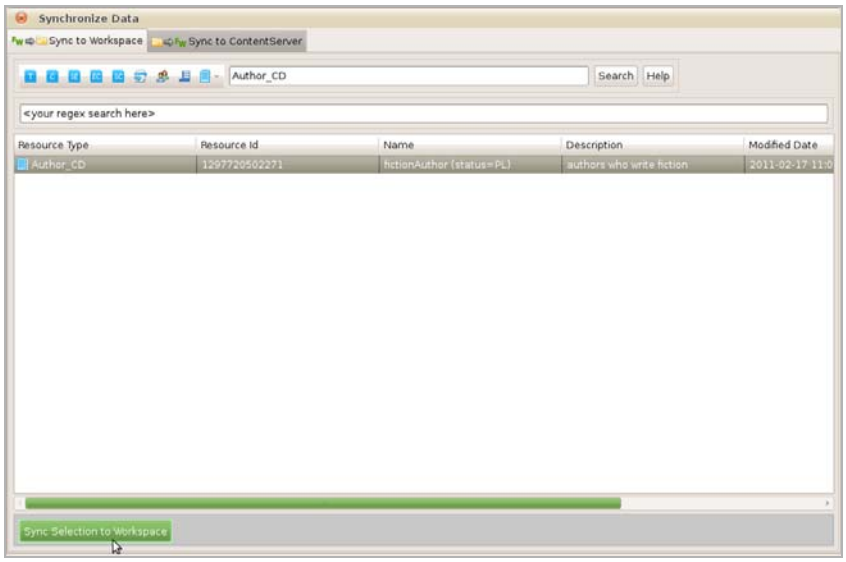
Artie selects the site he created earlier ("Acceptance" site) and clicks the **Sync Selection to Workspace** option to export the site definition from his Content Server instance to his workspace.

2. Next, Artie exports the site's associated flex family to the workspace. He uses the `@ASSET_TYPE` selector to list all the assets on his Content Server instance. To narrow down the results, he uses the `Author_` search string. Artie then selects all listed items and clicks **Sync Selection to Workspace**.



The flex family types are serialized to the workspace, including their type-specific ElementCatalog entries.

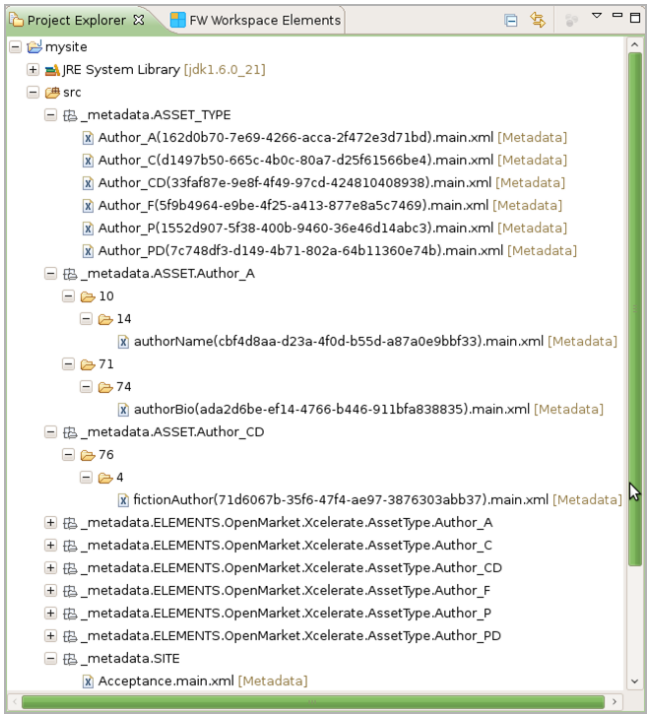
3. Now, Artie exports the flex definition to his workspace. He uses the `Author_CD` selector, which lists all available definitions of that type. In this case, there is only one definition (fictionAuthor).



Note

Artie did not select the flex attributes (Author_A instances) on which the site definition depends because he knows CSDT synchronizes them automatically with the definition.

4. Artie takes a quick look at his workspace in the Eclipse "Project Explorer" view to verify that all his work is there. From top to bottom, he sees the following under the project's `src` folder:
   - `_metadata.ASSET_TYPE` entries for each asset type he synchronized
   - `_metadata.ASSET.Author_A` files for both of the Author_A attributes
   - `_metadata.ASSET.Author_CD` file for the serialized definition
   - `_metadata.ELEMENTS` entries for ElementCatalog entries related to each of the serialized asset types
   - `_metadata.SITE` entry for the site definition

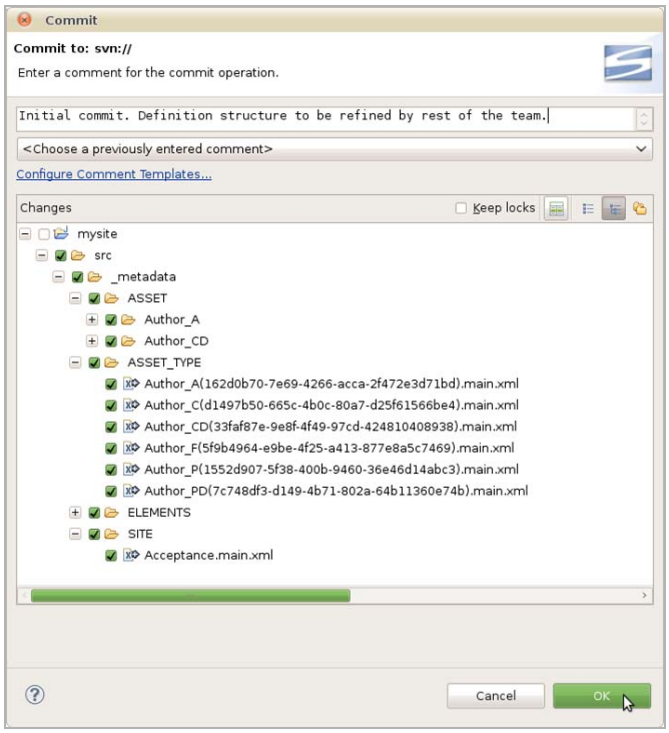


Note

Artie could have looked in the `export/envision/cs_workspace` folder in his Content Server installation directory to see the same data.

Looks like all the resources are in Artie's workspace now. However, this is all on Artie's laptop and the team has no access to it. Time to check-in.

5. Using Subclipse, Artie connects to the development team's SVN repository and shares his CSDT project by committing his main CSDT workspace folder (`src` folder) to the SVN repository.



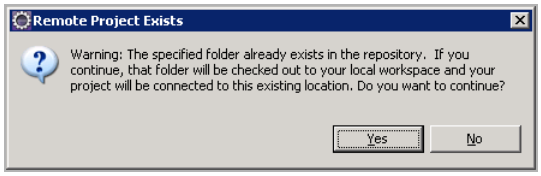
Note

The main CSDT workspace is located under the `src` folder in the Eclipse "Project Explorer" view. Only commit the files that are located inside the `src` folder. All other files are auxiliary local resources and must not be committed.

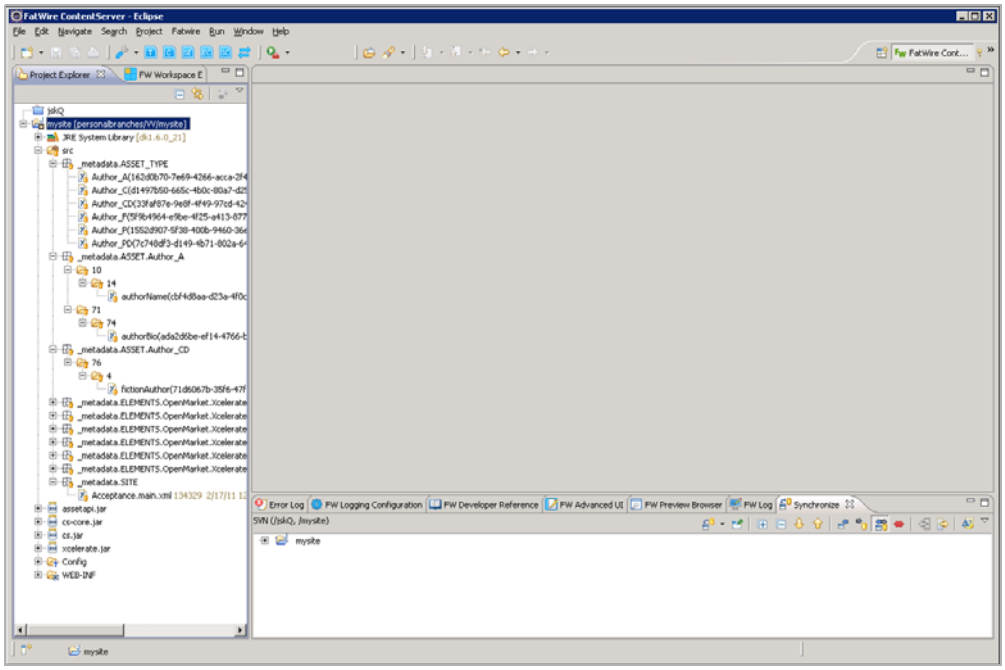
10:12 am – The Other Team Members Synchronize their Workspaces to the SVN Repository

Sonoko and Matthäus just finished setting up their own, individual Eclipse-integrated Content Server instances. They both connect their Eclipse projects to the SVN repository.

Since Artie checked the site and its resources into the SVN repository earlier, Subclipse detects that the target location already exists:



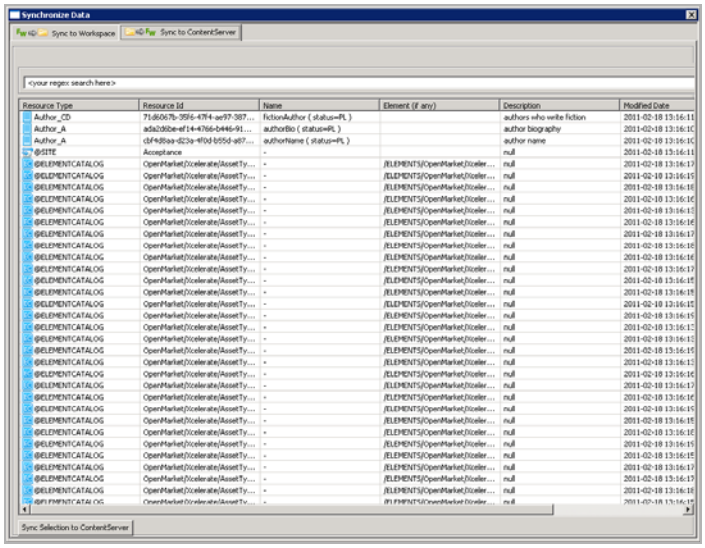
Sonoko and Matthäus both synchronize their main CSDT workspaces with the resources Artie made available in the SVN repository. Those resources are now accessible on both Sonoko and Matthäus' main CSDT workspaces.



However, the resources are not synchronized with Sonoko or Matthäus' Content Server instances yet.

10:18 am – Synchronize the Workspace to the Content Server Instance

Sonoko opens the CSDT Synchronization screen and selects the **Sync to Content Server** tab. All resources contained in Sonoko's main CSDT workspace are listed.

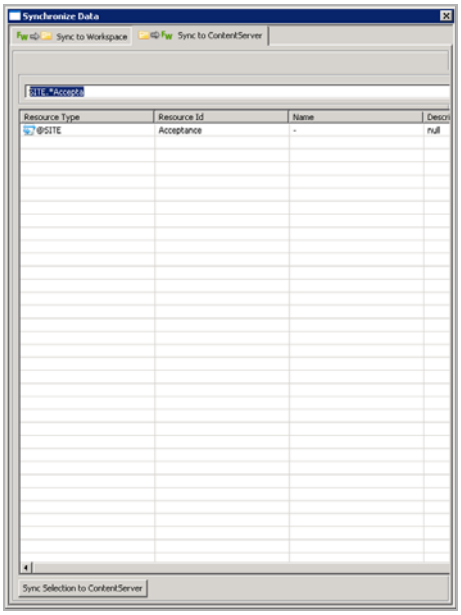


As required, she will first import the site definition, then the flex family, and then the assets, in separate runs as described below:

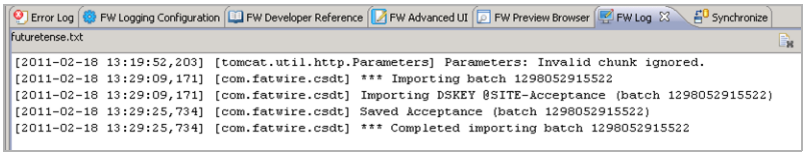
> **Note**
>
> Matthäus will do the same later, when he finishes his meeting with Marketing.

1. Import the site definition (“Acceptance” in this scenario):

   Sonoko imports the site definition first. She narrows down her search by using the `Site.*Accepta` expression in the search field. She then selects the site (“Acceptance”) and synchronizes it to her Content Server instance by clicking **Sync to ContentServer**.

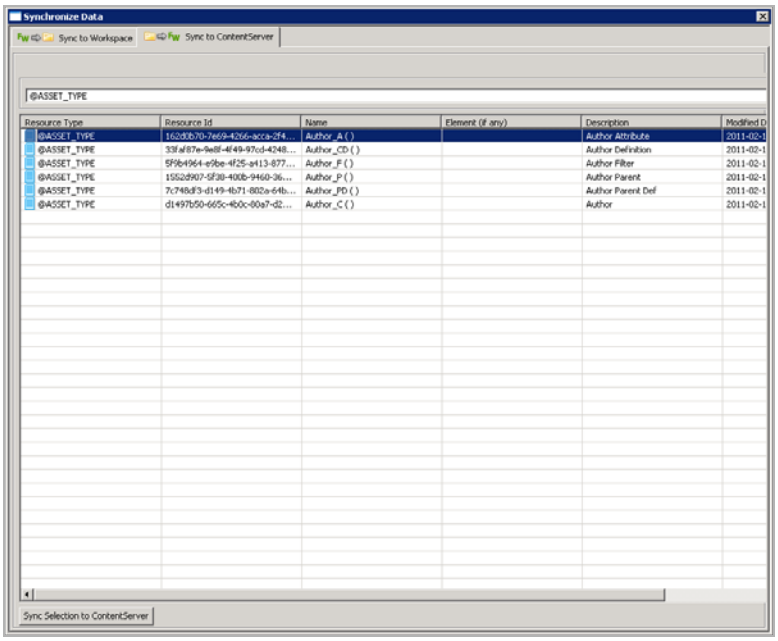


   Using the “FW Log” view, Sonoko verifies that the site is imported successfully:



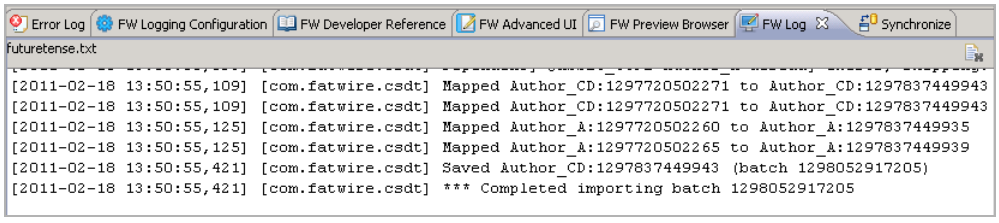
2. Sonoko opens the synchronization screen again, and import the site's flex family, starting with the flex attribute (Author_A in this scenario):

   Since Sonoko did not set up the "Acceptance" site's flex family on her Content Server instance, she must first import the flex attribute (Author_A) to her Content Server instance. Once the flex attribute is imported, she can then synchronize the rest of the asset types that comprise the site's flex family to her Content Server instance.



3. As a final step, Sonoko synchronizes the flex definition, which automatically imports the required attributes.

   The "FW Log" view shows that the local asset identifiers of all the site's resources are re-mapped when imported into the new Content Server instance.

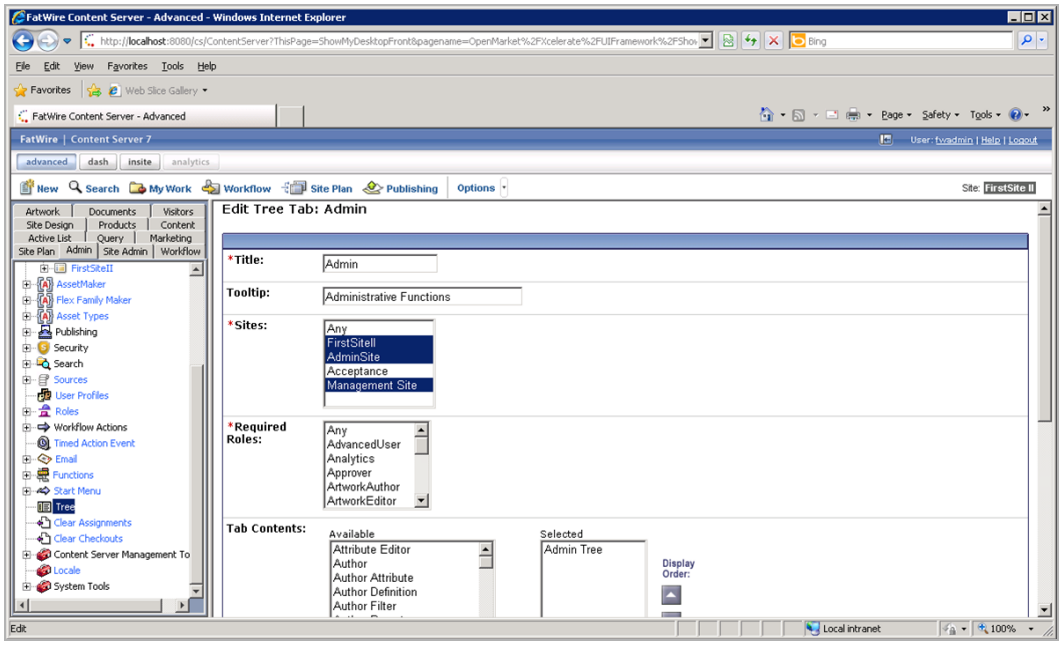


10:21 am – Assign Site Permissions

After synchronizing the resources to her Content Server instance, Sonoko assigns site permissions to herself. These permissions enable her to access the site and its resources from the Advanced interface.

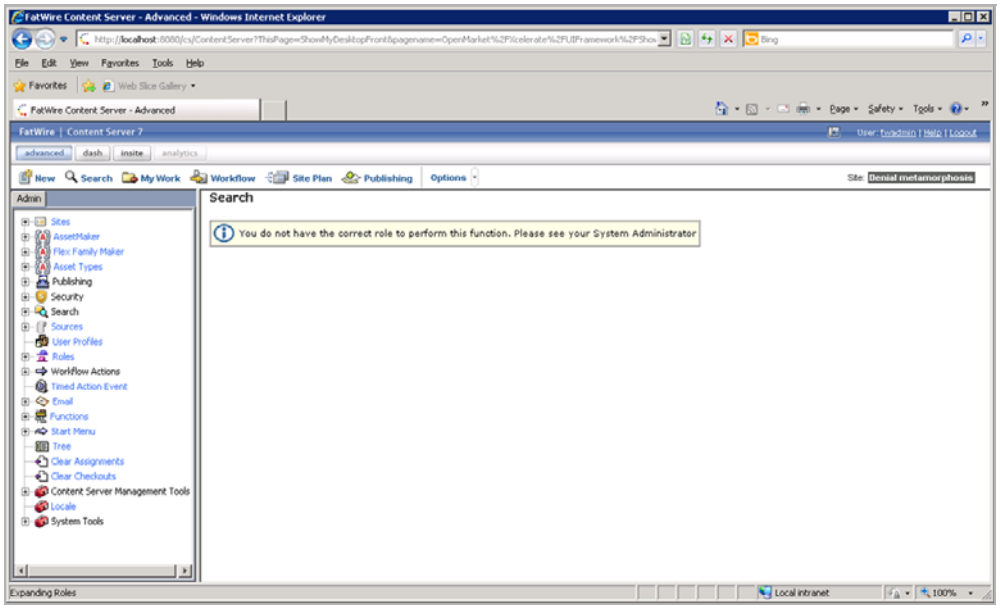
Note

To access the tree applet in the new site, Sonoko must assign at least one tree tab to the site.



10:22 am – The Start Menu Issue

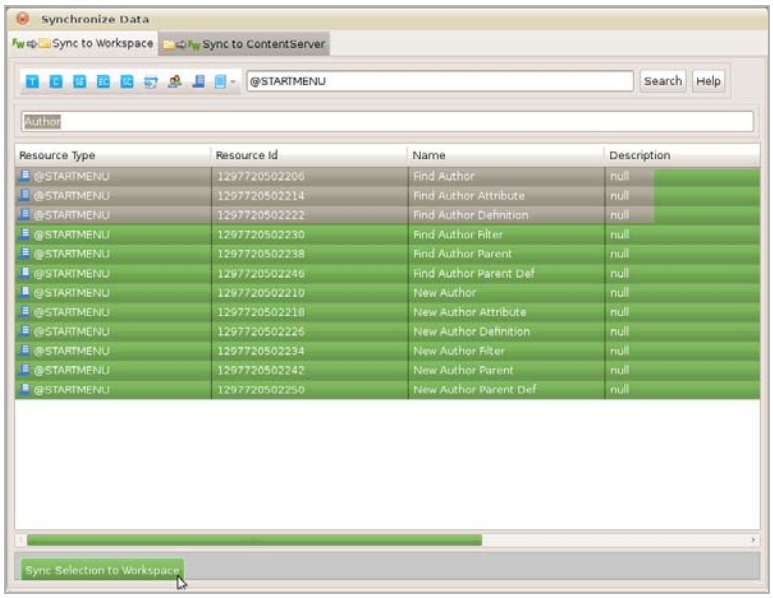
Sonoko logs into the site, and clicks the **New** option. However, she finds there are no start menu items available. Of course, Artie did not check the site's start menu items into the SVN repository.



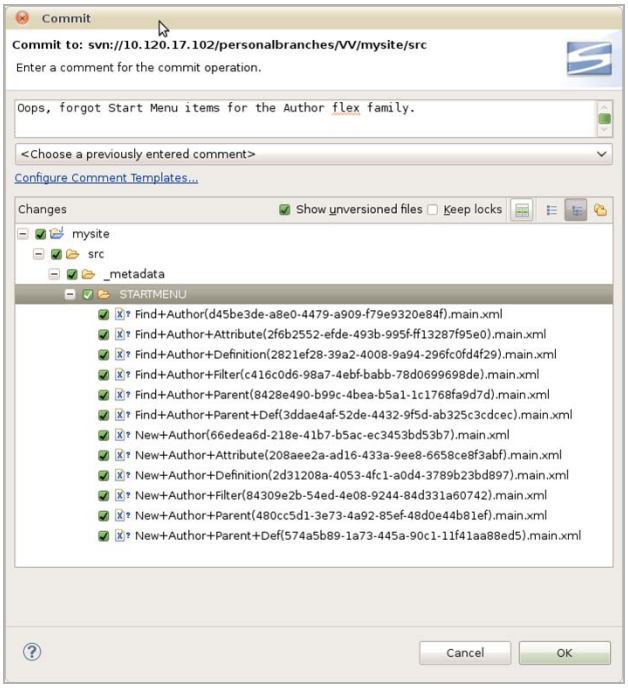
10:24 am – Resolving the Start Menu Issue

Sonoko sends Artie an IM informing him that he forgot to check in the new site's start menu items.

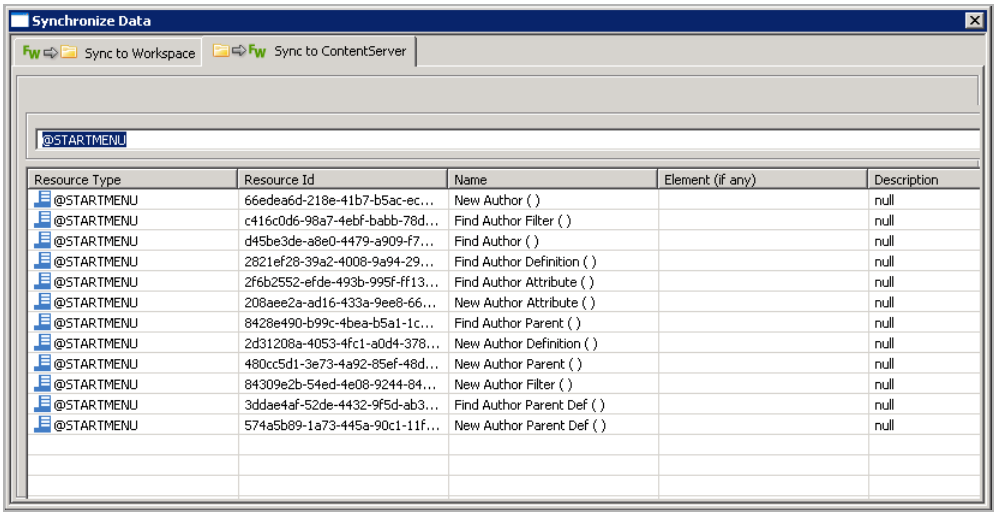
1. Artie synchronizes the site's start menu items to his main CSDT workspace.



2. Artie then checks the site's start menu items into the SVN repository.



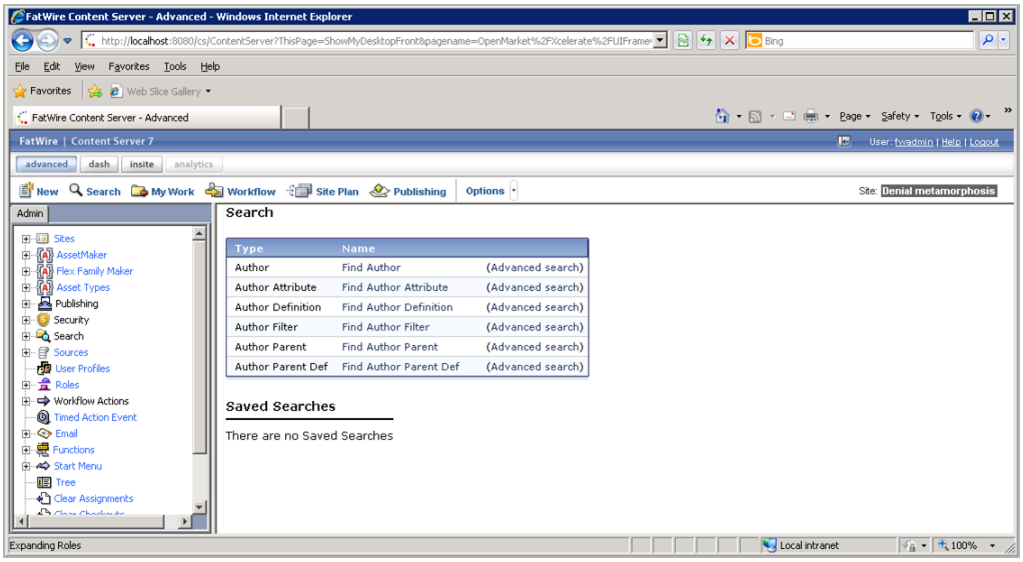
3. Sonoko just got a cup of earl grey tea with two sugars. She comes back to her desk to find that Artie committed the start menu items to the SVN. Sonoko then updates her Eclipse project. She accesses the SVN repository and synchronizes the start menu items to her main CSDT workspace. She then imports those start menu items to her Content Server instance.



Resource Type	Resource Id	Name	Element (if any)	Description
@STARTMENU	66edea6d-218e-41b7-b5ac-ec...	New Author ( )		null
@STARTMENU	c416c0d6-98a7-4ebf-babb-78d...	Find Author Filter ( )		null
@STARTMENU	d45be3de-a8e0-4479-a909-f7...	Find Author ( )		null
@STARTMENU	2821ef28-39a2-4008-9a94-29...	Find Author Definition ( )		null
@STARTMENU	2f6b2552-efde-493b-995f-ff13...	Find Author Attribute ( )		null
@STARTMENU	208aee2a-ad16-433a-9ee8-66...	New Author Attribute ( )		null
@STARTMENU	8428e490-b99c-4bea-b5a1-1c...	Find Author Parent ( )		null
@STARTMENU	2d31208a-4053-4fc1-a0d4-378...	New Author Definition ( )		null
@STARTMENU	480cc5d1-3e73-4a92-85ef-48d...	New Author Parent ( )		null
@STARTMENU	84309e2b-54ed-4e08-9244-84...	New Author Filter ( )		null
@STARTMENU	3ddae4af-52de-4432-9f5d-ab3...	Find Author Parent Def ( )		null
@STARTMENU	574a5b89-1a73-445a-90c1-11f...	New Author Parent Def ( )		null

4. Without restarting her Content Server instance, Sonoko clicks **Search**.

   The start menu items she imported into her Content Server instance are listed:

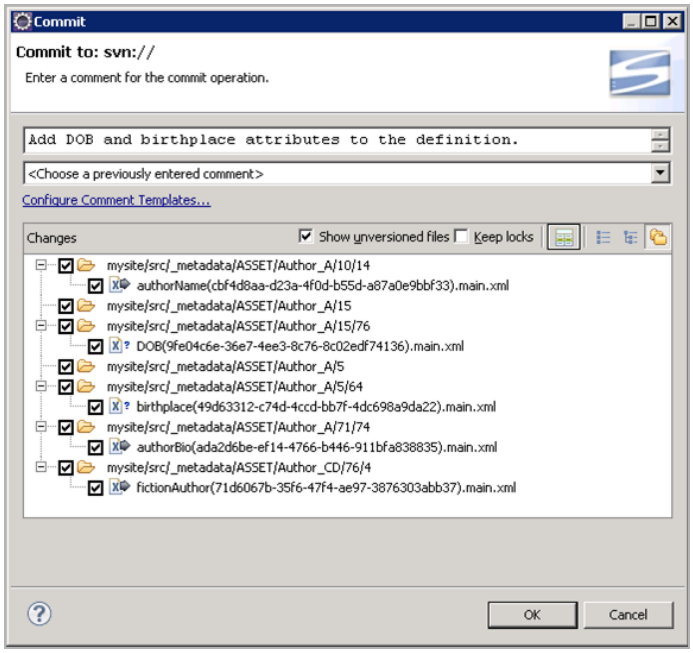


11:17 am – Marketing Requests Changes

```
Subject: Proposed Author Definition Changes
Date: Wed, 16 Feb 2011 11:17:39
From: matthäus.companynone.com
To: Tech-Development
Team,
I just synchronized your changes into my system. As per my meeting
with Marketing, we must have date of birth and birthplace
attributes in the Author Definition. I noticed these attributes do
not exist, so I will add them. Artie, can you review the changes I
make when you have the chance?
Regards,
Matthäus
```

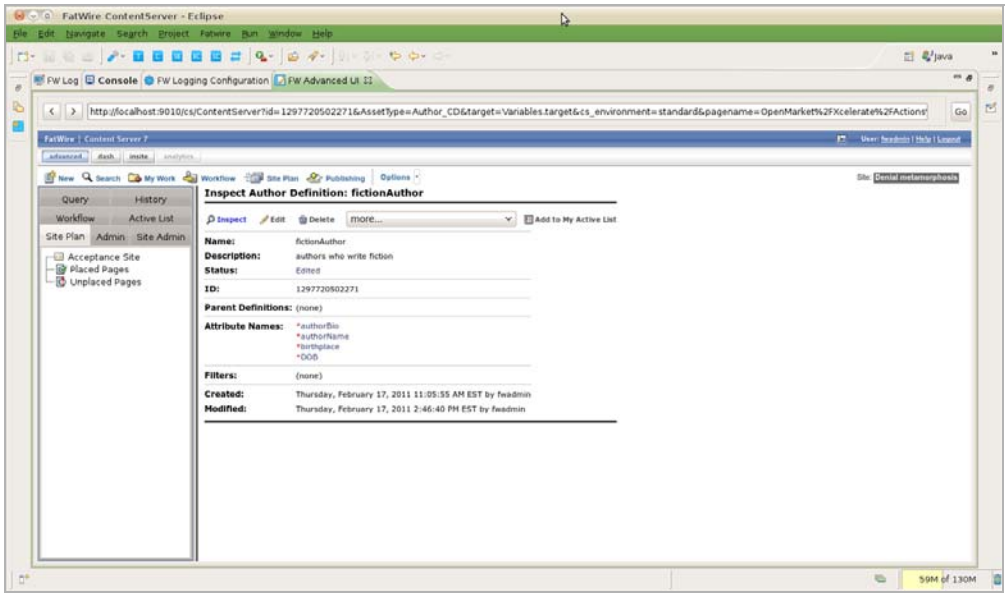
11:22 am – Adding New Attributes to the Author Definition

Matthäus creates the attributes Marketing requested and adds them to the flex definition (Author definition in this scenario). He then exports the new attributes and the flex definition to his main CSDT workspace and commits them to the SVN repository.



11:25 am – Reviewing the Changes to the Site

Artie retrieves the modified Author definition from SVN and imports it into his Content Server instance.

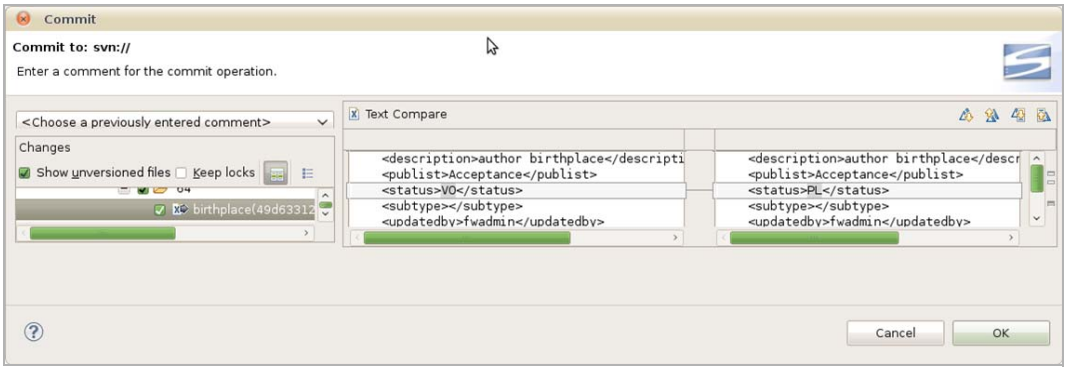


```
Subject: RE: Proposed Author Definition Changes
Date: Wed, 16 Feb 2011 11:37:31
From: artie.companynone.com
To: matthäus.companynone.com
Matthäus,
Thank you for taking care of this. Corporate standards require
us to capitalize the first letter of each subsequent word. I
will delete the birthplace attribute and add birthPlace
instead.
Thank you,
Artie
```

11:44 am – Modifying the Attributes of the Author Definition

1. Artie creates the “birthPlace” attribute and adds it to the flex definition. He then removes the original “birthplace” attribute from the site definition.
2. Artie commits the new attribute and the changes to the Author definition to the SVN repository. He then verifies that the “birthplace” attribute has a status of “VO,” indicating the attribute is voided.

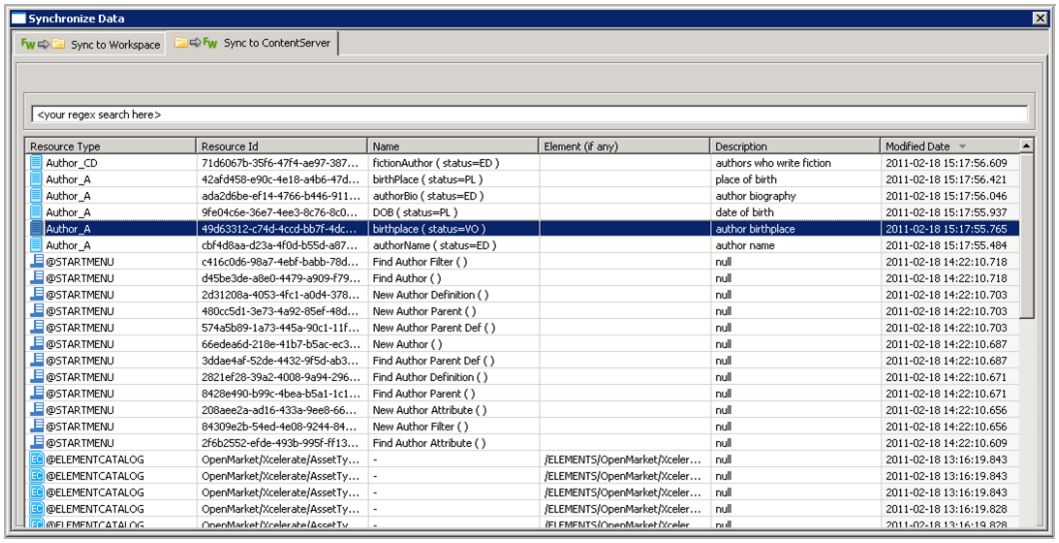
   When Sonoko and Matthäus update their Content Server instances, the “birthplace” attribute will correspondingly be voided on their own workspaces.



11:53 am – The Team Updates Their Workspaces and Content Server Instances

1. Sonoko and Matthäus update their main CSDT workspaces with the resources Artie checked in to the SVN repository.
2. They then import the resources in their workspaces to their Content Server instances by opening the “Synchronize to Content Server” tab. For convenience, they sorted by the “Modified Date” column so the most recent changes are shown on top.

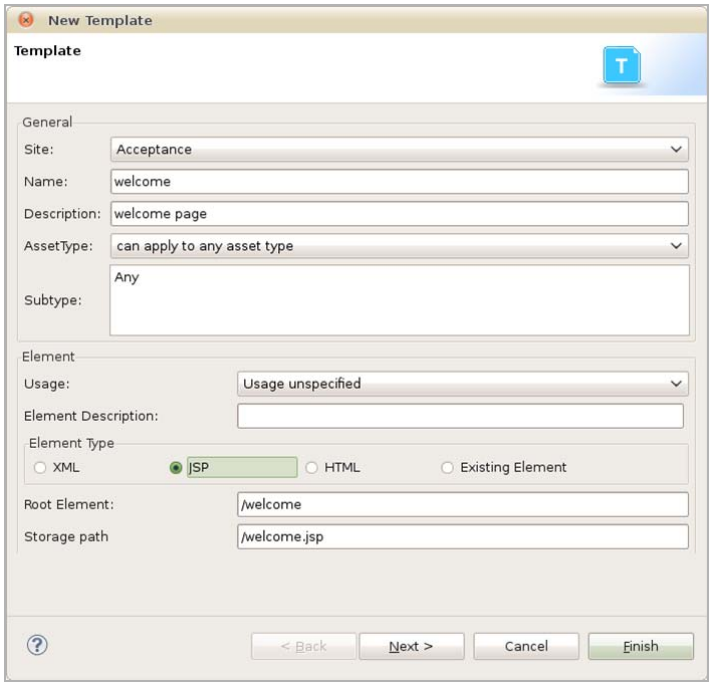
   Any voided attributes (such as the “birthplace” attribute Artie voided) show a status hint (`status=VO`) in the “Name” column.



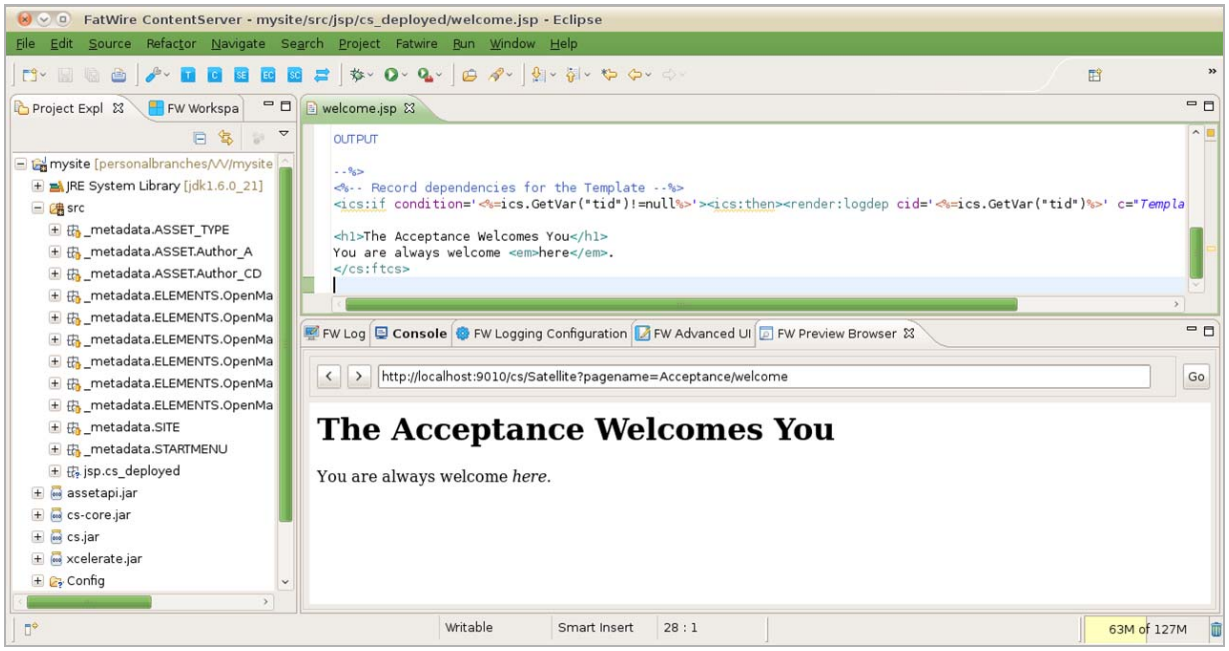
3. Sonoko and Matthäus import these changes from their workspaces to their Content Server instances. Their workspaces and Content Server instances are now up to date.

12:27 pm – The Team Creates a Template Asset for the Site

1. (12:27 pm) Matthäus creates a Template asset for the site's "Welcome" page.

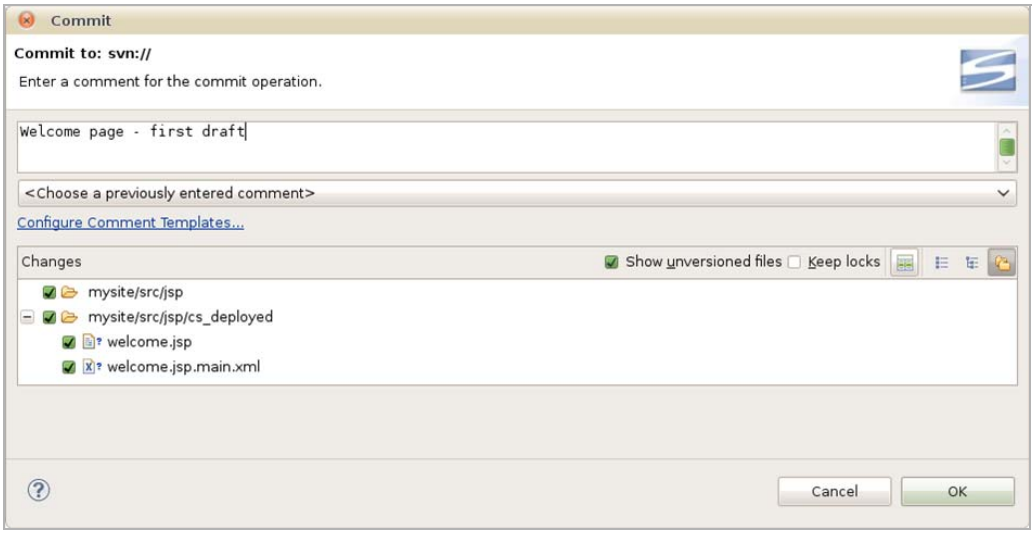


2. (12:34 pm) Matthäus edits the Template asset and previews the changes in the "FW Preview Browser" view. As soon as he saves the changes made to the Template asset's JSP, he uses the **Ctrl-r** keyboard command to refresh the preview browser.

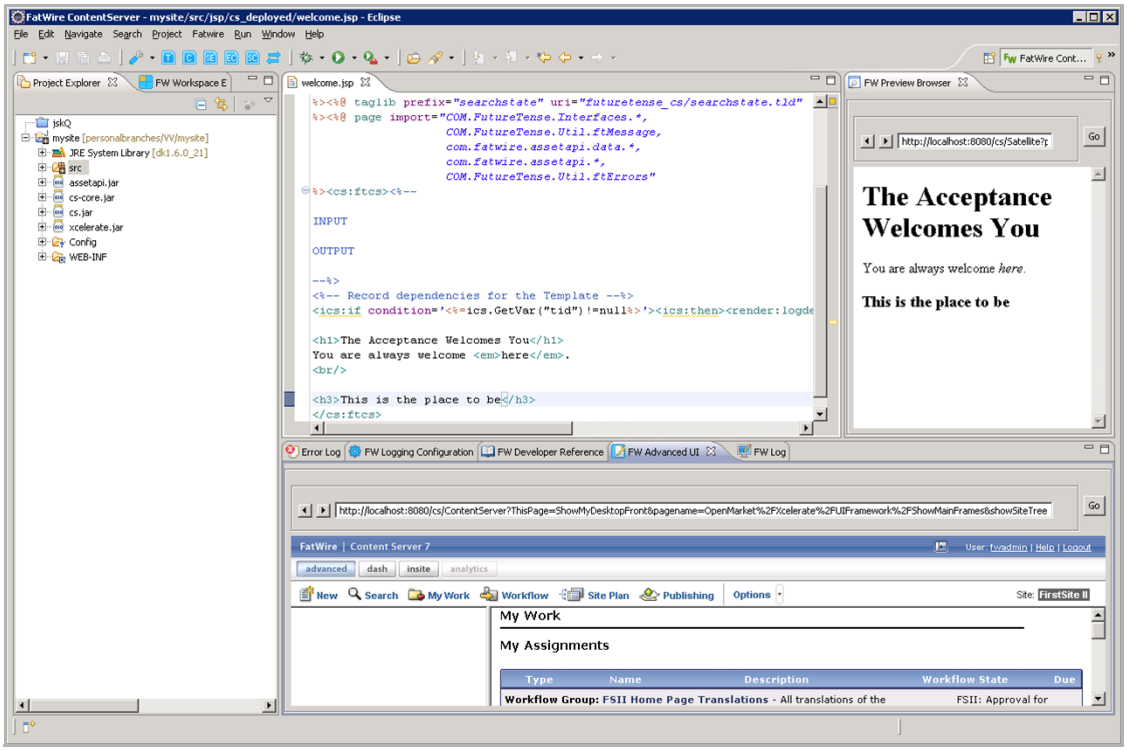


3. (12:39 pm) Matthäus commits the Template's `.jsp` and `.main.xml` files to the SVN repository.

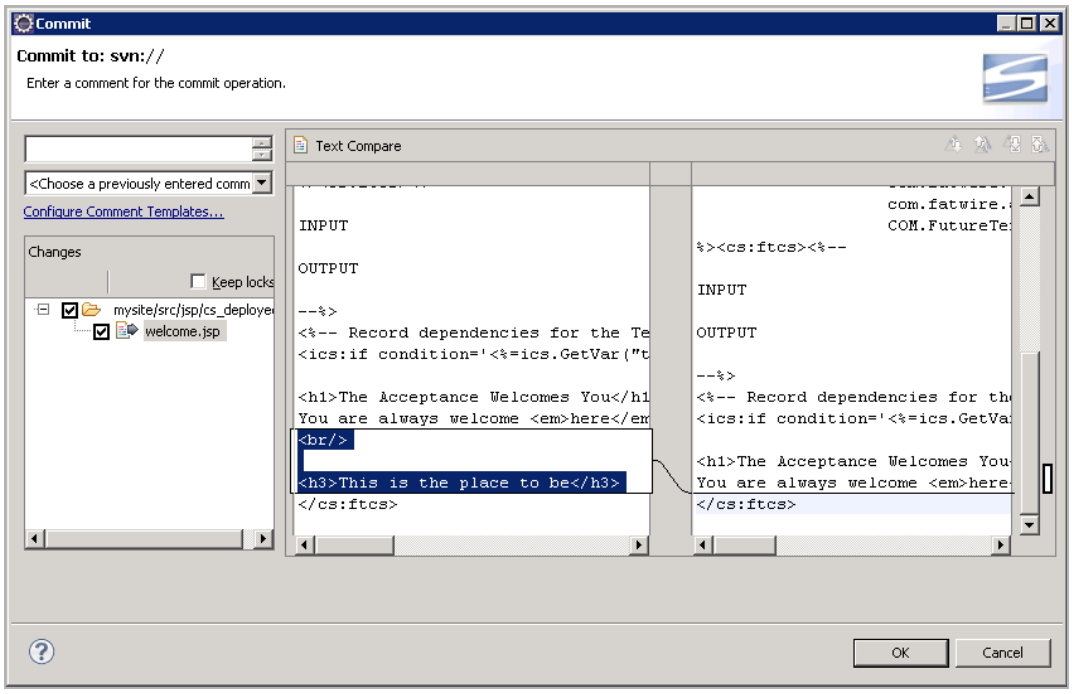
   Subclipse finds all changes to the project and brings those changes to the attention of the developer. Since the only new asset was the Template asset, Matthäus is able to deduce that the `.main.xml` file is the Template's metadata and the JSP file is the Template's code.



4. (12:44 pm) Sonoko makes some touch ups to the Template's JSP file in her own workspace.



5. Sonoko reviews the changes to the JSP file and then commits those changes to the SVN.



Note

If another team member were to modify and check in this file at the same time as Sonoko, SVN would indicate to Sonoko that another version of the file is already checked in. She would then be able to integrate those changes with her own to avoid inadvertent overwrites.

Three Days Later... Deployment

Yogesh uses the command-line tool to deploy the site. For information about the commands used in this section, see Chapter 7, "Command-Line Tool."

9:32 am – Preparing for Deployment

Yogesh finally got around to setting up the test environment and is preparing to deploy the current build using the command-line tool. He installed a Content Server system on hardware that matches the environment used in production.

To test the CSDT import before adding it to a fully-automated nightly script

1. Using the command-line tool, Yogesh checks the "Acceptance" site and its resources out of SVN and into the workspace of the target Content Server instance.

   Command:

```
## go to the workspace location under export/envision/
   cs_workspace in the CS install directory
## create if not there
/home$ mkdir /opt/cs/export/envision/cs_workspace
/home$ cd  /opt/cs/export/envision/cs_workspace

## checkout site from svn
/opt/cs/export/envision/cs_workspace$ svn checkout svn://
   yoursvnhost/projects/mysite/src
```

   Output:

```
A    mysite/src
A    mysite/src/_metadata
A    mysite/src/_metadata/ASSET
A    mysite/src/_metadata/ASSET/Author_A
A    mysite/src/_metadata/ASSET/Author_A/10
A    mysite/src/_metadata/ASSET/Author_A/10/14
A    mysite/src/_metadata/ASSET/Author_A/10/14/authorName(cbf4d8aa-
d23a-4f0d-b55d-a87a0e9bbf33).main.xml
A    mysite/src/_metadata/ASSET/Author_A/11
A    mysite/src/_metadata/ASSET/Author_A/11/79
A    mysite/src/_metadata/ASSET/Author_A/11/79/birthPlace(42afd458-
e90c-4e18-a4b6-47d322b46414).main.xml
A    mysite/src/_metadata/ASSET/Author_A/5
A    mysite/src/_metadata/ASSET/Author_A/5/64
A    mysite/src/_metadata/ASSET/Author_A/5/64/birthplace(49d63312-
c74d-4ccd-bb7f-4dc698a9da22).main.xml
A    mysite/src/_metadata/ASSET/Author_A/15
A    mysite/src/_metadata/ASSET/Author_A/15/76
A    mysite/src/_metadata/ASSET/Author_A/15/76/DOB(9fe04c6e-36e7-4ee3-
8c76-8c02edf74136).main.xml
A    mysite/src/_metadata/ASSET/Author_A/71
A    mysite/src/_metadata/ASSET/Author_A/71/74
A    mysite/src/_metadata/ASSET/Author_A/71/74/authorBio(ada2d6be-
ef14-4766-b446-911bfa838835).main.xml
A    mysite/src/_metadata/ASSET/Author_CD
```

```
A    mysite/src/_metadata/ASSET/Author_CD/76
A    mysite/src/_metadata/ASSET/Author_CD/76/4
A    mysite/src/_metadata/ASSET/Author_CD/76/4/fictionAuthor(71d6067b-
35f6-47f4-ae97-3876303abb37).main.xml
A    mysite/src/_metadata/ASSET_TYPE
A    mysite/src/_metadata/ASSET_TYPE/Author_F(5f9b4964-e9be-4f25-a413-
877e8a5c7469).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_P(1552d907-5f38-400b-9460-
36e46d14abc3).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_A(162d0b70-7e69-4266-acca-
2f472e3d71bd).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_CD(33faf87e-9e8f-4f49-
97cd-424810408938).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_PD(7c748df3-d149-4b71-
802a-64b11360e74b).main.xml
A    mysite/src/_metadata/ASSET_TYPE/Author_C(d1497b50-665c-4b0c-80a7-
d25f61566be4).main.xml
A    mysite/src/_metadata/STARTMENU
A    mysite/src/_metadata/STARTMENU/Find+Author+Attribute(2f6b2552-
efde-493b-995f-ff13287f95e0).main.xml
...
```

2. Yogesh runs a workspace listing (`cmd=listds`) to verify that the site and all of its resources will be imported into the Content Server instance. He uses the `@ALL_ASSETS` and `@ALL_NONASSETS` selectors to generate listings of all asset and non-asset resources in the workspace:

   - **Command** to use the `@ALL_ASSETS` selector:

     ```
     /opt/cs/export/envision/cs_workspace$ export
       CLASSPATH=csdt-client-1.0.2.jar
     /opt/cs/export/envision/cs_workspace$ java
       com.fatwire.csdt.client.main.CSDT
     http://localhost:9010/cs/ContentServer username=fwadmin
       password=xceladmin resources=@ALL_ASSETS cmd=listds
     ```

     Output:

     ```
     Resource Type |||    Resource Id   |||    Name     |||
     Description   ||| Modified On
     ----------------------------------------------------------------
     ----------------------
     Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName (
     status=ED ) ||| author name ||| 2011-02-17 15:26:34.000
     Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace (
     status=PL ) ||| place of birth ||| 2011-02-17 15:26:34.000
     Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB (
     status=PL ) ||| date of birth ||| 2011-02-17 15:26:34.000
     Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 |||
     fictionAuthor ( status=ED ) ||| authors who write fiction |||
     2011-02-17 15:26:34.000
     Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio (
     status=ED ) ||| author biography ||| 2011-02-17 15:26:34.000
     Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace (
     status=VO ) ||| author birthplace ||| 2011-02-17 15:12:43.000
     Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome
     (Typeless status=PL) ||| welcome page ||| 2011-02-17 23:18:18.000
     ```

- **Command** to use the `@ALL_NONASSETS` selector:

```
/opt/cs/export/envision/cs_workspace$ export
  CLASSPATH=csdt-client-1.0.2.jar
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
  password=xceladmin resources=@ALL_NONASSETS cmd=listds
```

Output:

```
Resource Type |||    Resource Id   |||    Name     |||
Description   ||| Modified On
------------------------------------------------------------------
----------------------
@STARTMENU ||| 66edea6d-218e-41b7-b5ac-ec3453bd53b7 ||| New Author
( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| c416c0d6-98a7-4ebf-babb-78d0699698de ||| Find
Author Filter ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 162d0b70-7e69-4266-acca-2f472e3d71bd ||| Author_A
( ) ||| Author Attribute ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2821ef28-39a2-4008-9a94-296fc0fd4f29 ||| Find
Author Definition ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| d45be3de-a8e0-4479-a909-f79e9320e84f ||| Find
Author ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 2f6b2552-efde-493b-995f-ff13287f95e0 ||| Find
Author Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| 7c748df3-d149-4b71-802a-64b11360e74b ||| Author_PD
( ) ||| Author Parent Def ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 208aee2a-ad16-433a-9ee8-6658ce8f3abf ||| New Author
Attribute ( ) ||| null ||| 2011-02-18 11:02:23.000
@STARTMENU ||| 8428e490-b99c-4bea-b5a1-1c1768fa9d7d ||| Find
Author Parent ( ) ||| null ||| 2011-02-18 11:02:23.000
@ASSET_TYPE ||| d1497b50-665c-4b0c-80a7-d25f61566be4 ||| Author_C
( ) ||| Author ||| 2011-02-18 11:02:23.000
…
```

3. Yogesh then makes sure all necessary asset types will be imported by using the `@ASSET_TYPE:*` selector:

Command:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@ASSET_TYPE:*' cmd=listds
```

Output:

```
Resource Type |||    Resource Id   |||    Name     |||   Description
||| Modified On
------------------------------------------------------------------------
-----------------
Author_A ||| cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 ||| authorName (
status=ED ) ||| author name ||| 2011-02-17 15:26:34.000
Author_A ||| 42afd458-e90c-4e18-a4b6-47d322b46414 ||| birthPlace (
status=PL ) ||| place of birth ||| 2011-02-17 15:26:34.000
```

```
Author_A ||| 9fe04c6e-36e7-4ee3-8c76-8c02edf74136 ||| DOB ( status=PL
) ||| date of birth ||| 2011-02-17 15:26:34.000
Author_CD ||| 71d6067b-35f6-47f4-ae97-3876303abb37 ||| fictionAuthor (
status=ED ) ||| authors who write fiction ||| 2011-02-17 15:26:34.000
Author_A ||| ada2d6be-ef14-4766-b446-911bfa838835 ||| authorBio (
status=ED ) ||| author biography ||| 2011-02-17 15:26:34.000
Author_A ||| 49d63312-c74d-4ccd-bb7f-4dc698a9da22 ||| birthplace (
status=VO ) ||| author birthplace ||| 2011-02-17 15:12:43.000
Template ||| 89b05c0f-227b-4dcb-961e-2ab6e6af2dae ||| welcome
(Typeless status=PL) ||| welcome page ||| 2011-02-17 23:18:18.000
```

4. Yogesh notes that all necessary resources for the site will be imported into the build.

10:04 am – Deploying the Site and its Resources

Using the command-line tool, Yogesh runs the import sequence.

1. First, he imports the site:

   Command:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9999/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@SITE:Acceptance' cmd=import
```

   Output:

```
 *** Importing batch 1297868431526
Importing DSKEY @SITE-Acceptance (batch 1297868431526)
Saved Acceptance (batch 1297868431526)
*** Completed importing batch 1297868431526
```

2. Then, the flex family:

   Command:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9999/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@ASSET_TYPE:*' cmd=import
```

   Output:

```
          *** Importing batch 1298064678765
Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd
(batch 1298064678765)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/
Author_A/LoadTree (batch 1298064678765)
Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadTree (batch
1298064678765)
…
```

3. Next, the assets:

   Command:

```
/opt/cs/export/envision/cs_workspace java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9999/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@ALL_ASSETS' cmd=import
```

   Output:

```
*** Importing batch 1298064679760
Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071437 (batch 1298064679760)
Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071441 (batch 1298064679760)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071445 (batch 1298064679760)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch
1298064679760)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071449 (batch 1298064679760)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1295889071453 (batch 1298064679760)
Importing DSKEY Author_A-49d63312-c74d-4ccd-bb7f-4dc698a9da22 (batch
1298064679760)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1295889071460 (batch 1298064679760)
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch
1298064679760)
Saved Template:1295889071461 (batch 1298064679760)
*** Completed importing batch 1298064679760
```

4. Since this is a delivery install, start menu items are optional. However, Yogesh imports the start menu items because he wants to use the Advanced interface to verify that all of the resources are successfully imported.

Command:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9999/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@STARTMENU:*' cmd=import
```

Output:

```
          *** Importing batch 1298064681075
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch
1298064681075)
Saved 1297720502210 (batch 1298064681075)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch
1298064681075)
Saved 1297720502230 (batch 1298064681075)
Importing DSKEY @STARTMENU-2821ef28-39a2-4008-9a94-296fc0fd4f29 (batch
1298064681075)
Saved 1297720502222 (batch 1298064681075)
Importing DSKEY @STARTMENU-d45be3de-a8e0-4479-a909-f79e9320e84f (batch
1298064681075)
Saved 1297720502206 (batch 1298064681075)
Importing DSKEY @STARTMENU-2f6b2552-efde-493b-995f-ff13287f95e0 (batch
1298064681075)
Saved 1297720502214 (batch 1298064681075)
Importing DSKEY @STARTMENU-208aee2a-ad16-433a-9ee8-6658ce8f3abf (batch
1298064681075)
Saved 1297720502218 (batch 1298064681075)
Importing DSKEY @STARTMENU-8428e490-b99c-4bea-b5a1-1c1768fa9d7d (batch
1298064681075)
Saved 1297720502238 (batch 1298064681075)
Importing DSKEY @STARTMENU-2d31208a-4053-4fc1-a0d4-3789b23bd897 (batch
1298064681075)
Saved 1297720502226 (batch 1298064681075)
Importing DSKEY @STARTMENU-480cc5d1-3e73-4a92-85ef-48d0e44b81ef (batch
1298064681075)
Saved 1297720502242 (batch 1298064681075)
Importing DSKEY @STARTMENU-84309e2b-54ed-4e08-9244-84d331a60742 (batch
1298064681075)
*** Completed importing batch 1298064681075
```

10:55 am – The Deployment is Successful

Yogesh concludes that the import sequence was successful. He plans to automated daily installs on this system by writing the following script:

```
## Reinstall ContentServer to start with a clean slate.
## Optionally skip this and just do an update
Reinstall_CS()

## Bring in the latest source from SVN
SVN_Update()

## Prepare for import: compile any Java code such as url
   assemblers and flex filters, etc
## Prepare the database with any custom settings, etc.
preImport()

## Run the CSDT import sequence
CSDT_Import()

## Run the test suite – sanity, performance, acceptance tests
runTestSuite()

## Report results to the team by email so they know about any
   failures first thing in the morning
runReports()
```

The script will run as a cron job at five past midnight every night.

Appendix B

Using the Command-line Tool to Create Reusable Modules

CSDT provides the ability to reuse and share resources in the form of modules. Modules are workspaces that are not site-specific and contain resources such as Templates, flex families, and ElementCatalog entries. Unlike the standard export/import functionality where assets are added to sites using natural mappings, modules typically utilize site overriding so they can be imported into any site you designate.

This appendix contains the following section:

Creating a Reusable Module

Artie has a flex family with a flex definition that he wants to reuse in other sites. He also has a Template asset associated with the flex definition. In the following scenario, Artie will create a module containing these resources. This scenario uses the command-line tool to create a module containing the resources Artie and his team developed in Appendix A, "Development Team Integration Use Case."

Note

To use the command-line tool, Artie must specify the user name and password of a general administrator in each command he executes. This user must be a member of the `RestAdmin` group. In this scenario, Artie uses `fwadmin/xceladmin`.

Step I. List the Resources in the Content Server Instance

Artie uses the command-line tool to browse his Content Server instance. He uses the `resources=@ALL_ASSETS` and the `fromSites=Acceptance` selectors to list all the assets of the "Acceptance" site. The command Artie uses is `listcs`, which lists all the resources on his Content Server instance.

Command:

```
/opt/cs/export/envision/cs_workspace$ export CLASSPATH=csdt-
  client-1.0.2.jar
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
  password=xceladmin resources=@ALL_ASSETS
  fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type |||    Resource Id   |||    Name     |||   Description
||| Modified On
-------------------------------------------------------------------------
--------------
Author_CD ||| 1297720502271 ||| fictionAuthor (status=ED) ||| authors who
write fiction ||| 2011-02-17 15:10:41
Author_A ||| 1297720502260 ||| authorName (status=ED) ||| author name |||
2011-02-17 14:46:40
Author_A ||| 1297720502265 ||| authorBio (status=ED) ||| author biography
||| 2011-02-17 14:46:40
Author_A ||| 1297720502289 ||| 1297720502289 (status=VO) ||| author
birthplace ||| 2011-02-17 15:12:35
Author_A ||| 1297720502293 ||| DOB (status=PL) ||| date of birth |||
2011-02-17 14:46:40
Author_A ||| 1297720502305 ||| birthPlace (status=PL) ||| place of birth
||| 2011-02-17 15:10:22
Template ||| 1297720502331 ||| welcome (Typeless, status=ED) ||| welcome
page ||| 2011-02-17 23:18:18
```

Artie notes that there are five Author_A flex attribute instances (one of which is voided), one Author_CD flex definition, and a Template asset.

Step II. List Start Menu Items

Artie further uses the command-line tool to browse for any start menu items that are assigned to the “Acceptance” site.

Command:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@STARTMENU:*'
  fromSites=Acceptance cmd=listcs
```

Output:

```
Resource Type |||    Resource Id   |||    Name     |||   Description
||| Modified On
--------------------------------------------------------------------------
---------------
@STARTMENU ||| 1297720502206 ||| Find Author ||| null ||| -
@STARTMENU ||| 1297720502214 ||| Find Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502222 ||| Find Author Definition ||| null ||| -
@STARTMENU ||| 1297720502230 ||| Find Author Filter ||| null ||| -
@STARTMENU ||| 1297720502238 ||| Find Author Parent ||| null ||| -
@STARTMENU ||| 1297720502246 ||| Find Author Parent Def ||| null ||| -
@STARTMENU ||| 1297720494070 ||| Find CSElement, FirstSiteII ||| Find
CSElement ||| -
@STARTMENU ||| 1297720494086 ||| Find Page, FirstSiteII ||| Find Page |||
-
@STARTMENU ||| 1297720494078 ||| Find SiteEntry, FirstSiteII ||| Find
SiteEntry ||| -
@STARTMENU ||| 1297720494066 ||| Find Template, FirstSiteII ||| Find
Template ||| -
@STARTMENU ||| 1297720502210 ||| New Author ||| null ||| -
@STARTMENU ||| 1297720502218 ||| New Author Attribute ||| null ||| -
@STARTMENU ||| 1297720502226 ||| New Author Definition ||| null ||| -
@STARTMENU ||| 1297720502234 ||| New Author Filter ||| null ||| -
@STARTMENU ||| 1297720502242 ||| New Author Parent ||| null ||| -
@STARTMENU ||| 1297720502250 ||| New Author Parent Def ||| null ||| -
@STARTMENU ||| 1297720501427 ||| New CSElement ||| null ||| -
@STARTMENU ||| 1297720494052 ||| New Page, FirstSiteII ||| New Page ||| -
@STARTMENU ||| 1297720501431 ||| New SiteEntry ||| null ||| -
@STARTMENU ||| 1297720501435 ||| New Template ||| null ||| -
```

Step III. Export All Resources to the Desired Workspace

Artie wants to create a module using all of the resources listed in steps I and II. He runs the following command to export all of the resources, at one time, into the specified workspace:

```
/opt/cs/export/envision/cs_workspace$ java
  com.fatwire.csdt.client.main.CSDT
http://localhost:9010/cs/ContentServer username=fwadmin
  password=xceladmin 'resources=@STARTMENU:*;@ALL_ASSETS'
  fromSites=Acceptance cmd=export datastore=authorModule
```

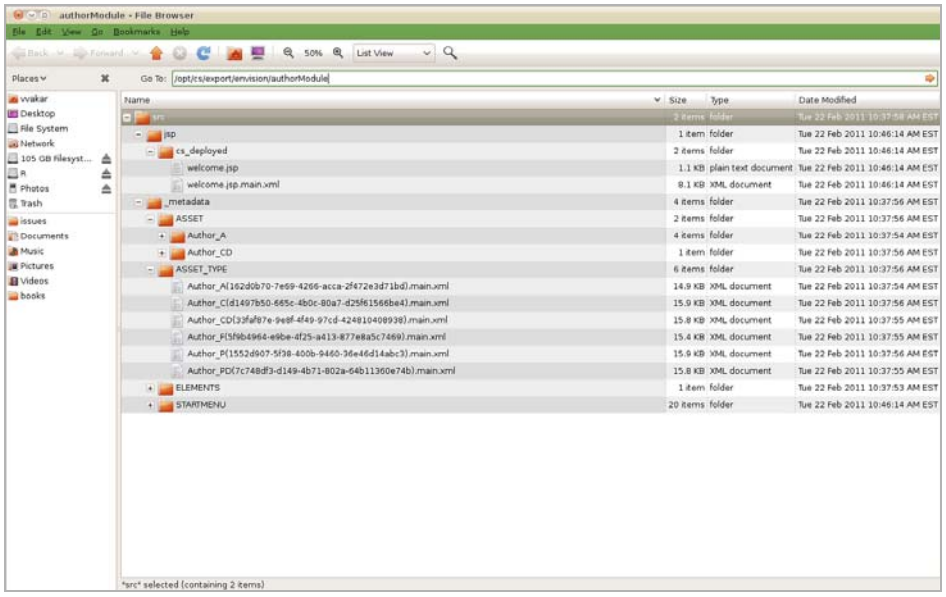
Output:

```
*** Exporting batch 1298385511005
Exporting ASSETDATA Author_CD:1297720502271 (batch 1298385511005)
Exporting ASSETDATA Author_A:1297720502260 (batch 1298385511005)
Exporting ASSET_TYPE Author_A (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
LoadSiteTree (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
AppendSelectDetails (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
AppendSelectDetailsSE (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd
(batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
IndexReplace (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
IndexCreateVerity (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
ContentDetails (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
ContentForm (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
PostUpdate (batch 1298385511005)
Exporting ELEMENTCATALOG OpenMarket/Xcelerate/AssetType/Author_A/
PreUpdate (batch 1298385511005)
...
```

All asset types for the flex family are included in the export. In addition, all elements belonging to those types are included as well. This information, although not usually modified, is necessary in order to make the module Artie is creating reusable on other Content Server instances.

Step IV. Inspect the Module's Content

Artie inspects the authorModule workspace on his file system.



Artie notes that the Template asset, flex family members, asset types, and start menu items were all exported to the workspace on his file system.

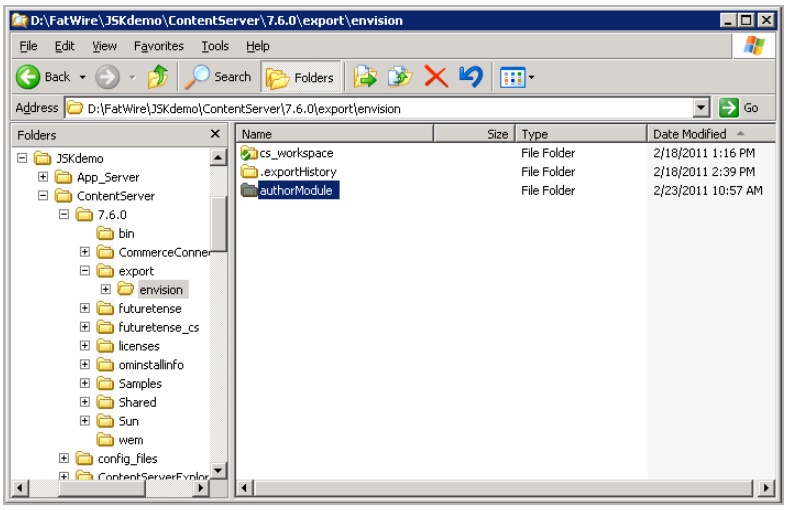
Step V. Archive the Module

Artie creates a `.zip` file archive of the authorModule workspace and saves it.

Step VI. Import the Module to a Content Server Instance

Artie decides to import the module into the FirstSiteII sample site.

1. Artie unzips the module into the workspace location of the target Content Server instance.



2. Using the command-line tool, Artie imports the asset types and start menu items into the target Content Server instance.

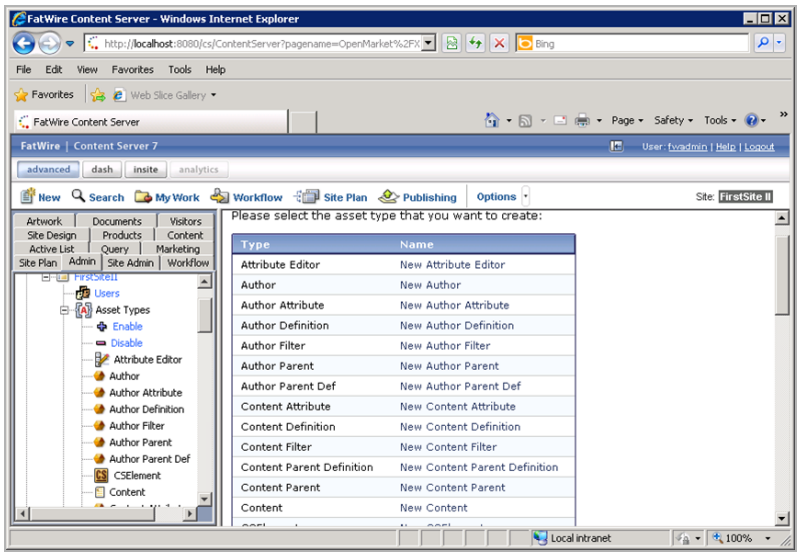
Command:

```
D:\FatWire\JSKdemo\ContentServer>java com.fatwire.csdt
  .client.main.CSDT
http://localhost:8080/cs/ContentServer username=fwadmin
  password=xceladmin resources=@ALL_NONASSETS cmd=import
  datastore=authorModule toSites=FirstSiteII
```

Output:

```
          *** Importing batch 1298052933085
Importing DSKEY @STARTMENU-4340b65d-a9e4-4131-ac7f-51185a79b18d (batch
1298052933085)
Saved 1297720494070 (batch 1298052933085)
Importing DSKEY @STARTMENU-0a2decd4-b6be-418c-9992-a4332480bb20 (batch
1298052933085)
Saved 1297720501435 (batch 1298052933085)
Importing DSKEY @STARTMENU-66edea6d-218e-41b7-b5ac-ec3453bd53b7 (batch
1298052933085)
Saved 1297720502210 (batch 1298052933085)
Importing DSKEY @STARTMENU-c416c0d6-98a7-4ebf-babb-78d0699698de (batch
1298052933085)
Saved 1297720502230 (batch 1298052933085)
Importing DSKEY @ASSET_TYPE-162d0b70-7e69-4266-acca-2f472e3d71bd
(batch 1298052933085)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/
Author_A/LoadSiteTree (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/LoadSiteTree (batch
1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/
Author_A/AppendSelectDetails (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetails
(batch 1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/
Author_A/AppendSelectDetailsSE (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/AppendSelectDetailsSE
(batch 1298052933085)
Importing DSKEY @ELEMENTCATALOG-OpenMarket/Xcelerate/AssetType/
Author_A/IndexAdd (batch 1298052933085)
Saved OpenMarket/Xcelerate/AssetType/Author_A/IndexAdd (batch
1298052933085)
...
```

3. Artie access the Advanced interface for the FirstSiteII sample site and confirms that the asset types and start menu items were imported successfully.



4. Now, Artie imports the assets.

Command:

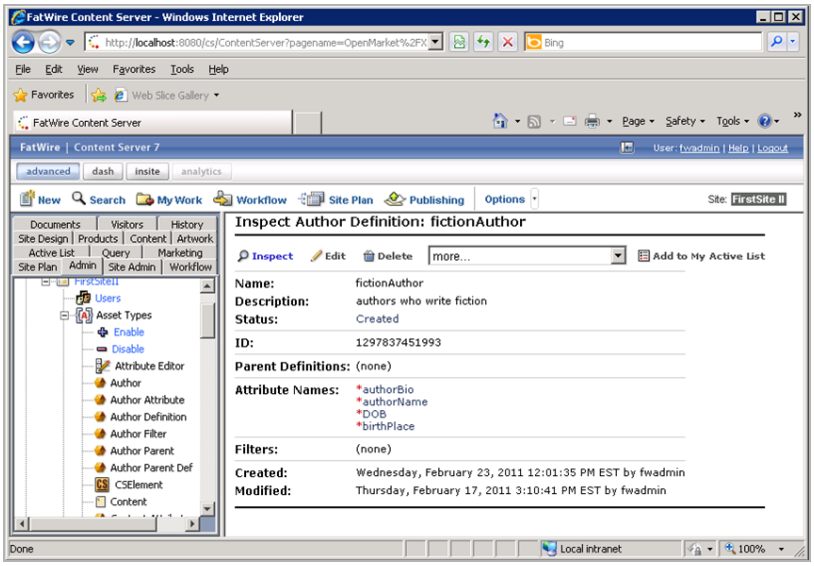
```
D:\FatWire\JSKdemo\ContentServer>java com.fatwire.csdt
  .client.main.CSDT http://localhost:8080/cs/ContentServer
  username=fwadmin password=xceladmin resources=@ALL_ASSETS
  cmd=import datastore=authorModule toSites=FirstSiteII
```

Output:

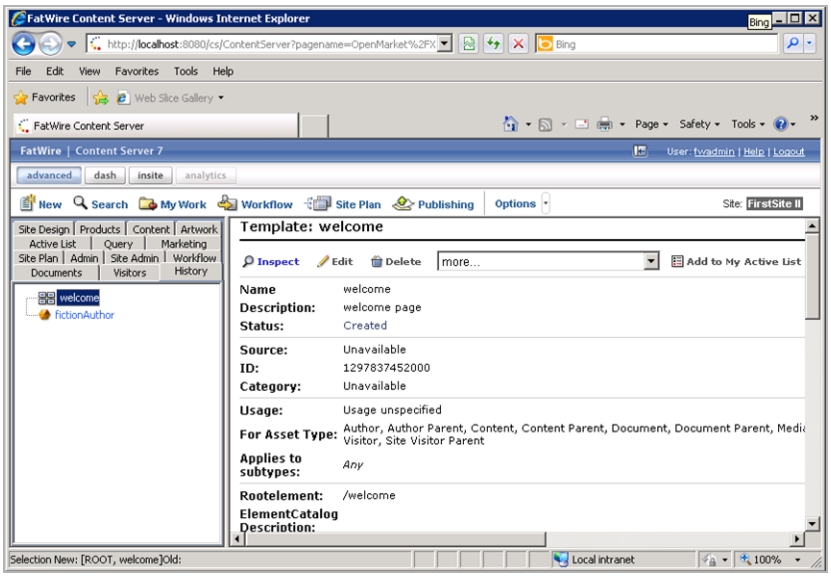
```
                *** Importing batch 1298480206533
Importing DSKEY Author_A-cbf4d8aa-d23a-4f0d-b55d-a87a0e9bbf33 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451977 (batch 1298480206533)
Importing DSKEY Author_A-42afd458-e90c-4e18-a4b6-47d322b46414 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451981 (batch 1298480206533)
Importing DSKEY Author_A-9fe04c6e-36e7-4ee3-8c76-8c02edf74136 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451985 (batch 1298480206533)
Importing DSKEY Author_CD-71d6067b-35f6-47f4-ae97-3876303abb37 (batch
1298480206533)
Importing DSKEY Author_A-ada2d6be-ef14-4766-b446-911bfa838835 (batch
1298480206533)
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_A:1297837451989 (batch 1298480206533)
Dependency @ASSET_TYPE-Author_C already exists, skipping.
Dependency @ASSET_TYPE-Author_P already exists, skipping.
Dependency @ASSET_TYPE-Author_CD already exists, skipping.
Dependency @ASSET_TYPE-Author_PD already exists, skipping.
```

```
Dependency @ASSET_TYPE-Author_F already exists, skipping.
Dependency @ASSET_TYPE-Author_A already exists, skipping.
Saved Author_CD:1297837451993 (batch 1298480206533)
Importing DSKEY Template-89b05c0f-227b-4dcb-961e-2ab6e6af2dae (batch
1298480206533)
Saved Template:1297837452000 (batch 1298480206533)
*** Completed importing batch 1298480206533
```

5. Artie verifies that the flex definition is imported into the FirstSiteII sample site successfully:



6. Using the command-line tool, he also imports the Template asset. He then accesses the Advanced interface again to verify the Template asset is imported correctly.



The entire module is imported successfully into the FirstSiteII sample site. This module can be reused and imported into any desired Content Server instance.