ORACLE®

ATG WEB COMMERCE

Version 10.0.2

Content Administration Programming Guide

Oracle ATG
One Main Street
Cambridge, MA 02142
USA

ATG Content Administration Programming Guide

Document Version

Doc10.0.2 PUBADMINv1 04/15/2011

Copyright

Contents

1 Introduction

This guide introduces ATG Content Administration concepts and tasks for developers and system administrators. It describes how to configure ATG Content Administration to deploy assets to target sites, and it includes information on setting up a versioned database, using workflows, and managing deployments.

Documentation Conventions

In this guide, the notation `<ATG10dir>` represents the ATG platform root directory—for example, `C:\ATG\ATG10.0.1`.

Product Support Information

For detailed information on the supported environments and configurations of the entire ATG product suite, see http://www.atg.com/en/products/requirements/. The following sections outline general product support issues.

Versioning Support

ATG Content Administration supports the versioning of content in SQL repositories, including:

- ATG Portal content
- ATG Consumer Commerce content
- ATG Business Commerce content
- File assets, such as targeters, content groups, Microsoft Word documents, image files, and JSPs.

Requirements

ATG servers in the production environment must run ATG 10.0.1 and the Publishing agent, which is installed with the ATG platform.

Browser Support

Some administration tasks are performed through the ATG Business Control Center. You can use a variety of Web browsers to access this tool, including Internet Explorer, Mozilla Firefox, and Safari. Cookies and scripting should be enabled in the browser.

Earlier versions of Internet Explorer (before version 7) might cause improperly formatted form posts. You can avoid these by reconfiguring your Windows registry. For more information on this problem and other issues related to Internet Explorer, navigate to the following URL:

`http://hostname:port/dyn/dyn/iefix/`

Some assets types can be configured to use an HTML editor that is downloaded to the ATG Business Control Center client machine as a Java applet. For more information, see the *ATG Content Administration Guide for Business Users*.

General Constraints

The following constraints apply to this release:

- Nested content groups and profile groups are not supported. Because the versioning system cannot detect dependencies between a container group and groups nested within, it cannot enforce that they are deployed together.
- Versioning of XML and HTML file content repositories is not supported.
- When deployed to production, all repository assets are non-versioned, and all file assets are deployed as files to the target server's file system.

Related Documentation

Other topics that are related to ATG Content Administration can be found in the following manuals:

Topic	Located in:
Assembling a Web application that includes ATG Content Administration	*ATG Programming Guide*
Creating and managing assets through the ATG Business Control Center	*ATG Content Administration Guide for Business Users*
Managing and customizing project workflows	*ATG Personalization Programming Guide*

2 Getting Started

This chapter provides information on the following topics:

Running ATG Content Administration

To run ATG Content Administration, you assemble a Web application that includes ATG platform and ATG Content Administration modules and deploy it to the appropriate server. The application assembly procedures you follow vary according to the application server you are using and the particular ATG Content Administration modules you require. They also vary according to whether you are using the instance of the product that interacts with the versioned database (usually called the asset management server) or an instance that represents a target site (a production or staging server, for example).

For example, you might include the following modules in your assembly command:

```
PubPortlet Publishing.WebAppRefVer
```

The `PubPortlet` module contains the ATG Portal pieces required for ATG Content Administration functionality in the ATG Business Control Center. It also includes the BIZUI module, which provides the ATG Business Control Center framework.

Note: You cannot run the `Portal.paf` and ATG Content Administration modules in the same ATG instance.

The `Publishing.WebAppRefVer` module holds the versioned module of the `WebAppRef` demo application included with ATG Content Administration.

For information on application assembly and ATG modules, see the *ATG Programming Guide*.

For information on running ATG Content Administration on a target server, refer to Setting Up Deployment.

Caution: Do not use any ATG Content Administration modules or the `DAF.Deployment` module on a standalone GSA lock manager server. Doing so causes the lock server to become a slave server of the

deployment. It attempts to deploy files and the deployment deadlocks as a result. Symptoms include the deployment cycling in a time out loop.

Overview of the ATG Business Control Center

The ATG Business Control Center is a browser-based tool that is the primary interface to most ATG Content Administration-related tasks. Business users perform all ATG Content Administration activity with this tool, except for some asset editing in the ATG Control Center (ACC). Administrators use ATG Business Control Center to configure and manage deployment of assets to one or more target sites.

This guide shows how to use the ATG Business Control Center to perform deployment-related tasks. Detailed information about using it to perform asset creation and editing is contained in the *ATG Content Administration Guide for Business Users*.

To access the ATG Business Control Center, point your Web browser to the server where ATG Content Administration is running. The URL takes the following form:

`http://`*`hostname`*`:`*`port-number`*`/atg/bcc`

The default port numbers are:

- IBM WebSphere: `9080`
- Oracle WebLogic: `7001`
- JBoss Application Server: `8080`

The following case-sensitive username and password correspond to the `EPub-Super-Admin` role defined in the ATG Business Control Center. The account gives you full access to all ATG Content Administration features within the ATG Business Control Center:

- Username: `publishing`
- Password: `publishing`

Before You Begin

Before running ATG Content Administration, you must perform the following tasks:

Task	Performed by:	For more information:
Set up asset management server.	Developer	Setting Up an Asset Management Server
Create and configure ATG Content Administration database.	System administrator	Setting Up an Asset Management Server1

Task	Performed by:	For more information:
Set up user accounts and repository security for ATG Content Administration.	System administrator Developer	Managing User Access and Security
Customize the workflows if necessary for business processes.	Developer	Project Workflows
Configure ATG Content Administration for deployment.	System administrator	Setting Up Deployment

3 Understanding the Content Development Environment

The content development environment is where ATG Content Administration runs, and business users—authors, editors, and other content creators—create and manage content that is deployed to production Web sites.

ATG Content Administration itself is a set of tools that lets you automate the creation, deployment, and maintenance of assets required to run an ATG-based Web site. ATG Content Administration can manage assets such as JSP pages and content files, and personalization assets such as scenarios and targeters.

ATG Content Administration extends the ATG data architecture so you can track multiple versions of data. It uses a workflow mechanism that defines routing and approval processes, and it automates ATG best practices for deployment.

This chapter contains the following sections:

- Terminology defines essential ATG content development terms.
- ATG Content Administration Architecture illustrates the basic configuration of the content development and production environments for a Web site that serves ATG Content Administration-managed content.
- Repositories provides a repository-level view of the content development environment.
- Versioning Assets describes the versioning system that is used by ATG Content Administration.

Terminology

This section defines essential ATG content development terms.

Assets

Assets are persistent, publishable objects that are used by one or more ATG applications. ATG Content Administration supports two kinds of assets:

- A *repository asset* is created and edited in the ACC or the ATG Business Control Center and is deployed as a repository item.

- A *file asset* is created in the ATG Business Control Center or an external application such as Word or Excel and is deployed as a file to the target server.

If a file asset is created in an external application, you can upload the file into ATG Content Administration. Users can edit the item either through the ATG Business Control Center or by downloading the file, changing it and uploading the file again.

In this guide, *asset* refers generally to repository and file assets. The terms *file asset* and *repository asset* are used for more specific references.

For technical information on file and repository assets, see the chapter Understanding the Content Development Environment in this guide. For information on working with assets in the ATG Business Control Center, see the chapter *Creating and Managing Assets* in the *ATG Content Administration Guide for Business Users*.

Project

A project is a persistent entity that encompasses additions, changes and deletions to a set of assets. Each project moves independently through a workflow that typically includes these tasks:

- Author or revise asset content
- Review content changes
- Approve the project for deployment
- Verify deployment

Project objects are defined by the project `item-descriptor` in the `publishing.xml` repository definition file, located in:

```
<ATG10dir>/Publishing/base/config/atg/epub
```

Project Workflow

Project Workflows define and control the user tasks that make up a project and modify its state. Project workflows also define actions that are triggered by state changes, such as workspace check-in, email notification, and tagging of versions for deployment. For more information, see Project Workflows.

Process

A process is the parent object of a project workflow.

Process objects are defined by the process `item-descriptor` in the `publishing.xml` repository definition file.

Versioning

ATG Content Administration maintains versions of all assets, to ensure that the latest changes are deployed to target sites, and to coordinate multi-user access to the same asset. For more information, see Versioning Assets.

Task

A task is a step in a workflow, such as Author, Review, or Deploy. For more information, see the *Managing Tasks* chapter in the *ATG Content Administration Guide for Business Users*.

Deployment

ATG Content Administration defines a deployment model that incorporates server clusters for development, staging, and production. You can modify the workflow that is provided with ATG Content Administration so its deployment process suits your specific requirements. For example, you can deploy to several servers at the same time; or you can define staged deployments in a staging/production workflow, which lets you review assets on a staging server before deploying them to production and checking them in.

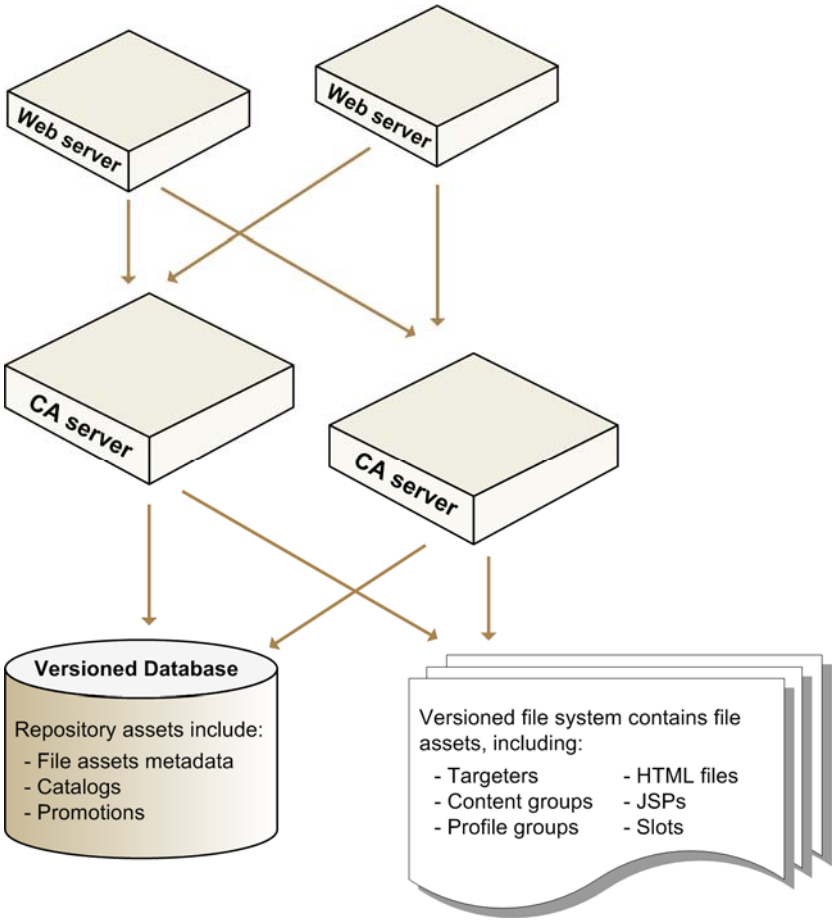
ATG Content Administration Architecture

A Web site that serves assets managed by ATG Content Administration requires:

- A content development environment—that is, an authoring and editing environment that runs ATG Content Administration and is configured to manage multiple versions of assets.
- A production environment that is configured to receive deployments of assets from the content development environment.

Content Development Environment

The following diagram outlines the configuration of a content development environment:

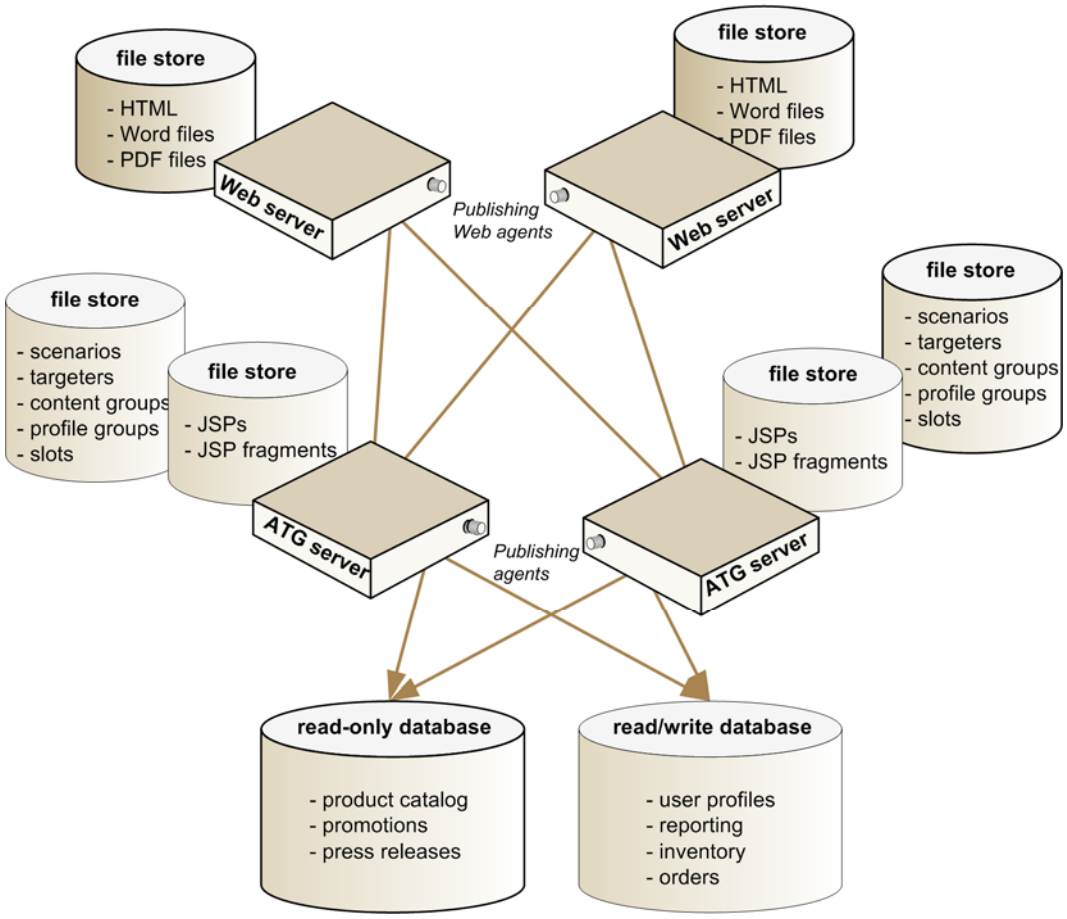


This diagram illustrates several important concepts about the content development environment:

- The content development environment uses a *versioned* database schema. The asset management server's database schema is configured to store successive versions of assets, including version metadata.
- Because a production environment does not use a versioned database schema, it is isolated from the content development environment.
- Versioned repository assets and file asset metadata are stored in the *versioned database*. The actual versioned file assets are stored in a file system.
- To facilitate scalability, and improve performance and reliability, you can cluster multiple asset management servers.

Production Environment

The following diagram illustrates the basic configuration of a production environment where ATG Content Administration-managed assets are deployed:



The preceding diagram illustrates several important concepts about the production environment:

- The production environment does not use a versioned schema. When you deploy a specific version of an asset to your production environment, its version metadata is excluded.
- The DeploymentManager moves assets from the content development environment to the production environment. However, each target site server runs an agent that handles deployment-related tasks on that server.

  Two types of agents run on target sites:

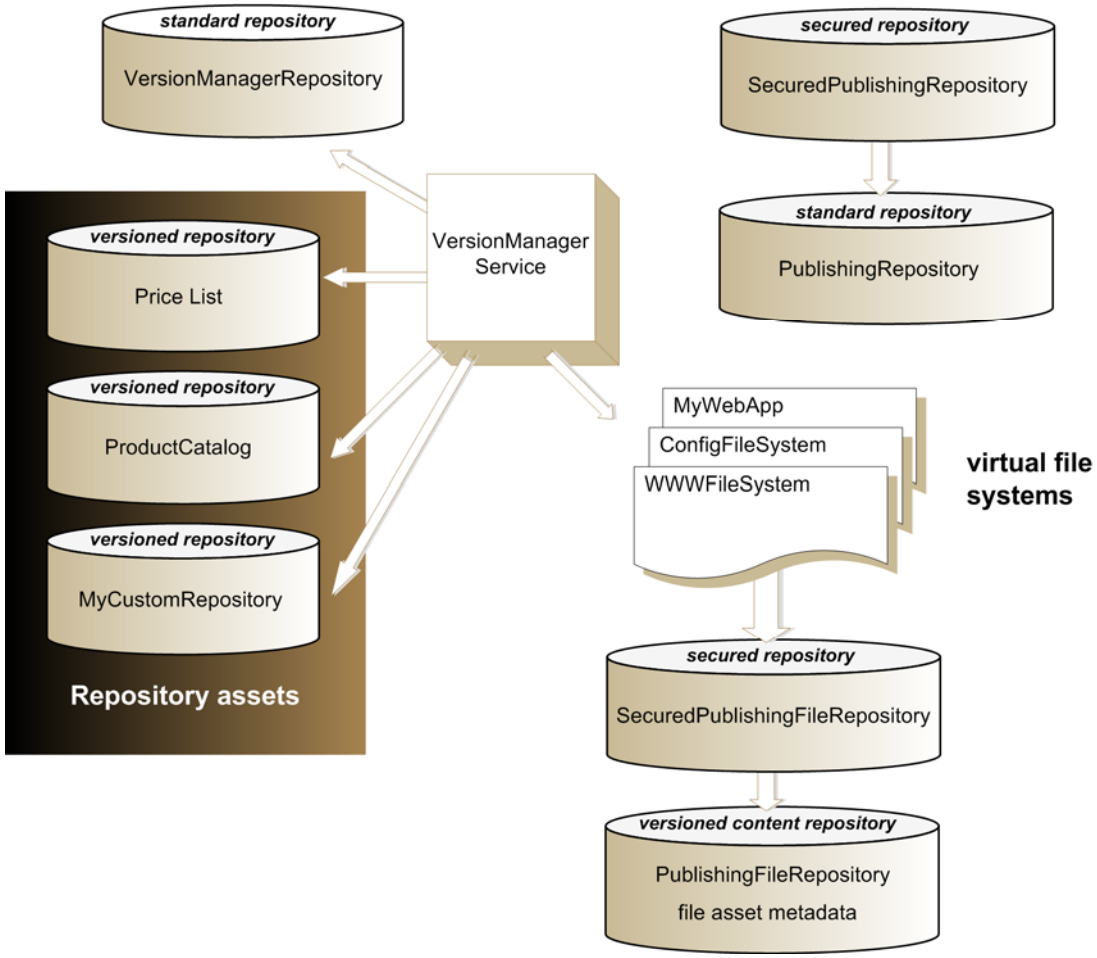
  - **Publishing agents** run on the ATG servers. These agents are configured to manage deployment-related tasks for repository and file assets on the target site.
  - **Publishing Web agents** run on the Web servers. These agents are configured to manage deployment-related tasks for file assets on the target Web server.

The preceding diagram shows a production environment with a single ATG cluster, where multiple ATG instances point to the same data stores. However, a target site can include multiple ATG clusters. For

simplification, it also indicates a configuration for online deployment; however, a production environment should always be configured to use switch deployment.

Repositories

The following diagram shows how repositories are used in a content development environment.



A content development environment has three types of repositories:

- Versioned repositories store an application's repository assets
- A single *versioned content repository* (the PublishingFileRepository) stores metadata for application file assets.
- *Standard repositories* store data required by ATG Content Administration.

Asset Types

Project assets are of two types: repository assets or file assets. The following table briefly describes how repository and file assets are maintained in content development and production environments:

Asset type/Examples	Development environment	Production environment
Repository asset ATG Commerce assets such as product catalogs	Stored in versioned repository, instance of VersionRepository.	Stored in a standard repository, instance of GSARepository.
File asset ATG personalization assets: targeters, content and profile groups, user segments, scenarios, slots JSPs and JSP fragments Static text files: HTML and readme files Binary files: Microsoft Word files, PDFs	Metadata stored in versioned content repository, instance of VersionRepository; exposed via virtual file system ContentRepositoryVFSService for use in various contexts, such as deployment.	Stored in a virtual file system.

Versioned Repositories

Versioned repositories store an application's repository assets—that is, assets that are deployed as repository items to a GSARepository in the production environment.

As part of the process of adding an asset management server to your configuration, you must create and configure the versioned repositories that store the application's repository assets. Next, you can import the initial set of repository assets. For more information, see Configure Repository Asset Support.

Versioned Content Repository

File assets are stored and versioned in a file system on the asset management server (see File Asset Storage). The versioned content repository `/atg/epub/file/PublishingFileRepository` stores all file asset metadata. It also acts as the temporary storage layer for an application's file assets, providing access to persistent storage in the underlying versioned file system. A virtual file system ContentRepositoryVFSService reads file assets from the PublishingFileRepository and exposes them for use in various contexts, such as deployment.

A file asset's deployment destination is determined by the virtual file system (VFS) that exposes its item type in the content development environment. For this reason, every VFS that exposes a subset of item types in the PublishingFileRepository requires at least one corresponding VFS at the same Nucleus

location on the target site. File assets are deployed from the asset management server-side VFS to each target site VFS at the same Nucleus location.

The following table describes the general classes of assets that the default PublishingFileRepository supports.

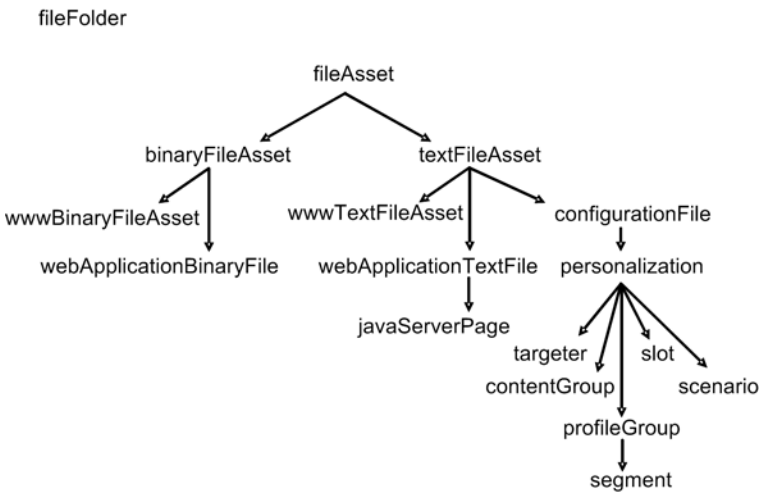
Class of assets	Description
Personalization assets	Include targeters, profile groups, content groups, user segments, slots, and scenarios. In the content development environment, personalization assets are exposed as files via the `/atg/epub/file/ConfigFileSystem`, which is a configured VFS that is included with ATG Content Administration. Personalization assets are deployed as files to each VFS in the target environment at the same Nucleus location. The Publishing agent, which runs on target site ATG servers and handles deployments, includes a configured ConfigFileSystem VFS. The Publishing agent is installed with the ATG platform.
Web assets	Include text file assets such as HTML and readme files, and binary file assets such as Word documents and PDFs. In the content development environment, Web assets are exposed as files via the `/atg/epub/file/WWWFileSystem`, which is a configured VFS that is included with ATG Content Administration. These assets are deployed as files to each VFS in the target environment at the same Nucleus location. The Publishing Web agent, which runs on target site Web servers, includes a configured WWWFileSystem VFS. You must manually install the Publishing Web agent on a target site server.

The following sections discuss the following topics:

- Item type hierarchy of the PublishingFileRepository
- Extending the PublishingFileRepository

Item Type Hierarchy of the PublishingFileRepository

The following illustration describes the hierarchy of item types that is defined in the repository definition file `<ATG10dir>/Publishing/base/config/atg/epub/file/publishingFiles.xml`:



In the content development environment, a file type is determined only by its item descriptor type; its file extension is irrelevant. Targeters that are created manually and imported into the ATG Content Administration system provide a good reason why this is so: depending on the implementation, manually created targeters might consist of two files with different extensions:

- `.properties`: a `RuleSetService` configuration file
- `.rules`: a text file that stores the actual rule set

Despite their different extensions, both files are of type `targeter` in the PublishingFileRepository. This enables them to be exposed via the same VFS on the asset management server and deployed to the same corresponding VFS on the target server.

When creating assets in the ATG Business Control Center, users can optionally enter file extensions in order to associate third-party applications with assets on download. For example, while ATG Content Administration identifies a JPEG image by its asset type (`wwwBinaryFileAsset`), entering the `.jpg` extension on asset creation enables the operating system to identify this asset as an image file and open the appropriate editor when the user downloads the file for editing.

Extending the PublishingFileRepository

If you do not need to deploy assets to a VFS other than ConfigFileSystem or WWWFileSystem, the default `/atg/epub/file/PublishingFileRepository` supports file asset deployment to your Web site. However, you might need to deploy file assets to other destinations—for example, deploy JSPs to a Web application, or text and binary files to an FTP server.

In general, configuring the asset management server to support an additional deployment destination requires you to extend the PublishingFileRepository to support new item types and configure the appropriate VFSs in the content development and target environments accordingly. Depending on your requirements, see one of the following sections:

- Configure JSP File Asset Support
- Configure Support for Other File Assets

Caution: Never create and configure a second version content repository in order to manage files in the content development environment. The PublishingFileRepository should be the sole repository for all file assets. Simply extend it to support the additional item types and, therefore, asset destinations in your deployment targets.

While you can extend default item types to support additional properties, those properties are not deployed; only a file's contents are deployed. To support an additional property, simply extend the repository definition file, modify the database schema accordingly, and customize the ATG Business Control Center as needed.

Standard Repositories

The following two standard repositories store the data that ATG Content Administration requires:

- VersionManagerRepository
- PublishingRepository

VersionManagerRepository

The `/atg/epub/version/VersionManagerRepository` is a GSARepository that stores the development lines (branches, workspaces, and snapshots) and asset version metadata used by the `/atg/epub/VersionManagerService`, the default VersionManager provided with ATG Content Administration.

For more information on versioning concepts and terms, see Versioning Assets.

PublishingRepository

The `/atg/epub/PublishingRepository` is a GSARepository that stores the items required by the ATG Business Control Center: processes, projects, workflows, and so on.

To secure access to its items, the PublishingRepository has a secured repository configured on top. The secured repository is located in Nucleus at `/atg/epub/SecuredPublishingRepository`. For information on configuring security of the PublishingRepository, see the chapter Managing User Access and Security.

Note the following about the PublishingRepository:

- If you export the items in the `/atg/epub/PublishingRepository` for any reason before reimporting the items—for example, when rebuilding your content development environment—you must manually remove the following items from the data file:
  - All workflow-related items. That is, all items of types defined in:

    `<ATG10dir>/Publishing/base/config/atg/epub/workflow.xml`
  - All process- and project-related items. That is, all items of type `process` and `project` and all child items defined in:

    `<ATG10dir>/Publishing/base/config/atg/epub/publishing.xml`
- These items should not be reimported into the PublishingRepository.

- Each `process` item in the `/atg/epub/PublishingRepository` stores a set of processTaskInfo items in its `taskInfos` property. These items store information about the project's workflow tasks—for example, the segment name for each task. After a project completes its workflow, its processTaskInfo items are no longer needed and should be purged. This is an important clean-up task, as performance can be affected if the number of items exceeds its `item-cache-size`.
- The `/atg/epub/workflow/process/TaskInfoPurger` component is a service that periodically purges the processTaskInfo items of completed projects. By default it is scheduled to run every day at 1:00 a.m. You can change this schedule so it better reflects how often ATG Business Control Center users create projects—for example, every 6 hours. To do so, edit the `TaskInfoPurger.schedule` property.

File Asset Storage

File asset metadata is stored in the PublishingFileRepository. Actual file assets—`WWWFileSystem` files, scenarios, targeters, segments, folders, and JSPs—are stored and versioned in a file system. The file system is located by default at:

```
<ATG10dir>/home/Publishing/versionFileStore/PublishingFiles/fa100/
```

The `fa100` directory contains one or more subdirectories named after integers. For example:

```
<ATG10dir>/home/Publishing/versionFileStore/PublishingFiles/fa100/0
```

By default, the 0 directory contains a maximum of 1000 file assets and 30 versions of those assets. If the number of file assets exceeds 1000, a new directory called 1 is created, and so on.

Individual files are stored with a file asset ID that has the root `fa100`. The version number is appended. For example:

```
_cricket_0020fan_xproperties._fa100778#2
```

Note the following:

- File metadata is stored in the `epub_file_asset` table. If you need to examine any deployment issues with file assets, this table can help you determine which asset might be causing a problem. (File assets are referred to by their ID in deployment errors.)
- Folder assets are not visible in the file system.

Caution: Be sure to back up file asset directories up at the same time as the ATG Content Administration database. Conflicts between the file store and the database can cause serious errors.

Deleting File Assets from the File System

You should only delete file assets in the context of projects that you create in the ATG Business Control Center. For more information, refer to the *ATG Content Administration Guide for Business Users*.

Warning: Never move or delete file assets directly by manipulating the file system. Doing so can fatally compromise data integrity.

Changing File Asset Storage Location

The location of the file system used for file assets is determined by the `pathPrefix` attribute of the `content` property for the `fileAsset` item descriptor in the `publishingFileRepository`. To change the location, create a `publishingFiles.xml` file with a new `pathPrefix` attribute that overrides the default, and put the file in your application's configuration path.

For example:

```
<gsa-template>
   <item-descriptor name="fileAsset">
     <property name="content" xml-combine="append">
       <attribute name="pathPrefix"
             value="{atg.dynamo.server.home}/Publishing/myFiles"
             xml-combine="replace"/>
     </property>
   </item-descriptor>
</gsa-template>
```

The new location must be on the asset management server.

Note: The `pathPrefix` attribute value should be set with an environment variable, as shown in the previous example. Do not use an explicit file system path such as `c:\atg\atg2007.1\home`.

Processes and Projects

ATG Content Administration defines process and project objects, which are stored in the PublishingRepository (defined by `publishing.xml`):

- The process object manages the lifecycle of the subordinate project workflow object.
- The tasks that users complete through the ATG Business Control Center—for example, Author, Content Review, and Deploy—are managed by the project workflow.

Process Object Properties

The process object has the following properties:

Property	Description
`workflow`	The workflow associated with this process

Property	Description
id	The process ID
version	The GSA version
acl	The access control list for the process, which defines the ACC user roles that have security rights to this process
displayName	The display name for the associated project in the ATG Business Control Center
description	The description that appears for the associated project in the ATG Business Control Center
creator	The ID of the user profile responsible for creating the process instance in the ATG Business Control Center
project	The ID of the current or last checked in project for this process
processData	Associated processData from the /atg/epub/process/ProcessDataRepository
workflowID	The ID of the workflow associated with this process
autodeploy	The autodeploy flag used by deployment. Set to false by default.
status	Valid values are EDIT, DEPLOYED, RUNNING, COMPLETED, EDIT_AND_RUNNING
statusDetail	Additional status text
creationDate	Timestamp recording when the process was created
completionDate	Timestamp recording when the process was completed
history	Any history items stored for this process in the epub_proc_history table
previousProjectIds	A list of previous project IDs for this process from oldest to newest
workflowInstances	The individual workflow instances for this process
taskInfos	The associated processtaskInfos for this process

Project Object Properties

The project object has the following properties:

Property	Description
assets	A set of non-hidden assets in the project

Property	Description
checkedInAssets	A set of all the checked-in assets in the project
workflow	The workflow associated with this project
locked	The locked flag for deployment
id	The project ID
version	The GSA version
acl	The access control list for the process, which defines the ACC user roles that have security rights to this project
displayName	The display name for this project in the ATG Business Control Center
description	The description that appears for this project in the ATG Business Control Center
creator	The ID of the user profile responsible for creating the instance of the project in the ATG Business Control Center
workspace	The workspace name associated with the project
workflowId	The ID of the workflow associated with this project
checkedIn	A flag indicating whether this project progressed past the workflow element that checks in its assets
editable	A flag marking the project as editable or not. The flag is set to false when the Author task is complete.
status	Valid values are ACTIVE and COMPLETED
statusDetail	Additional status text
checkinDate	A timestamp indicating when check-in occurred for this project (when the appropriate workflow element is completed)
creationDate	Timestamp recording when the project was created
completionDate	Timestamp recording when the project was completed
history	Any history items stored for this project in the epub_pr_history table and displayed in the History tab

The project object also contains a number of properties that let it handle deployment. For more information on these, see the deployment chapters in this guide.

Versioning Assets

Before you begin working with ATG Content Administration, you should have a basic understanding of its versioning system. The following sections introduce key terms and concepts:

- Versioning Terminology
- Versioning Process
- Versioning APIs

Versioning Terminology

This section defines the versioning terms that are important to understand.

Asset

A resource under the control of the version management system used by ATG Content Administration—for example, `foo.html`. An asset maintains a list of its own asset versions.

Asset Version

A specific version of an asset—for example, version 2 of `foo.html`. Each asset version has a unique version number. The initial version of an asset is always version 1. When a new asset version is created, it is immediately assigned the next available version number.

Head Version

The most recent version of an asset for a particular branch.

Development Line

In other version management systems, a set of assets is often called a line of development or code line. In ATG Content Administration, a development line is one of the following:

- **Branch:** A named, modifiable line of development that includes a specific set of assets. ATG Content Administration provides an empty *main branch*. As part of the process of setting up your asset management server, you import your initial set of assets into the main branch.

  Branch assets are modifiable only if they are checked out into a workspace.

- **Workspace:** A working, editable set of asset versions. Workspaces are always created from branches. Asset versions are checked out into workspaces, where they can be edited or deleted.

  Every ATG Content Administration project has its own workspace that includes the working set of asset versions for the project. When a user creates a project in the ATG Business Control Center, the VersionManager is called to create a workspace for the project. The user can then add one or more asset versions into the project workspace. When the project is complete, the updated versions of the project assets are checked into the repository, and the workspace itself is marked as checked in.

- **Snapshot:** A set of assets identified by a timestamp in the database and the project `checkin_date`.

VersionManager

The VersionManager service manages all development lines that are created and used in the content development environment. It is the central factory for creating, storing, and querying all versioning objects: development lines, assets, and asset versions. ATG Content Administration provides a default VersionManager, which is located in Nucleus at `atg/epub/version/VersionManagerService`.

The development lines managed by the VersionManagerService are stored in the `/atg/epub/version/VersionManagerRepository`, which is also provided with ATG Content Administration.

Versioning Process

When you initially import assets into ATG Content Administration, the VersionManager creates version 1 of every imported asset. In the case of file assets, file names are appended with the string `#1`.

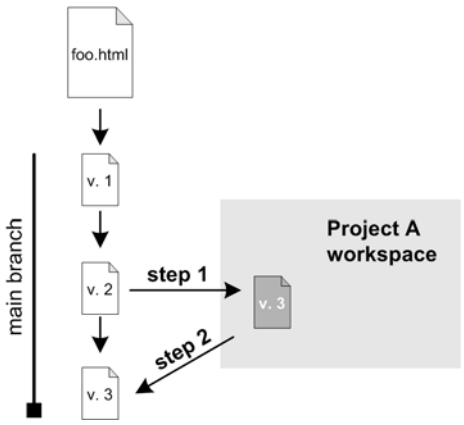
When a user adds an existing asset to a project, the VersionManager checks out the head version from the main branch by copying the asset and assigning the next available version number to this copy. The VersionManager then adds the new version to the project's workspace. Within the project, users can modify this asset's workspace version, or delete it from the versioning system. When the user completes the project, its workspace assets are checked into main, and the workspace itself is marked as checked in.

The versioning system typically deals with one of the following scenarios:

- A single user modifies an asset and checks it in.
- Multiple users modify an asset at the same time.

Single User Modifies an Asset

The following diagram shows how an asset is modified in a project and versioned.



Step 1: The user adds version 2 of `foo.html` to project A. The VersionManager performs these steps:

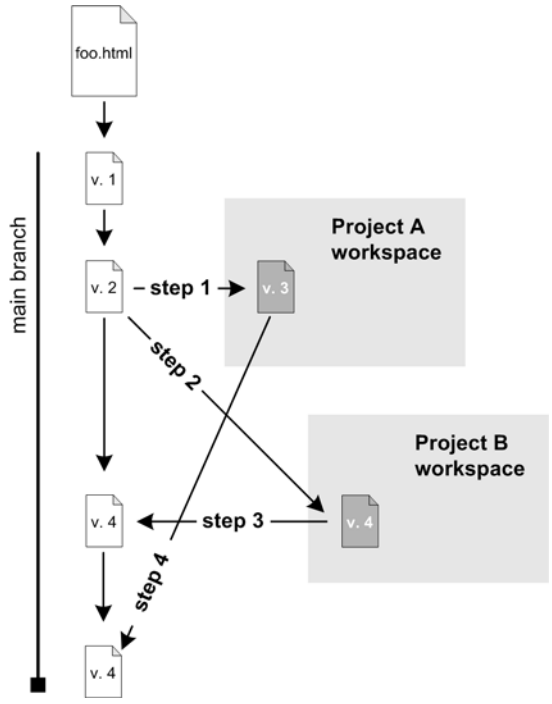
1. Checks out version 2 by creating a copy.
2. Assigns it the next available version number: version 3.
3. Adds version 3 to the project's workspace.

The user can now modify or delete version 3 of `foo.html`.

Step 2: The user completes project A. The VersionManager checks in the asset versions in project A's workspace and marks the workspace as checked in. Version 3 is the new head version of `foo.html` on the main branch.

Multiple Users Modify an Asset

The following diagram shows what happens when two users check out the same asset and the second user's project completes first:



Step 1: User Angela adds version 2 of `foo.html` to project A. The VersionManager performs these steps:

1. Checks out version 2 by creating a copy.
2. Assigns the next available version number: version 3.
3. Adds version 3 to project A's workspace.

Angela modifies version 3 within project A.

Step 2: User Bradley adds version 2 of `foo.html` to project B. The VersionManager performs these steps:

1. Checks copies version 2 by creating a copy.
2. Assigns the next available version number: version 4.
3. Adds version 4 to project A's workspace.

Bradley modifies version 4 within project B.

Step 3: Bradley completes project B. The VersionManager checks in version 4 of `foo.html`. Version 4 is now the head version on the main branch.

Step 4: Angela completes project A. The VersionManager cannot check in out-of-date version 3, and alerts the user about the conflict. Angela decides to merge her version into head version 4. The VersionManager assigns version number 5 to the merged version. When Angela completes the project, version 5 becomes the new head version of `foo.html` on the main branch.

For more about resolving asset conflicts, see Creating and Managing Assets in the *ATG Content Administration Guide for Business Users*.

Versioning APIs

For information on versioning APIs, see the following packages in the *ATG API Reference*:

- `atg.versionmanager`
- `atg.versionmanager.event`
- `atg.versionmanager.exceptions`

4 Setting Up an Asset Management Server

An asset management server is an ATG instance that runs ATG Content Administration and whose SQL repositories and database schema are configured to manage multiple versions of assets. Unlike the development server, ATG Content Administration tables use a versioned schema, so it requires its own set of repositories and database tables.

By default, ATG Content Administration is configured to use the SOLID database server, which is suitable only for demonstration and evaluation purposes. Before starting serious development work, it is important that you reconfigure ATG Content Administration to work with the database instance to be used in your production environment. All non-SOLID database servers such as Oracle and DB2 require you to perform the setup steps described in this chapter.

In general, it is good practice to design your databases so that ATG Content Administration and production data are maintained separately, and transactions that pertain to one have no affect on the other. Transactions that are initiated by ATG Content Administration can sometimes incur considerable overhead; by separating versioned and production data, you can ensure that these transactions have no effect on production data processing.

Note: After you import the application's content into the versioned repositories of ATG Content Administration, you cannot reexport the content and import it into another system.

This chapter shows how to create and import the basic data required by an asset management server, in the following steps:

1. Create ATG Content Administration tables.
2. Initialize the database.
3. Back up the database.

CIM Usage

You can use ATG's Configuration and Installation Manger (CIM) to perform these tasks. For more information, see *ATG Installation and Configuration Guide*.

For descriptions of the ATG Content Administration database tables, see Appendix A: Database Schema.

Prerequisites

Before you set up the asset management server database tables:

- Install the latest versions of ATG Content Administration and other ATG applications that depend on ATG Content Administration, such as ATG Commerce and ATG Merchandising. For more information, see *ATG Installation and Configuration Guide*.
- Determine which database system you will use in your production environment, and install an instance of that database on the asset management server.
- Set the following environment variables:

  `DYNAMO_HOME`: `<ATG10dir>/home`

  `JAVA_HOME`: Java installation directory—on Windows, to the JDK directory

Create ATG Content Administration Tables

This section describes how to create and configure tables for the versioned database. For descriptions of the ATG Content Administration database tables, see Appendix A: Database Schema.

Note: If you run Microsoft SQL Server, you must be the database owner and its compatibility level must be set to 80, before you run the scripts referenced in this section.

To install and configure database tables, run this script:

`<ATG10dir>/BIZUI/sql/install/`*`db-vendor`*`/bizui_all_ddl.sql`

where *`db-vendor`* is a vendor-specific directory such as `mssql` or `oracle`. The `bizui_all_ddl.sql` script installs all tables that are required to run ATG Content Administration and ATG Business Control Center, including tables for the ATG platform and ATG Portal. For detailed information about the tables specifically required by these products, see "Creating Database Tables Using SQL Scripts" in the *ATG Installation and Configuration Guide*

For information about table creation scripts that are specific to other ATG applications, see the relevant product documentation.

Table Creation Subscripts

The `bizui_all_ddl.sql` script executes a number of subscripts:

- ATG Content Administration table creation scripts
- ATG Business Control Center table creation scripts

ATG Content Administration Table Creation Scripts

The `bizui_all_ddl.sql` script executes the following subscripts in order to create and configure database tables for ATG Content Administration:

Script file name	Description
internal_user_profile_ddl.sql	Creates tables for the extensions to the internal user profile specifically for ATG Content Administration.
publishing_ddl.sql	Creates tables for the /atg/epub/PublishingRepository, which stores process, and deployment topology information.
versioned_file_repository_ddl.sql	Creates the tables for the /atg/epub/file/PublishingFileRepository, the versioned repository that is provided for storing file asset metadata.
versioned_process_data_ddl.sql	Creates the tables for the /atg/epub/process/ProcessDataRepository, the versioned repository that is provided for storing information about ATG Content Administration processes.
versionmanager_ddl.sql	Creates tables for storing versioning information.
workflow_ddl.sql	Creates tables for storing workflow information in the /atg/epub/PublishingRepository.

These scripts are located in:

<ATG10dir>/Publishing/base/sql/db_components/*db-vendor*

ATG Business Control Center Table Creation Script

The bizui_all_ddl.sql script executes viewmapping_ddl.sql to create and configure the database tables required by the ATG Business Control Center. This subscript creates tables that are used by the View Mapping framework.

This script is located in:

<ATG10dir>/BIZUI/sql/db_components/*db-vendor*

Destroying ATG Content Administration Tables

You can destroy all ATG Content Administration, ATG Business Control Center, ATG Portal, and ATG platform tables with this script:

<ATG10dir>/BIZUI/sql/install/*db-vendor*/drop_bizui_all_ddl.sql

For information on the individual scripts used to destroy tables for the ATG platform and ATG Portal, see "Destroying Database Tables" in the *ATG Installation and Configuration Guide*. For information about destroy scripts that are specific to other ATG applications, see the relevant product documentation.

The drop_bizui_all_ddl.sql script executes a number of subscripts:

- ATG Content Administration database drop scripts
- ATG Business Control Center database drop script

ATG Content Administration Database Drop Scripts

The drop_bizui_all_ddl.sql script executes the following subscripts in order to destroy the database tables for ATG Content Administration:

File Name	Description
drop_internal_user_profile_ddl.sql	Destroys the tables for the extensions to the user profile specifically for ATG Content Administration.
drop_publishing_ddl.sql	Destroys the tables for the ATG Content Administration repository, which holds processes and process types.
drop_versioned_file_repository_ddl.sql	Destroys the tables for the /atg/epub/file/PublishingFileRepository, the versioned repository that is provided out-of-the-box for the following types of file assets: ATG Scenario Personalization assets (scenarios, targeters, slots, and so on), text file assets, and binary file assets.
drop_versioned_process_data_ddl.sql	Destroys the tables for the /atg/epub/process/ProcessDataRepository, the versioned repository that is provided out-of-the-box for storing information about ATG Content Administration processes.
drop_versionmanager_ddl.sql	Destroys the tables for storing versioning information.
drop_workflow_ddl.sql	Destroys the tables for the workflow portion of the ATG Content Administration repository.

These scripts are located in:

<ATG10dir>/Publishing/base/sql/uninstall/*db-vendor*

ATG Business Control Center Database Drop Script

The drop_bizui_all_ddl.sql script executes drop_viewmapping_ddl.sql to destroy the database tables used by the ATG Business Control Center. This script is located in:

<ATG10dir>/BIZUI/sql/uninstall/*db-vendor*

Initialize the Database

In order to populate a non-SOLID database with ATG Content Administration and ATG Business Control Center data, you must run the ATG import scripts described in this section.

Prerequisites

Before you run the import scripts, two requirements apply:

- In `<ATG10dir>/home/localconfig/atg/dynamo/service/jdbc`, configure:
  - `FakeXADataSource.properties` file that points to the versioned database.
  - `FakeXADataSource_production.properties` file that points to the production (unversioned) database
- Remove this directory:

  `<ATG10dir>/home/Publishing/versionFileStore`

Import Steps

Run the import scripts in the following order:

1. `<ATG10dir>/Publishing/base/install/importPublishing.{sh|bat}`
   Imports the roles, users, item descriptors, and items (views, actions, and so on) for ATG Content Administration.

   Note: Several `ContentRootPathProvider` and `FileExtensionTypeMapper` errors might appear while the script is running; you can safely ignore these errors, as they have no effect on the import process.
2. `<ATG10dir>/BIZUI/install/importBizui.{sh|bat}`
   Imports default data for the ATG Business Control Center framework.
3. `<ATG10dir>/AssetUI/install/importAssetUI.{sh|bat}`
   Imports view mapping data used by the `AssetUI` module.
4. `<ATG10dir>/DPS-UI/install/importDPSUI.{sh|bat}`
   Imports Personalization data for the ATG Business Control Center.
5. Optionally, `importPublishing-webapprefver.{sh|bat}`
   Located in:

   `<ATG10dir>/Publishing/WebAppRefVer/install/`

   This script Imports the items for the `WebAppRef` reference implementation, which illustrates a proper configuration for a Web site that manages JSP file assets. It also imports the items for base ATG Content Administration.

Back Up the Database

To ensure that your versioned data is safe and can be reloaded if necessary, it is highly recommended that you regularly back up the ATG Content Administration database. Specifically, you should back up the following tables:

- All tables used by the `/atg/epub/PublishingRepository`, the `/atg/epub/file/PublishingFileRepository`, and the `/atg/epub/process/ProcessDataRepository`. The names of these tables begin with the prefix `epub_`.
- All tables used by the `/atg/epub/version/VersionManagerRepository`. The names of these tables begin with the prefix `avm_`.
- All tables used by your custom versioned repositories.

For descriptions of the ATG Content Administration tables, see Appendix A: Database Schema.

5 Creating a Versioned Module

This chapter shows how to import site content into versioned repositories and schemas, in the following steps:

1. Copy the application module to the asset management server.
2. Create the versioned module that is to store the application's versioning configuration.
3. Configure repository asset support, if required.
4. Register the versioned repositories with ATG Content Administration.
5. Configure JSP file asset support, if required.
6. Configure support for other file assets, if required.
7. Configure the VersionManagerService to manage versioned repositories and virtual file systems.
8. Import initial repository assets into versioned repositories.
9. Import initial file assets.

Note: This procedure assumes an existing Web site that is packaged as an ATG application module, with well-defined repository definitions and database schemas.

Setup Scripts

ATG Content Administration provides two scripts that can help you import content into the versioned repositories. These are described at the end of this chapter:

- exportRepository exports the contents of one or more standard repositories.
- importRepository imports the contents of a data file generated with `exportRepository` into a standard or versioned repository.

Related Tasks

Before you set up the versioned database, you must first create and install the versioned database schema, as described in the previous chapter.

If your environment includes ATG applications that run on top of ATG Content Administration, such as ATG Commerce or ATG Outreach, you must install database tables that are specific to those applications and import data into them. See the manual that describes your application for instructions.

Copy the Application Module to the Asset Management Server

Your production Web site must be packaged as an ATG application module. This module includes all the class files, configuration files, JSPs, and HTML files that the site uses. You copy the application module from the production (or development) server to the asset management server.

The nature and complexity of the application module determines whether you copy over the entire module. For example, in the case of commerce applications, you can omit components for the checkout process, because these components are not utilized by ATG Content Administration. However, it might be simpler to copy over an entire module with complex dependencies.

The list of specific class files, configuration files, and JSPs to copy varies for each application module; however, you must copy the following module resources to the asset management server:

- Any custom events, actions, and conditions that you added to the Scenarios module, if you intend to create and manage scenarios with ATG Content Administration.
- Definition and configuration files for all repositories that you want to manage with ATG Content Administration, including any configuration files on which the repositories depend. You must copy the definition files for repositories that contain read-only data, such as Product Catalog and Price Lists repositories. You can omit definition files for repositories that contain read/write data that can be modified in the production environment, such as Profile, Order, and Inventory repositories.
- If application repositories have secured repositories layered above them, you can also copy their definition and configuration files. This lets you define access rights to the assets for ATG Content Administration users. For more information on defining access rights to assets, see Managing User Access and Security.

Create the Versioned Module

The versioned module stores the application's versioning configuration, so it is maintained separately from the application's production configuration. When you create the versioned module, you specify the source module and appropriate ATG Content Administration modules in the `ATG-Required` property of the versioned module's manifest.

When creating the versioned module's manifest, list ATG Content Administration modules before the source module:

```
ATG-Required: Publishing.base source-module-name
```

Note: Although not required, it is a good idea to differentiate the source module from the original application module by appending `ver` to the source module name. For example, given source module `Catalog`, you might name the versioned module `CatalogVer`.

For detailed information on creating an ATG application module and its manifest file, see the *ATG Programming Guide*.

Configure Repository Asset Support

Configuring the asset management server to support your application's repository assets involves these general steps:

1. Create and configure versioned repositories.
2. Create and install the versioned database schema.

Note: These steps are required for versioned repositories that store an application's repository assets. To configure file assets, see Configure JSP File Asset Support and Configure Support for Other File Assets.

Create and Configure Versioned Repositories

In order to create and configure a versioned repository for one of your application's repositories, perform these tasks:

1. Verify the versioned repository definition.
2. Modify the repository configuration.
3. Modify the repository definition if necessary, to indicate which item types should be versioned.
4. Optionally, set the behavior of versioning caches, which control the behavior of each versioned item's HeadOfLine and CurrentVersionItem caches.
5. Modify the secured repository definition if applicable, to give ATG Business Control Center users access to the items.

Verify the Versioned Repository Definition

Before creating a versioned repository, verify the original repository definition file. In general, make sure it conforms to the original (unversioned) database schema.

Specifically, verify the following in the repository definition file:

Verify...	Requirements

readable/writable attributes	Set to true in all <property> tags within a <table>, for all properties whose values you want to maintain from one asset version to the next.
	
	If a property is not readable and writable, its value for a given item is not checked in and maintained in the versioning system, nor is it deployed to a staging or production target. By default, all properties are readable and writable.
	
	Note 1: If necessary, indicate not to deploy a readable and writable property by setting its deployable attribute to false, as in the following example:
	
	`<property name="myProperty">`
	`  <attribute name="deployable" value="false" data-type="boolean"/>`
	`</property>`
	
	You can also set the deployable attribute for item-descriptors:
	
	`<item-descriptor name="myItemDescriptor">`
	`  <attribute name="deployable" value="false"/>`
	`</item-descriptor>`
	
	Note 2: If two properties have a many-to-many relationship, designate one as read-only by setting its writable attribute to false. This ensures that only one side of the relationship is updated, which prevents duplicate copies of the same asset versions and potential data corruption. Read-only properties can be displayed but not edited via the ATG Business Control Center.

required attribute	Set to true in <property> tags in a <table> tag for properties whose corresponding database columns are defined as NOT NULL. **Note:** If a property references an item that is defined in the database as NOT NULL but you cannot mark the property as required, indicate this by adding the references attribute tag and set its value to true.

Verify...	Requirements

`references` attribute	Set to true in `<table>`-nested `<property>` tags, for all properties that reference another item that is defined in the database as `NOT NULL`. In the case of one-to-many relationships between item descriptors, the underlying join tables are defined in the database as `NOT NULL`. Thus, the multi-table definition must set the `references` attribute to `true`, as in the following example:
	
	`<table name="dcs_cat_chldprd" type="multi"`
	`id-column-name="category_id"`
	`multi-column-name="sequence_num" shared-table-sequence="1">`
	
	`<property category-resource="categoryProducts"`
	`name="fixedChildProducts" data-type="list"`
	`component-item-type="product" column-name="child_prd_id"`
	`queryable="true" display-name-resource="fixedChildProducts">`
	
	`<attribute name="propertySortPriority" value="-4"/>`
	`<attribute name="references" value="true"/>`
	
	`</property>`
	`</table>`

Repository ID	Repository IDs in versioned repositories cannot contain special characters such as pound/hash (#), slash (/), and colon (:).
`id-column-names` attribute	In all `<table>` tags, specify the correct column name(s) in the database.
All reference properties	Define correctly. For example, all reference properties should specify an `item-type` attribute, not a `data-type` attribute.
`display-name-resource` attribute	Specify in all `<item-descriptor>` tags.

For detailed information on the attributes and tags specified above, see the *ATG Repository Guide*.

Modify the Repository Configuration

You must modify the repository component in order to configure a VersionRepository instance. To do so, layer on a configuration file for the component by placing it in the versioned module's main `config` directory.

The configuration file might look like this:

```
$class=atg.adapter.version.VersionRepository

versionManager=/atg/epub/version/VersionManagerService
versionItemsByDefault=true
```

The `versionManager` property should be set to the default VersionManager:

`/atg/epub/version/VersionManagerService`

The `versionItemsByDefault` property sets the default for versioning all item types:

- **False** (default)**:** Versioning is disabled for all item types except those whose repository definitions explicitly specify versioning.
- **True:** All item types are versioned except those whose repository definition explicitly excludes them from versioning.

The next section, Modify the Repository Definition, shows how the repository definition file specifies versioning for individual item types.

You can also set the `String` properties that are listed in the following table. Typically, you only need to set one of these properties if a default table or column name used to store version metadata matches a name in your existing, unversioned schema.

Property Name	Description
`assetVersionColumnName`	The column name of the `asset-version` item property. Default: `asset_version`
`branchColumnName`	The name of the column used to persist the branch of a version. **Note:** ATG Content Administration only uses the versioning system's main branch. Default: `branch_id`
`checkinDateColumnName`	The name of the column used to persist the date that a checked-in version of an asset was created. Default: `checkin_date`
`historyIdPropertyName`	The name of the `historyId` item property. This property is dynamically created and represents the unversioned, original ID property. Default: `historyIdProperty`
`isBranchHeadColumnName`	The name of the column used to persist the flag that determines if a version is the head of a branch. Default: `is_head`
`predecessorColumnName`	The column name of the `predecessor` item property. Default: `pred_version`

Property Name	Description
`versionDeletedColumnName`	The column name of the `version-deleted` item property. Default: `version_deleted`
`versionEditableColumnName`	The column name of the `version-editable` item property. Default: `version_editable`
`workspaceColumnName`	The column name of the `workspace` item property. Default: `workspace_id`

You can also use the Component Editor of the ATG Control Center (ACC) to modify a repository component's properties. If using the Component Editor, be sure to save changes to the correct module.

Modify the Repository Definition

When you use ATG Content Administration with your versioned module, the definition files for all versioned repositories are automatically extended to support asset version metadata. Thus, you should not modify the definition files to do so.

Depending on your requirements, you might need to modify the versioned repository's definition file in two ways:

- Specify whether items referenced by an item property are automatically checked out with the parent item.

  By default, all required single-item references are checked out with the parent item. This ensures that a referenced item cannot be deleted in one project while its parent item is checked out by another project. You can turn off this safeguard by setting the property's `auto-checkout` attribute to `false` as follows:

  ```
  <attribute name="auto-checkout" value="false"/>
  ```

- Set an item descriptor's `versionable` attribute to indicate whether to enable or disable versioning for individual item types.

The setting for the `versionable` attribute depends on the setting of the versioned repository's `versionItemsByDefault` property:

If versionItemsByDefault is set to:	Override for an item type by setting versionable to:
`true`: version all repository items	`false`: do not version items of this type
`false`: do not version any item types	`true`: version items of this type.

You modify the definition file for a versioned repository by layering on a definition file. Place the new definition file in the `config` directory of your versioned module—for example, `/MyCatalogVer/config/myApp`.

In some cases, a repository is defined using multiple definition files that are located across several application modules. If this is true for your versioned repository and you need to change each definition file, store the modified definition files in separate `config` directories in the versioned module—one for each `config` directory across all source modules. For example:

Location of source repository file:	Location of versioned repository file:
`/Catalog/config/myApp/catalog.xml`	`/CatalogVer/config/myApp/catalog.xml`
`/MyCatalog/config/myApp/catalog.xml`	`/MyCatalogVer/config/myApp/catalog.xml`

Note: If you create additional `config` directories in your application's versioned module, add them to the `ATG-Config-Path` variable specified in the versioned module's manifest file. For more information on application modules, see the *ATG Programming Guide*.

It's also important to note that, while the definition files for versioned repositories are extended automatically at startup, in a subsequent setup step you must manually create and install the corresponding database schema (see Create and Install the Versioned Database Schema).

Set the Behavior of Versioning Caches

Versioned repositories have two caches for each item type, which maintain information about an item's latest version:

- *HeadOfLineCache* caches information that identifies the item's head version on a given branch.
- *CurrentVersionItemCache* caches CurrentVersionItem objects, which encapsulate the most recent versions of repository items on a given branch.

Together, these caches help optimize performance of versioned repositories, by avoiding overhead otherwise incurred through repeated queries for an item's head version, and creation of redundant CurrentVersionItem objects.

By default, the size of versioning cache sizes and when they time out are set by the item descriptor's `item-cache-size` and `item-cache-timeout` attributes, respectively. In general, you can rely on these settings for versioning caches if they are set appropriately for the item type itself. If desired, you can explicitly set the behavior of versioning caches independently of the item cache through the following item descriptor attributes:

- `headOfLineCacheSize`
- `currentItemCacheSize`
- `headOfLineCacheTimeOut`
- `currentItemCacheTimeOut`

The timeout attributes are set in milliseconds. If no timeout is set for versioning caches or the item cache, the HeadOfLineCache and CurrentVersionItemCache use the VersionRepository properties `maxHeadOfLineCacheTimeout` and `maxCurrentVersionItemCacheTimeout`, respectively.

Modify the Secured Repository Definition

If you copy definition files for the secured repositories that sit on top of your repositories, you must modify those files in order to provide access to their data via the ATG Business Control Center.

Specifically, you must modify the `<descriptor-acl>` and `<creation-base-acl>` attributes of the item descriptors to include the four ATG Content Administration roles that are provided with the product:

- `EPub-Super-Admin`
- `EPub-Admin`
- `EPub-Manager`
- `EPub-User`

To give users access to secured assets within the ATG Business Control Center, modify the access control lists (ACLs) for the item descriptors to include ATG Content Administration roles. If you also configure item-level security, you must also modify the ACLs for the individual items to include ATG Content Administration roles.

The following XML for the `catalog` item descriptor shows how access rights might be defined. The ACLs below are similar to those defined for the item descriptors and items in the PublishingRepository.

```
<item-descriptor name="catalog">

  <!-- The ACL that applies to the item view/descriptor -->
  <descriptor-acl value="Profile$role$epubSuperAdmin:read,write,create,
delete;Profile$role$epubAdmin:read,write,create,delete;
Profile$role$epubManager:read,write,create,delete;
Profile$role$epubUser:read;Admin$role$administrators-
group:read,write,create,delete"/>

  <!-- The property that the ACL will be stored in -->
  <acl-property name="acl"/>

  <!-- An ACL fragment that is assigned to all new items -->
  <creation-base-acl value="Profile$role$epubSuperAdmin:read,write,
list,destroy,read_acl,write_acl;Profile$role$epubAdmin:read,write,
list,destroy,read_acl,write_acl;Profile$role$epubManager:read,write,
list,destroy;Profile$role$epubUser:read,list;
Admin$role$administrators-group:read,write,list,destroy,read_acl,
write_acl"/>

</item-descriptor>
```

These changes get you started with repository data. As you customize the ATG Business Control Center to meet your needs, you are likely to further modify access rights for various item descriptors and items to subsets of ATG Content Administration users. For more information on ATG Content Administration security, see the chapter Managing User Access and Security. For more information on secured repositories, see the *Secured Repositories* chapter in the *ATG Repository Guide*.

Create and Install the Versioned Database Schema

In order to store asset version metadata, the asset management server's database schema requires additional columns for the primary tables used by your application's repositories, and additional tables to store versioning information. This section describes the following tasks:

- Create the versioned schema.
- Configure a versioned repository to use shared tables.
- Install the versioned database schema.

Note: You only create the versioned schema for repositories that store application repository assets. ATG Content Administration provides the versioned schema for the PublishingFileRepository, which stores the metadata for application file assets.

See Appendix A: Database Schema for descriptions of the ATG Content Administration tables and columns.

Create the Versioned Schema

To modify the asset management server's database schema to store versioning data, follow these steps:

Note: Repeat this process for each VersionRepository for which you need to create a DDL file.

1. Copy each database DDL file that you plan to modify.

   Note: Do not edit the original DDL files to be used with the production database; only modify the copies.

2. Add the following columns to every table that represents a primary table for an item descriptor:

```
asset_version      INT           NOT NULL
workspace_id       VARCHAR(40)   NOT NULL
branch_id          VARCHAR(40)   NOT NULL
is_head            NUMERIC(1)    NOT NULL
version_deleted    NUMERIC(1)    NOT NULL
version_editable   NUMERIC(1)    NOT NULL
pred_version       INT           NULL
checkin_date       TIMESTAMP     NULL
```

   Note: If the existing schema has a column that uses one of these names, you can use a different name for the new column. In this case you must also configure the VersionRepository component to override the default column name and use the new name. See Modify the Repository Configuration for more information.

3. Add the following column to every table that represents an auxiliary or multi table in an item descriptor:

   ```
   asset_version   INT   NOT NULL
   ```

   See also Configure a Versioned Repository to Use Shared Tables.

4. Change all primary keys to composite primary keys, composed of the original primary key column(s) and the `asset_version` column.

5. Create an index for the `workspace_id` and `checkin_date` columns that are added to each primary table.

6. Remove from all tables:

   - All foreign key references.
   - All unique constraints on columns. Also remove `unique` attributes from all `<property>` tags. For more information, see the discussion on unique properties in the *ATG Repository Guide*.
   - All unique indexes on columns.

The following table describes the additional columns that are required for all primary tables:

Column Name	Description
`asset_version`	Counter that specifies the version of the asset.
`branch_id`	ID used to persist the branch of a version. ATG Content Administration always sets this to the ID of the main branch.
`is_head`	Flag that determines whether a version is the head of a branch.
`checkin_date`	Stores the check-in date of a version. Null for working versions of assets.
`version_deleted`	Flag that indicates whether the asset version is a deleted version.
`version_editable`	Flag that indicates whether the asset version is an editable version. That is, the version is a *working version* in a workspace, where modifications to it can be made.
`pred_version`	The asset version upon which this version was based. For example, if you create version 2 by checking out version 1, version 2's *predecessor version* is version 1.
`workspace_id`	The ID of the workspace where the asset version was initially created.

For example, the original DDL for table `type_x` might look like this:

```
create table type_x (
       type_x_id       VARCHAR(16)    NOT NULL,
       name            VARCHAR(128)   NULL,
       type_y_ref_id   VARCHAR(16)    NULL,
       FOREIGN KEY (type_y_ref_id) REFERENCES type_y (type_y_id),
       PRIMARY KEY (type_x_id)
);
```

The modified DDL for table type_x DDL looks like this:

```
create table type_x (
       type_x_id          VARCHAR(16)    NOT NULL,
       asset_version      INT            NOT NULL,
       branch_id          VARCHAR(40)    NOT NULL,
       is_head            NUMERIC(1)     NOT NULL,
       version_deleted    NUMERIC(1)     NOT NULL,
       version_editable   NUMERIC(1)     NOT NULL,
       workspace_id       VARCHAR(40)    NOT NULL,
       pred_version       INT            NULL,
       checkin_date       TIMESTAMP      NULL,
       name               VARCHAR(128)   NULL,
       type_y_ref_id      VARCHAR(16)    NULL,
       PRIMARY KEY (type_x_id, asset_version)
);
create index type_x_workspace_id on type_x (workspace_id);
create index type_x_workspace_id on type_x (checkin_date);
```

Configure a Versioned Repository to Use Shared Tables

The following constraints apply to versioned repositories with shared database tables:

- A versioned repository cannot support more than two shares of a table in a single repository definition.
- In a one-to-many relationship using a shared table, the table whose type is not `multi` cannot be a primary table. It must be an auxiliary table.

If you need to use table sharing with a versioned repository, also perform these steps:

1. To the shared table's database schema definition, add the version column `sec_asset_version`. This column must be defined as part of the primary key.

   Note: This name is configurable. To change it, use the `secondAssetVersionColumnName` property in the versionRepository component.

2. In the repository definition that includes the shared table, specify an additional attribute called `shared-table-sequence`. Add the attribute to the `<table>` tag of the shared table (along with `name` and `multi-column-name`, for example). The `shared-table-sequence` attribute takes values 1, 2, 3, and so on. The side that wants

to associate the `sec_asset_version` column as its asset version column must have a value of 2 for this attribute. The table tag where this attribute is set to 2 is considered to be the owner of the `sec_asset_version` column, meaning that the side that wants to associate the column as its asset version column must have a value of 2 for this attribute. The value must not be changed after data is entered into the shared table.

Install the Versioned Database Schema

After you configure your database for ATG Content Administration (described in the previous chapter, Setting Up an Asset Management Server), finish installing your versioned database schema by running the custom DDL scripts that you created earlier .

Register the Versioned Repositories

All versioned repositories must be registered with ATG Content Administration as follows:

1. In the asset management server's `localconfig` layer, create this properties file:

   `/atg/registry/ContentRepositories.properties`

2. Set its `$class` property as follows:

   `$class=atg.repository.nucleus.RepositoryRegistryService`

3. Add the versioned repositories to the list property `initialRepositories`, in this format:

   ```
   initialRepositories+=\
      versioned-repository[,versioned-repository]...
   ```

   For example:

   ```
   $class=atg.repository.nucleus.RepositoryRegistryService
   ...
   initialRepositories+=\
      /atg/myApp/MyRepositoryVer1,\
      /atg/myApp/MyRepositoryVer2
   ```

Configure JSP File Asset Support

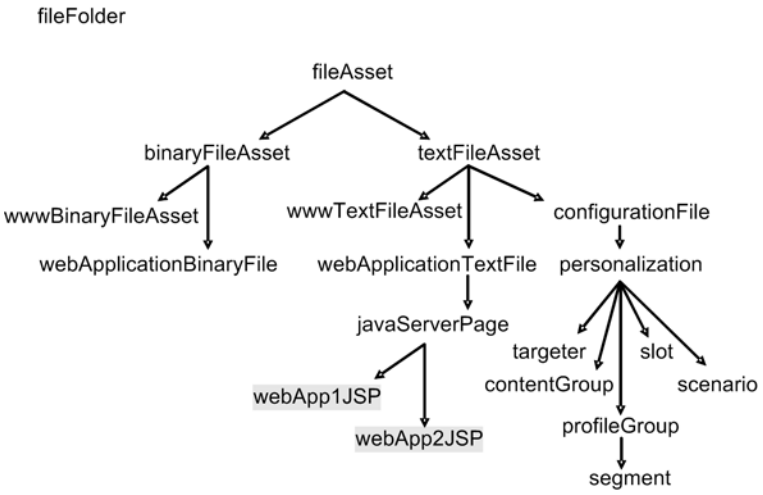
ATG Content Administration supports versioning and management of JSPs and their deployment to Web applications running from an exploded directory in the target environment.

Like all file assets, the JSPs for a Web application have the following characteristics:

- They are stored in the content development environment in a versioned file system.
- Their metadata is stored in the `/atg/epub/file/PublishingFileRepository`.

- They are exposed as files in the content development environment via a ContentRepositoryVFSService.
- They are deployed as files to a VFS at the same Nucleus location in the target environment.

To support JSPs for Web applications, you must extend the PublishingFileRepository so it defines a subtype of the `javaServerPage` item type for each Web application to manage, as shown in the figure below:



You must also configure two virtual file systems for each subtype of `javaServerPage`:

- The first VFS exposes the items as files in the content development environment.
- The second VFS exposes items as files in the target environment where they are deployed as actual files.

Deployment Requirements and Constraints

The following requirements and constraints apply to JSP deployment support:

- The ATG servers on each target site must be running the Publishing Agent, which is installed with the ATG platform.
- The Web application in the target environment to which JSPs are deployed must be running from an exploded directory; it cannot be run from a Web Archive (WAR) file.
- You must import and maintain all the JSPs and only the JSPs for the given Web application in the content development environment. While other types of files might be found in a Web application—for example, servlets—they cannot be maintained in the content development environment, nor deployed to the Web application.
- The VFS in the target environment that contains the Web application JSPs must be configured for online deployment.

  Deployment of JSPs in *switch* mode is not supported because the inactive directory where the JSPs are first deployed cannot contain files other than the JSPs that are

managed in and deployed from the content development environment. Because most Web applications contain other resources such as servlets, an inactive directory cannot be used.

- The deployment process only deploys the JSPs for a Web application to the target-side VFS; it does not restart the Web application. As a result, new and updated JSPs are compiled as each is requested.
- To recompile all JSPs for a given Web application, you must configure the server to explicitly precompile the JSPs. You can accomplish this by configuring a DeploymentEventListener to initiate the precompilation process after a deployment is complete.

  Note: The target site must have an internet connection during compilation; otherwise, the page compiler cannot verify DTDs.

Configuration Steps

To configure the asset management server to manage and deploy JSPs to a Web application, follow these steps:

1. Copy the Web application module to the asset management server.
2. Create the Web application's versioned module.
3. Configure targets for deployments to the Web application.
4. Import Web application's JSPs into the PublishingFileRepository. You can do so when you import the rest of your file assets (see Import Initial File Assets later in this chapter).

Copy the Web Application Module to the Asset Management Server

In order to configure the asset management server to support JSP file assets, you must copy the module that contains your Web application's Web Archive (WAR) file or exploded directory from the production server to the asset management server.

If the module also contains repository definition files and other related resources, this step might be already complete. If not, copy over the module now.

You do not need to copy over all of the resources contained in the module; but you must include the JSPs that are imported into the `/atg/epub/file/PublishingFileRepository`. If you intend to run the Web application on the asset management server, you should include all required resources.

Create the Web Application's Versioned Module

Your Web application requires a versioned module that acts as an additional configuration layer for storing all resources required to manage the Web application's JSP file assets. The procedure for setting up the versioned module for a Web application is basically the same as the procedure described earlier in this chapter for non-Web applications (see Create the Versioned Module), although you structure and configure module resources differently for JSP-based applications.

To set up the versioned module for a Web application, copy the demonstration `WebAppRefVer` module that is distributed with ATG Content Administration. For more information, see WebAppRef Reference Implementation.

If you copy the generated configuration files into another versioned module, be sure to modify the module's manifest file accordingly so all required modules are included.

After you copy the structure and contents of the demonstration module, edit the following module resources:

- Manifest file
- Configuration files
- lib file (I18N)
- liveconfig file

Manifest File

The manifest file for the versioned module is located in:

`<ATG10dir>/`*`versioning-module-name`*`/META-INF/`

Specify the Web application's source module and `Publishing.base` as required modules. For example:

```
ATG-Required: Publishing.base MyWebApp
```

You must specify the Web application's source module if it contains other resources, such as the original definition and configuration files for one or more repositories. This might be necessary for other reasons as well. For example, you might intend to run the Web application in the content development environment, using file mirroring to copy the new head versions of each JSP file asset to the Web application upon their check-in.

If there is no identified dependency on the Web application's source module, you can omit it from the list of required modules.

Configuration Files

The necessary configuration files for the versioned module are located at `<ATG10dir>/`*`versioning-module-name`*`/config/` or a subdirectory:

`publishingFiles.xml` file for the `/atg/epub/file/PublishingFileRepository`	Extends the repository to support the new item descriptor that represents the Web application's JSP file assets.
`.properties` configuration file for the `ContentRepositoryVFSService`	Sits on top of the PublishingFileRepository and exposes the Web application's JSPs as files. Note that the VFS name is composed of the source module's name plus `FileSystem`.

Configuration file for the `/atg/epub/version/VersionManagerService`	Adds the new VFS to the list of VFSs it must manage; this list is specified in the `VersionManagerService.versionedVirtualFileSystems` property.
Type mapping component	Converts the Web application's JSPs into content repository items. The component name is composed of the item descriptor name plus `TypeMapping`. This component is used when importing the JSPs into the content development environment using the Repository Loader.
Configuration file for the component `/atg/epub/file/typemappers/PublishingTypeMapper`	Adds the new type mapping component to the list of available type mapping components. For use when importing the JSPs into the content development environment using the Repository Loader

lib File (I18N)

A single resource configuration file that stores the resource strings for the repository definition file, used for internationalization and localization purposes. The resource file name is composed of the item descriptor name plus `FileRepositoryResources`.

The resource file is located in:

`<ATG10dir>/`*`versioning-module-name`*`/lib/classes/atg/epub/file/`

liveconfig File

A `publishingFiles.xml` file for the `/atg/epub/file/PublishingFileRepository` that sets the new item descriptor's cache mode to distributed in the `liveconfig` layer. This file is located at

`<ATG10dir>/`*`versioning-module-name`*`/liveconfig/atg/epub/file`

Configure Targets for Deployments to the Web Application

The target site requires a configuration file for the VFS `SelectiveDeleteVFSService`, to which the JSPs are deployed from the content development environment. Like all VFSs in your deployment targets, this VFS is located at the same location in Nucleus as the corresponding VFS in the content development environment, `ContentRepositoryVFSService`.

For an example, see:

`<ATG10dir>/Publishing/WebAppRef/config/atg/epub/file/WebAppRefFileSystem.properties`

To complete setup of your target server for deployment of JSPs to the Web application:

1. Create the configuration files and copy them to the target server, placing them in the `config` directory of the Web application's source module.
2. If necessary, modify the `localDirectory` property of the VFS to point to the Web application's exploded directory.

WebAppRef Reference Implementation

ATG Content Administration includes a reference implementation that shows how to set up a Web application to be managed by ATG Content Administration. The reference implementation includes two modules:

- Publishing.WebAppRef
- Publishing.WebAppRefVer

Publishing.WebAppRef

An example of a standard module that you can expect to see on a target server. This module contains a Web application `webappref`, and the configuration files required to deploy JSPs to the Web application—that is, a `SelectiveDeleteVFSService` VFS that lives at the same Nucleus location as its counterpart in the content development environment. To examine this module's resources, browse the `<ATG10dir>/Publishing/WebAppRef` directory and its subdirectories.

Note: While this module is an ATG Content Administration sub-module, this is for packaging purposes only; it does not require ATG Content Administration in any way.

Publishing.WebAppRefVer

An example of a versioned module that you can expect to see on an asset management server. This module illustrates the configuration of a versioned module to support a Web application. It is configured based on the `Publishing.WebAppRef` source module. It contains all the resources required to support and manage the Web application's JSPs in the content development environment, including:

- A repository definition file that extends `/atg/epub/file/PublishingFileRepository` in order to define a subtype of the `javaServerPage` item type
- A `ContentRepositoryVFSService` VFS that exposes the JSP file assets as files for deployment purposes

To examine this module's resources, browse the `<ATG10dir>/Publishing/WebAppRefVer` directory and its subdirectories.

Note: While the module is also an ATG Content Administration sub-module, this is for packaging purposes only; it is only required that the versioned module specifies an ATG Content Administration module as required in its manifest file.

Configure Support for Other File Assets

By default, the `/atg/epub/file/PublishingFileRepository` is configured to support file assets that are deployed to the following virtual file systems on a target site:

- `/atg/epub/file/ConfigFileSystem`: ATG personalization assets such as scenarios and targeters.
- `/atg/epub/file/WWWFileSystem`: static Web assets such as JSPs.

You extend the `/atg/epub/file/PublishingFileRepository` to support additional item types only if you require additional deployment destinations on a target site. If the asset destination is a Web application where you wish to deploy JSPs, refer to the previous section, Configure JSP File Asset Support. For all other asset destinations, follow the process described in this section.

You extend the PublishingFileRepository in the following steps:

1. Extend the PublishingFileRepository repository definition to support new item types.
2. Extend the SecuredPublishingFileRepository repository definition to support the new item types.
3. Configure a custom VFS to expose new item types in the content development environment.
4. Set up the VersionManagerService to manage the custom VFS.
5. Optionally, configure TypeMapping components for new item types to support file imports with the Repository Loader.
6. Configure a custom VFS on deployment targets.
7. Customize the ATG Business Control Center to support new item types.

Note: The sections that follow illustrate each step with an example, where the PublishingFileRepository is extended to support a new VFS asset destination `/mycompany/FtpFileSystem`, which is located on a production machine that is running an FTP server.

Extend the PublishingFileRepository Definition

You extend the repository definition of the PublishingFileRepository in order to support new item types. Because files are either in text or binary format, most extensions require two new item types—one that extends `textFileAsset` and another that extends `binaryFileAsset`. You can extend either or both, as needed. You can also create multiple item types with the same supertype, as with the `personalization` item type.

Two requirements are important for custom item types:

- A custom item type for text files must extend `textFileAsset`
- A custom item type for binary files must extend `binaryFileAsset`.

Refer to the figure shown earlier in Versioned Content Repository, which shows the item type hierarchy of the PublishingFileRepository.

By default, both textFileAsset and binaryFileAsset are extensions of fileAsset. The content property for fileAsset is defined as a transient property of a custom type, VersionFilePropertyDescriptor:

```
<property name="content"
property-type="atg.epub.file.VersionFilePropertyDescriptor"
</property>
```

VersionFilePropertyDescriptor is an extension of FilePropertyDescriptor that returns the content property as a java.io.File. It is required so ATG Content Administration can store the file asset contents on a file system. Make sure that your custom items do not override the VersionFilePropertyDescriptor property.

To extend the repository definition of the PublishingFileRepository, modify the repository's definition file publishingFiles.xml:

1. Define the new item descriptors.
2. Add the new item types as options to the sub-type-property definition of the fileAsset parent item descriptor.

Modify the definition by layering on a new definition file. Place the new file in the config directory of your versioned module at /atg/epub/file/publishingFiles.xml.

In the FTPFileSystem example, you want to manage versioned FTP assets that are ultimately deployed as text and binary files to a target FTP server. The repository must be extended to support two new item types:

- ftpTextFileAsset
- ftpBinaryFileAsset

The new definition file to layer on looks like this:

```
<item-descriptor name="ftpTextFileAsset" super-type="textFileAsset"
  content="true" sub-type-value="ftpTextFileAsset"
  display-name="FTP text file" item-cache-size="1000"
  query-cache-size="500">
</item-descriptor>

<item-descriptor name="ftpBinaryFileAsset" super-type="binaryFileAsset"
  content="true" sub-type-value="ftpBinaryFileAsset"
  display-name ="FTP binary file" item-cache-size="1000"
  query-cache-size="500">
</item-descriptor>

<item-descriptor name="fileAsset">
  <table name="epub_file_asset">
    <property name="type">
      <option value="ftpTextFileAsset" code="10001"/>
      <option value="ftpBinaryFileAsset" code="10002"/>
```

```
    </property>
  </table>
</item-descriptor>
```

Extend the SecuredPublishingFileRepository Definition

A default secured repository `/atg/epub/file/SecuredPublishingFileRepository` is configured on top of the PublishingFileRepository, which lets you set item descriptor-level and item-level security on the PublishingFileRepository.

You must modify the secured repository definition of the `SecuredPublishingFileRepository` in order to support the new item types `ftpTextFileAsset` and `ftpBinaryFileAsset` described earlier. To do so:

1. Modify the repository definition by layering on a new definition file.
2. Place the new file in the `config` directory of your versioned module at `/atg/epub/file/publishingFileSecurity.xml`.

At this stage in the setup process, it is unlikely you modified `publishingFileSecurity.xml`. If so, define the item descriptor-level security for the new item types as it is defined for all existing item types. This sets up the proper access rights for the ATG Content Administration roles that are provided by default.

In the case of the `FTPFileSystem` example, the new secured definition file to layer on looks like this:

```
<item-descriptor name="ftpTextFileAsset">
  <acl-property name="acl"/>
  <descriptor-acl value="Profile$role$epubSuperAdmin:read,write,create,
delete;Profile$role$epubAdmin:read,write,create,delete;Profile$role$epubUs
er:read;Admin$role$administrators-group:read,write,create,delete;
Admin$role$everyone-group:read"/>
</item-descriptor>

<item-descriptor name="ftpBinaryFileAsset">
  <acl-property name="acl"/>
  <descriptor-acl value="Profile$role$epubSuperAdmin:read,write,create,
delete;Profile$role$epubAdmin:read,write,create,
delete;Profile$role$epubUser:read;Admin$role$administrators-
group:read,write,create,delete;Admin$role$everyone-group:read"/>
</item-descriptor>
```

Later in the development process, you can create the principals—for example, roles, and organizations—required for your content development environment, and modify the security of the SecuredPublishingFileRepository accordingly. For more about security, see the chapter Managing User Access and Security.

Configure a Custom VFS to Expose New Item Types

After you extend the repository definition of the PublishingFileRepository to support new item types, you must configure a custom VFS on the asset management server to expose items of those new types as files. Later, you configure a custom VFS at the same Nucleus location on the deployment target(s). The files exposed via the asset management server-side VFS are deployed to each VFS at that Nucleus location on the target site.

To configure the custom VFS on the asset management server, create a configuration file for the VFS. The VFS must be an instance of class `atg.vfs.repository.ContentRepositoryVFSService`, which is a VFS implementation that enables items stored in a content repository to be accessed as if they were stored in a file system.

The following table describes each property you should configure:

Property	Description
`contentRepository`	The content repository to expose as a file system. Always set this property to `/atg/epub/file/SecuredPublishingFileRepository`. This restricts file asset access to users with the proper permissions.
`itemDescriptorNames`	A comma-separated list of item descriptor names that are viewable and accessible through the VFS. **Note:** Each item descriptor can be exposed via just one VFS. An item descriptor specified in this property for a given VFS cannot be specified in this property for another VFS.
`mutableFolderDescriptorName`	The item descriptor in the repository specified in the `contentRepository` property that represents a folder in the content repository. Always set this property to `fileFolder`, as this item descriptor represents a folder in the PublishingFileRepository.

Place the configuration file for the custom VFS at a Nucleus location in the `config` directory of your versioned module. (Later, you must configure a VFS at the same Nucleus location on your production target.)

In the `FTPFileSystem` example, you create the actual VFS component on the asset management server, locating it in Nucleus at `/mycompany/FTPFileSystem`. The configuration file for `FTPFileSystem` looks like this:

```
$class=atg.vfs.repository.ContentRepositoryVFSService
```

```
contentRepository=/atg/epub/file/SecuredPublishingFileRepository
itemDescriptorNames=ftpTextFileAsset,ftpBinaryFileAsset
mutableFolderDescriptorName=fileFolder
```

Note: Because `ftpTextFileAsset` and `ftpBinaryFileAsset` are exposed via `FTPFileSystem`, they must not be exposed via any other VFS in the content development environment.

Set Up the VersionManagerService to Manage the Custom VFS

As with any virtual file system that is configured on top of the PublishingFileRepository, you must add the custom VFS to the list of VFSs that are managed by the `/atg/epub/version/VersionManagerService`.

In the `FTPFileSystem` example, this step requires you to add `/mycompany/FTPFileSystem` to the VersionManagerService's list of VFSs.

For more information on the VersionManagerService and completing this step (including a configuration file example), see Configure the VersionManagerService in this chapter.

Configure TypeMapping Components for New Item Types

Note: This step is required only if you import file assets with the Repository Loader.

As described later in Import Initial File Assets, you use the ATG platform's Repository Loader to import file assets into the ATG Content Administration system. The Repository Loader uses a set of TypeMapping components to convert files of different types into content repository items, and it uses a `TypeMapper` component to determine which TypeMapping component to use for a given file.

ATG Content Administration provides a set of TypeMapping components for the default content item descriptors in the PublishingFileRepository, as well as a `PublishingTypeMapper` component that defines this array of TypeMapping components. These components are located in:

```
<ATG10dir>/Publishing/base/config/atg/epub/file/typemappers
```

However, if you intend to import file assets of the custom item types with the Repository Loader, you need to configure additional TypeMapping components to support the custom content item descriptors:

1. Configure a TypeMapping component for each new item descriptor that you added to the PublishingFileRepository's definition file.

   Place the configuration files for the new TypeMapping components in the `config` directory of your versioned module at `/atg/epub/file/typemappers/`.

2. Modify the `PublishingTypeMapper` to support the new TypeMapping components you created in step 1.

   Modify the `PublishingTypeMapper` by layering on a new configuration file for the component. Place the new configuration file in the `config` directory of your versioned module at `/atg/epub/file/typemappers/`.

For more information on these tasks, see the *Repository Loader* chapter in the *ATG Repository Guide*.

In the FTPFileSystem example, the TypeMapping component for the ftpTextFileAsset item descriptor is named FTPTextFileAssetTypeMapping.properties and looks like this:

```
$class=atg.repository.loader.TypeMappingImpl

itemDescriptorName=ftpTextFileAsset
repository=/atg/epub/file/PublishingFileRepository
contentHandler=/atg/dynamo/service/loader/typemapping/Xml2RepositoryConten
tHandler
contentIsXML=false
parseContent=false
updatePropertyConfiguration=CONTENT_ITEM_DESCRIPTOR_ID_AND_PATH_PROP
applicationLogging=/atg/epub/file/VersionedLoaderEventListener
contentRootPathProvider=/atg/epub/file/VersionedLoaderEventListener
```

Similarly, the TypeMapping component for the ftpBinaryFileAsset item descriptor can be named FTPTBinaryFileAssetTypeMapping.properties:

```
$class=atg.repository.loader.TypeMappingImpl

itemDescriptorName=ftpBinaryFileAsset
repository=/atg/epub/file/PublishingFileRepository
contentHandler=/atg/dynamo/service/loader/typemapping/Xml2RepositoryConten
tHandler
contentIsXML=false
parseContent=false
updatePropertyConfiguration=CONTENT_ITEM_DESCRIPTOR_ID_AND_PATH_PROP
applicationLogging=/atg/epub/file/VersionedLoaderEventListener
contentRootPathProvider=/atg/epub/file/VersionedLoaderEventListener
```

Finally, the new configuration file to layer on for the PublishingTypeMapper looks like this:

```
$class=atg.epub.loader.PublishingTypeMapper

typeMappings+=\
  FTPTextFileAssetTypeMapping,\
  FTPBinaryFileAssetTypeMapping
```

For a description of properties you can configure for TypeMapping and TypeMapper components, see the *Repository Loader* chapter in the *ATG Repository Guide*. For more information on the /atg/epub/file/VersionedLoaderEventListener component (referenced in the configuration file examples above), see the *ATG Repository Guide*.

Configure a Custom VFS on Deployment Targets

As with any VFS that is configured on top of the PublishingFileRepository, a custom VFS must have a corresponding VFS at the same Nucleus location on the appropriate servers in each staging or production target where you intend to deploy the files.

In the `FTPFileSystem` example, there might be a single production target, which includes a single FTP server. You need to configure a VFS named `/mycompany/FTPFileSystem` on that FTP server.

For detailed information on configuring custom VFSs in deployment targets, see Configure Custom Target VFSs for Switch Deployments or Configure Custom VFSs for Online Deployments in the *Setting Up Deployment* chapter.

Customize the ATG Business Control Center to Support New Item Types

Finally, you need to configure the ATG Business Control Center so authors and editors can access and view file assets of the new item types. To do so, you must set up view mappings for each of the file asset types you added. Use the view mapping provided for the default text file asset type as a template for how to set up a custom view mapping for a file asset type.

You can view the sample view mappings through the Publishing > View Mappings option in the ACC; however, you might need to configure the appropriate Control Center Groups options for your ACC user profile in order to display the View Mappings option.

For more information, refer to the chapter Customizing Asset Display.

Configure the VersionManagerService

The VersionManagerService manages development lines that are used by configured VersionRepository components, such as the main branch and project workspaces. The VersionManagerService has the following Nucleus path:

```
/atg/epub/version/VersionManagerService
```

The VersionManagerService has two map properties that identify versioned repositories and virtual file systems that it manages:

- versionedRepositories
- `versionedVirtualFileSystems`

versionedRepositories

Stores a map of the versioned repositories to manage. The mapping is between a short name used by the VersionManager and the fully qualified Nucleus path of a VersionRepository.

By default, this map includes two repositories:

```
versionedRepositories+=\
  PublishingFiles=/atg/epub/file/SecuredPublishingFileRepository,\
  ProcessData=/atg/epub/process/ProcessDataRepository
```

Add to this map each VersionRepository that you create and configure, as described in Configure Repository Asset Support.

If a versioned repository has a secured repository instance configured on top, specify the secured repository and not the underlying versioned repository; otherwise, the VersionManager cannot access the repository with the specified security settings. The name specified in the `repositoryName` property of both the secured repository and the underlying VersionRepository must be identical.

For more information about secured versioned repositories, see VersionRepository Security later in this chapter.

versionedVirtualFileSystems

Stores a map of the virtual file systems to manage. The mapping is between a short name used by the VersionManager and the fully qualified Nucleus path of a `ContentRepositoryVFSService`.

By default this map includes only the `/atg/epub/file/ConfigFileSystem` and the `/atg/epub/file/WWWFileSystem`.

If `/atg/epub/file/PublishingFileRepository` is extended to support additional item types (see Configure Support for Other File Assets), add to this map any VFSs you configured to support those item types.

For example, to add two additional repositories and a VFS, you can layer on a configuration file like this (note use of the appending operator +=):

```
versionedRepositories+=\
  Catalog=/myApp/Catalog,\
  PressReleases=/myApp/PressReleases

versionedVirtualFileSystem+=\
  FTPFileSystem=/mycompany/FTPFileSystem
```

Optimizing Merge, Revert, and Check-in Performance

For merge, revert, and check-in operations that involve a large number of assets, ATG Content Administration automatically uses optimization features to improve database performance. When optimization is disabled, these operations iterate over workspace assets and issue SQL statements for each one. When optimization is enabled, a single SQL operation is performed on all workspace assets.

By default, the threshold number of workspace assets required to trigger an optimized operation is set to 500. To change this number, set the following VersionManagerService properties to an integer value:

- `mergeOptimizationThresholdCount`

- `revertOptimizationThresholdCount`
- `workspaceOptimizationThresholdAssetCount` specifies the minimum number of workspace assets required to trigger optimized check-in.

You can set these properties to the same or different values. To ensure optimization for a given operation, set the corresponding property to 0.

Note: For Oracle, the setting for `mergeOptimizationThresholdCount` works only if the database does not contain LONG datatype columns. To enable merge optimization, change the following column datatypes: LONG or LONG VARCHAR to CLOB, and LONG RAW to BLOB.

Optimized operations require table joins and invalidation of GSA caches. Thus, optimization settings can enhance performance for large loads but might adversely affect performance for small loads.

To disable optimized merge or revert, set two VersionManagerService properties to false:

- `useOptimizedReplicationForMerge`
- `useOptimizedReplicationForRevert`

Optimizing Workflow Performance

If projects routinely include a very large number of assets—for example, 10 thousand or more—you should optimize asset locking by setting two VersionManagerService properties as follows:

Property	Setting
`useOptimizedAssetLocking`	true
`assetLockOptimizationThresholdCount`	Set to the desired threshold. A setting of 0 ensures that optimized asset locking is always in effect.

Also, disable secure repositories by removing the applicable secured versioned repositories from the VersionManagerService's versionedRepositories property and specifying the corresponding unsecured versioned repositories instead (see Disabling a Secured Repository).

The number of project assets that might require optimization can vary, depending on other site-specific factors such as processing speed and network load.

Import Initial Repository Assets into Versioned Repositories

After you install the versioned database schema on your asset management server, you can migrate the repository data that is to be the initial version of data on the asset management server from the repositories on your production or staging server.

You import repository data into the ATG Content Administration system in two steps:

1. Export repository data from the production server.
2. Import repository data into the asset management server.

To perform these steps, use the `exportRepository` and `importRepository` scripts that are provided with ATG Content Administration, and described in this chapter.

Export Repository Data from the Production Server

Caution: Before starting this process, it is strongly recommended that you back up production server content.

You use the `exportRepository` script to export the contents of one or more standard repositories on the production server to a `.jar` data file. After doing so, you run the `importRepository` script, which imports the exported data into one or more versioned repositories on the asset management server.

The repository data that you export serves as the initial version of repository data that is managed by ATG Content Administration. If you also plan to manage file assets with ATG Content Administration, you must import the files from the same logical site utilized in this step. Together, the repository and file data become the initial version of ATG Content Administration data and the earliest version that you can deploy back to your production servers.

Run exportRepository

To run `exportRepository`:

1. Navigate on the production server to `<ATG10dir>/home/bin`
2. Enter the `exportRepository` command.

For example:

```
exportRepository –m Catalog
                –file /users/joe/CatalogExport.jar
                –r /myApp/Catalog
```

Import Repository Data into the Asset Management Server

The `importRepository` script imports the contents of the data file created by `exportRepository` into the versioned repositories on the asset management server. By default, the `importRepository` script performs these tasks:

1. Deletes all data in the SQL repositories affected by the source data file.
2. Imports the data into the repositories and checks it in as the initial version of repository data in the versioning system.

You can modify this behavior through various `importRepository` switches that apply to versioned and non-versioned repositories.

Run importRepository

To run `importRepository`:

1. Copy the data file created by `exportRepository` to the asset management server.
2. Navigate to the `<ATG10dir>/home/bin` directory on the asset management server
3. Enter the `importRepository` command.

For example:

```
importRepository –m CatalogVer
                 –file /users/joe/CatalogExport.jar
                 –workspace initialcheckin
```

Import Initial File Assets

Note: Before reading this section or completing this step, review the *Repository Loader* chapter in the *ATG Repository Guide*.

The following procedure shows how to use the Repository Loader's RMI client RLClient to import file asset metadata into the PublishingFileRepository, and write the file contents to the file system.

1. If you are running on Windows, set this property:

   `/atg/epub/file/PublishingFileRepository.pathSeparator`

   to backslash (\):

   `pathSeparator=\\`

   Backslash is an escape character in Java properties files. Alternatively, you can set this property using a single backslash via the ACC.
2. If you import any manually created targeters that store their rule sets in separate `.rules` files, modify each applicable `RuleSetService` configuration file to specify the virtual file system that stores the `.rules` file.
3. On the asset management server, create a manifest file that identifies the production server directories and files to import into the PublishingFileRepository.

   Each `<add>` tag should include a `type-mapping` attribute that specifies the correct TypeMapping component to use for the given file or folder, and its body should specify the file or folder's path.

Because you are performing the import in manifest mode, the manifest must specify all folders to be imported. They are not automatically imported as folder content items.

For example, to import a directory and two HTML files into the PublishingFileRepository, add three <add> tags to the manifest:

```
<manifest>
  <add type-mapping=
     "/atg/epub/file/typemappers/FileFolderTypeMapping">
     /users/joe/import/myHtmlFiles
  </add>

  <add type-mapping=
     "/atg/epub/file/typemappers/WWWTextFileAssetTypeMapping">
     /users/joe/import/myHtmlFiles/page1.html
  </add>

  <add type-mapping=
     "/atg/epub/file/typemappers/WWWTextFileAssetTypeMapping">
     /users/joe/import/myHtmlFiles/page2.html
  </add>
</manifest>
```

4. On the asset management server, configure the `/atg/epub/file/VersionedLoaderEventListener` as appropriate for the import operation:

   - For initial import of file assets, do not specify a workspace name. Instead, let the system generate a workspace and workspace name. Using a system-generated workspace is appropriate for initial imports, because the workspace and files should be checked in immediately.
   - Set the property `VersionedLoaderEventListener.checkinOnCompletion` to true (the default) in order to check in the initial set of file assets. This is particularly important when using an system-generated workspace; if the files are imported into an system-generated workspace but not checked in, there is no way to access the assets via the ATG Business Control Center, because they were not imported into a known workspace—that is, a workspace associated with an existing project.

   For more information about VersionedLoaderEventListener component properties, see the *Repository Loader* chapter in the *ATG Repository Guide*.

5. Start an application that includes the `Publishing.base` module, or any module that requires `Publishing.base`. The `Publishing.base` module starts the RL (Repository Loader) module automatically.

6. Check that the `DYNAMO_HOME` environment variable is set on the content development or production server; then change to the `<ATG10dir>/RL/` directory and run the Repository Loader's RLClient. Specify the manifest file you created in step 3 as an argument.

For example:

```
bin/RLClient –h localhost –m /users/joe/temp/initialFileImport.xml
```

If the –h argument specifies a host other than `localhost`, the manifest file path supplied by the `-m` switch should be on the remote machine.

7. Because the VersionManager requires a forward slash (`/`) path separator, on Windows reset the following property to forward slash:

   `/atg/epub/file/PublishingFileRepository.pathSeparator`

exportRepository

`exportRepository` exports the contents of one or more standard repositories to a `.jar` data file. You typically use this tool together with `importRepository`, which imports the exported data into a standard or VersionRepository.

Note: When running `exportRepository` on a third-party application server, you must configure the server to use an ATG data source and transaction manager, not your native application server's data source and transaction manager.

Syntax

You run `exportRepository` from `<ATG10dir>/home/bin` as follows:

```
exportRepository [–m startup-modules] [-s server-name]
  –file output-file { -all | -r repository-list } [-batchSize size]
```

Command-Line Help

To obtain command-line help on syntax usage, type:

```
exportRepository -help
```

Required Arguments

Argument	Description
`-file` *`output-file`*	Specifies the target output file for the exported content. The file path can be absolute or relative to the current directory.
`-all`	Specifies to export the contents of all repositories registered in the repository registry service, located in Nucleus at `/atg/registry/ContentRepositories`. You must specify this option or `-r`.

`-r` *`repository-list`*	Specifies the absolute Nucleus component path of one or more source repositories, where *`repository-list`* is a list of comma-delimited repositories. You must specify this option or `-all`.

For example:

```
exportRepository –m Catalog –file /users/joe/CatalogExport.jar –r /myApp/Catalog
```

Optional Arguments

Argument	Description
`-m` *`startup-module`* `[-m` *`startup-module`*`]...`	Lists a module to start for the export process, which contains the source repositories of the data to export. Supply multiple `–m` options in order to start more than one module. This argument must precede all others, including `-file`.
`-s` *`server-name`*	The ATG instance on which to run this script. Use this argument when you have multiple servers running on a machine. If specified, this argument must precede all others except `-m`.
`-batchSize` *`size`*	The number of items to query at one time. The larger the specified number, the faster the export but the greater the amount of memory required. The default is 1000. Specify -1 to export all items in a single batch. **Note:** If using Oracle, avoid setting `-batchSize` to -1 unless the total number of items to export is less than 1000. Do not set–`batchSize` to an integer greater than 999.

Print Arguments

See Print Arguments under importRepository.

importRepository

`importRepository` imports the contents of a data file generated by `exportRepository`, into a standard or versioned repository. `importRepository` is especially useful for importing large numbers of items.

Note: When running `importRepository` on a third-party application server, you must configure the server to use an ATG data source and transaction manager, not your native application server's data source and transaction manager.

Syntax

You run `importRepository` from `<ATG10dir>/home/bin` as follows:

```
importRepository [–m startup-module]... [-s server-name] –file source-file
          {project-spec | workspace-spec}
          [optional-arguments]
```

For example:

```
importRepository –m CatalogVer –file /users/joe/CatalogExport.jar
–project MyFirstProject
```

Importing to a Project or Workspace

If importing to a versioned repository, you must specify either a project or a workspace as follows:

- `-project` *`name`* `[-workflow` *`name`*`] -noDeleteAll –username` *`name`*
- `-workspace` *`name`* `[-nocheckin] [-noDeleteAll]`

Note: After deployment targets are initialized, use `–project` with `importRepository` instead of `-workspace`. When `–project` is used, assets are imported into a new project with the default or specified workflow. Users can then access this project and perform the tasks associated with its workflow.

Command-Line Help

To obtain command-line help on syntax usage, type:

```
importRepository -help
```

Versioning Arguments

The following options used to import repository data to a versioned repository. In order to use them, the Publishing module must be running.

Argument	Description

`-project` *name*
`-noDeleteAll`
`[ -workflow` *name*`]`
	Specifies the name of the project to create for the import operation. This option is available only if the Publishing module is running. You must qualify this option with `-noDeleteAll`.
	
	After running `importRepository` with this argument, the imported assets must be checked in manually through the ATG Business Control Center.
	
	If qualified by `-workflow`, the project uses the specified workflow; otherwise, it uses the default workflow:
	
	`/Common/commonWorkflow.wdl`

`-workspace` *name*
`[ -nocheckin]`
`[ -noDeleteAll]`
	Specifies the workspace to use during the import operation, where *name* is a user-defined string with no embedded spaces and is unique among all workspace names. Use `-workspace` only during the initial import to the target repository, before you initialize any target sites.
	
	The workspace is the area in the VersionManager where the import takes place. If the specified workspace does not exist, the system creates it.
	
	You must specify this option or `-project`.
	
	If qualified by `-nocheckin`, the import does not check in imported data. This allows use of the workspace for multiple import operations, so all assets can be checked in at the same time.
	
	If qualified by `-noDeleteAll`, the imported data is added to items that already exist in the workspace. If omitted, all items in the workspace and target repositories are deleted before the import operation begins.

`-checkinComment` *comment*	Comment to use when checking in imported data. The default is `import`.
`-username` *name*	Username to use when checking in imported data. The default is `importRepository`. This argument is required when the `project` argument is supplied, so the user can be identified as the project creator.
`-versionManager` *path*	Component path of VersionManager to use for versioned imports. Use this argument only if the VersionManager runs in a non-standard location.

General Arguments

Argument	Description
`-m` *`startup-module`* `[-m` *`startup-module`*`]...`	Lists the modules to start for the export process. Specify the modules that contain the source repositories for exported data. To start multiple modules, you can supply multiple `–m` options, or delimit multiple modules with a semi-colon (;) on Windows and a colon (:) on UNIX. This argument must precede all others, including `-file`.
`-s` *`server-name`*	The ATG instance on which to run this script. Use this argument when you have multiple servers running on your machine. This argument must precede all others except `-m`.
`–file` *`source-file`*	Required, specifies the file with the data to import. The path that you specify can be absolute or relative to the current directory.
`-batchSize` *`size`*	The number of items to commit in a transactional batch. The larger the specified number, the faster the import. However, a large number requires more memory and a larger transaction log in the database. The default is 1000. Specify -1 to import all items in a single batch.
`-temp` *`directory-name`*	The temporary directory to use in order to expand the data file during the import process. Typically, this argument is not needed. By default, an appropriate OS-specific temporary directory is used. The default directory is the one used by `java.io.File.createTempFile(String, null, null)`, which should be appropriate for most systems.
`-noworkspace`	Specifies that the import should not use the versioning system. Use this argument to import data into unversioned repositories.

Print Arguments

`exportRepository` and `importRepository` use a compressed, binary file format. However, both utilities provide arguments to control message detail, and generate information about exported and imported data.

Utility Message

Argument	Description
`-v`	Show more detail in the message. Specify multiple times in order to increase the level of detail.
`-showTime`	print the current time in milliseconds with each info message

Data File

The following arguments print the data in a file:

Argument	Description
`-print`	Prints a summary of the data file and exits.
`-printItems [`*`range`*`]`	Prints the IDs of data file items.
`-printItemDetail [`*`range`*`]`	Prints detailed information about data file items, including item property values.
`range`	Constrains output to a specified range with the following arguments: `-printStartIndex` *`index`* where *`index`* is the zero-based starting index. `-printEndIndex` *`index`* where *`index`* is the zero-based ending index. Set *`index`* to -1 in order to print to the last item.

6 Managing User Access and Security

This chapter provides technical information on managing the security of ATG Content Administration and the assets that you manage with it. It includes the following sections:

- ATG Content Administration Users describes the ATG Content Administration users who manage assets via the ATG Business Control Center and ATG Control Center. Also describes the principals that are provided by default.
- Project and Workflow Security describes the default security settings for the installed project workflows.
- Access to Generic Activities describes how to control access to the Business Control Center Operations list.
- To Do List describes how to configure the list of projects that that are accessible to users who are logged in to the ATG Business Control Center.
- PublishingRepository Security describes the default security of the `/atg/epub/PublishingRepository`, the standard repository that stores the items required by ATG Content Administration, such as processes and projects.
- VersionRepository Security describes how to manage the security of the versioned repositories that store your assets.
- PublishingFileRepository Security describes the default security of the PublishingFileRepository, the content VersionRepository that is provided out-of-the-box for file assets.
- Disabling a Secured Repository describes a simple process for turning off security for a secured repository. Useful if you do not require security for the out-of-the-box repositories provided with ATG Content Administration.

Note: This chapter assumes you are familiar with the concepts discussed in the following chapters:

- *Secured Repositories* in the *ATG Repository Guide*.
- *Managing Access Control* in the *ATG Programming Guide*.

For information on managing the security of ATG Content Administration-managed assets on your deployment targets, see Manage Asset Security on Target Sites in the chapter *Setting Up Deployment*.

ATG Content Administration Users

ATG Content Administration users can be grouped into two categories according to the interface with which they manage both application assets and ATG Content Administration itself:

- ATG Business Control Center users
- ATG Control Center users

ATG Business Control Center Users

Each ATG Content Administration user requires a profile in the ATG profile repository; the values of the `login` and `password` profile properties are used as the username and password to log into the ATG Business Control Center. Access to activities within the ATG Business Control Center, however, is controlled by ACC roles (People and Organizations > Roles in the ACC).

The following table lists the preconfigured roles that ATG Content Administration provides for use in the ATG Business Control Center. For detailed, information, see PublishingRepository Security in this chapter.

Role	Intended for...
EPub-User	Users who create and manage assets.
EPub-Manager	Users who perform activities such as reviewing and approving content created by an EPub-User. An EPub-Manager can also deploy assets to production targets.
EPub-Admin	Users who require administrative privileges—for example, to configure the ATG Business Control Center or modify user access rights.
EPub-Super-Admin	Users who require full access to the PublishingRepository. The EPub-Super-Admin role is set as the role in the `superAdminRole` property of the `/atg/epub/Configuration` component. When checking the access rights to items in the PublishingRepository for a given user, ATG Content Administration first checks whether the user is assigned the role defined in `Configuration.superAdminRole`. If this is the case, the system assumes that the user has full access to the PublishingRepository, and no additional security checks are made. It's important to note that the role defined in `Configuration.superAdminRole` is automatically granted full access to all items in the PublishingRepository only. It is not automatically granted access to any other items, such as those stored in versioned repositories. To specify a different role as the Super Admin role, simply set the `/atg/epub/Configuration.superAdminRole` property to a fully qualified role name via the ATG Control Center.

In the initial stages of development, you should assign the EPub-User, EPub-Manager, or EPub-Admin role to any new ATG Business Control Center users that you create. However, you typically want to restrict access to various projects and assets to subsets of users, such as merchandisers, scenario authors, system administrators, and so on. Consequently, early in the development process, you should identify the user types that are required for your content development environment, create the appropriate principals (roles, organizations, and so on), and configure their access rights accordingly. You should complete this step early in the development process in order to minimize its difficulty and avoid runtime access problems. For more information on adding new principals, see the *ATG Personalization Programming Guide*.

Note: The ATG Content Administration user profile that is provided for default access to the ATG Business Control Center (username: `publishing`, password: `publishing`) is assigned the EPub-Super-Admin role. This user account exists only for evaluation and initial setup, and should be deleted as soon as you establish real user accounts. Only users who require full access to the PublishingRepository should use the default account or the EPub-Super-Admin role.

ATG Portal Roles

In addition to one of the ATG Content Administration roles described above, users also require an ATG Portal role that provides access to the ATG Business Control Center UI—for example `100001-member`. These roles are also assigned through the People and Organizations > Roles window in the ACC. The appropriate roles are located in the Global Roles > `Bizui` folder.

ATG Control Center Users

ATG Content Administration users access the ACC to manage versioned scenario and personalization assets, workflows, and occasionally other PublishingRepository items, such as projects. To do so, they need to be set up as Control Center Users (People and Organizations > Control Center Users) and assigned to specific ACC groups (People and Organizations > Control Center Groups). To manage ATG Content Administration items, the groups need access to the following areas in the ACC:

- Scenarios task areas
- Targeting task areas
- Publishing > Workflows task area
- Publishing > Publishing task area

By default, the `administrators-group` that is provided with the ATG platform has access to the areas shown above. See PublishingRepository Security in this chapter for information on the access rights of the administrators-group to the PublishingRepository.

For more information on setting up ACC users and groups, refer to the *ATG Personalization Programming Guide*.

Project and Workflow Security

This section discusses how to configure security for projects and workflows used in ATG Content Administration. It relies on familiarity with general workflow security, which uses the Access Control List mechanism. Workflow security is described in *Setting Up Security Access for Workflows* in the *ATG Personalization Programming Guide*.

User access to a project and its tasks in the ATG Business Control Center is controlled by the access settings for the project workflow. For example, access to a project's Author task options depends on having Execute access rights to that task in the underlying workflow.

Workflow access rights are themselves determined by roles. For example, in any project that uses unmodified the installed project workflow, write and execute access to the Content Review task is given to roles EPub-Manager, EPub-Super-Admin, and managers-group. Any user who has one of these roles can complete this task.

Write access lets a user change task attributes, such as its priority, owner, and access control list. Execute access lets a user complete or release a task.

The following table shows the access rights required to perform project and workflow-related tasks in the ATG Business Control Center:

Task	Required access
Create a project	Execute access to the project workflow
Add an asset to or remove an asset from a project	Execute access to Author task, appropriate access rights to the asset repository
Assign tasks to other users	Write access to the task
Release task	Execute access to the task
Complete a task—that is, change its status in the ATG Business Control Center	Execute access to the task
Deploy project	Execute access to the workflow Deploy task

Default Workflow Access Settings

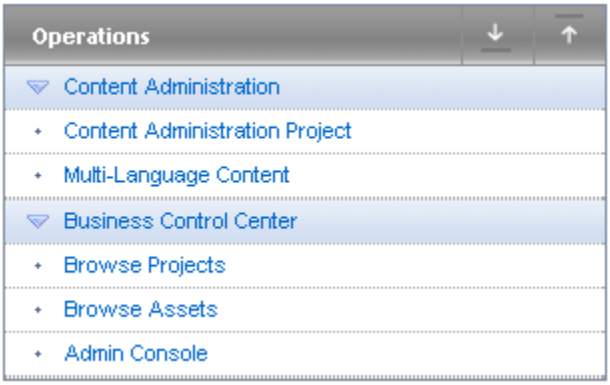
The following table describes the access rights that are initially set for the staging/production workflow:

Task	Roles	Access rights
Create project	administrators-group EPub-Admin EPub-Manager EPub-Super-Admin EPub-User managers-group	Execute
Author	All	Write/Execute
Content review	EPub-Manager EPub-Super-Admin managers-group	Write/Execute
Approve for staging deployment	EPub-Manager EPub-Super-Admin managers-group	Write/Execute
Wait for staging deployment completion	administrators-group EPub-Admin EPub-Super-Admin	Write/Execute
Verify staging deployment	EPub-Manager EPub-Super-Admin EPub-User managers-group	Write/Execute
Approve for production deployment	EPub-Manager EPub-Super-Admin managers-group	Write/Execute
Wait for production deployment completion	administrators-group EPub-Admin EPub-Super-Admin	Write/Execute
Verify production deployment	EPub-Manager EPub-Super-Admin EPub-User managers-group	Write/Execute

To change access rights for a workflow or its individual tasks, open the workflow in the ACC and edit the appropriate elements. For more information, see *Setting Up Security Access for Workflows* in the *ATG Personalization Programming Guide*.

Access to Generic Activities

The Operations list on the Home page of the ATG Business Control Center contains links that are used to access different functional areas within the UI. The following image shows this list with some sample data:



The Business Control Center options are classified as generic activities, and access is controlled by the Activity Manager's `PublishingActivities.xml`file, which is located in the application's configuration path. The file contains `<acl>` tags for each ATG Content Administration activity, which specify user directory roles and their access rights. The following example shows the default entry for the Admin Console option, which provides access to deployment management:

```
<generic-activities>

  <activity>
    <id>adminConsole</id>
    <resource-bundle>atg.epub.activity.ActivityResources</resource-bundle>
    <display-name-resource>adminDisplayName</display-name-resource>
    <description-resource>adminDescription</description-resource>
    <destination-page>
      <url>/atg/atgadmin</url>
      <acl>Profile$role$epubAdmin:read;Profile$role$epubSuperAdmin:read</acl>
    </destination-page>
  </activity>

</generic-activities>
```

To see the entire contents of `PublishingActivities.xml` and additional configuration pieces, look in:

```
<ATG10dir>/Publishing/base/config/atg/bizui/activity
```

The roles shown correspond to the EPublishing global roles shown in the ATG Business Control Center. Add or remove roles as needed to give or revoke access to a generic activity. (Edit a copy of `PublishingActivities.xml` and add it to your `localconfig` layer—see XML File Combination in the *ATG Programming Guide*.) For detailed information on access control lists (ACLs), refer to *User Directory Security* in the *ATG Personalization Programming Guide*.

The `PublishingActivities.xml` file is checked at intervals for modifications, so you can make changes to it without needing to restart the server. The interval is set to 5 minutes by default and is defined by the

genericActivityFileModificationInterval property in the /atg/bizui/activity/PublishingActivitySource.properties component.

Note: If no items at all appear in the Operations list, the session might have expired. Start a new session by logging out of the ATG Business Control Center and logging back in again.

To Do List

The Home page of the ATG Business Control Center displays to logged-in users a To Do List that contains projects that they can access.

The To Do List is configurable through the /atg/bizui/Configuration component, by editing the processDisplayStatus property list. You can set this property to one or more of the following values:

Value	Description
Edit	The project is open for editing. The display name that appears for this status—for example, in the Available Projects page—is Active Project.
Completed	The project is deployed and cannot be modified.
Deployed*	The project is deployed to the target site.
Running*	The Outreach campaign is deployed and running on the target site.
EditRunning*	The Outreach campaign is deployed and running on the target site, and is open for modification.

* Valid only for ATG Outreach

For example:

```
processDisplayStatus=Edit
```

By default, the To Do list contains projects with the following status:

- Edit
- Deployed (ATG Outreach only)
- EditRunning (ATG Outreach only)

PublishingRepository Security

The /atg/epub/PublishingRepository is the standard unversioned repository that stores all items required by ATG Content Administration—by default, processes and projects.

ATG Content Administration is configured to access the PublishingRepository through the component `/atg/epub/SecuredPublishingRepository`, which overlays the PublishingRepository and instantiates this class:

```
atg.adapter.secure.GenericSecuredMutableContentRepository
```

The SecuredPublishingRepository uses a custom security policy that is implemented by the component `/atg/dynamo/security/PublishingSecurityPolicy`. This policy grants all users who are assigned the role specified in `/atg/epub/Configuration.superAdminRole` full access to the PublishingRepository.

The following table summarizes the security defined for the PublishingRepository item descriptors. You can also refer to the secured repository definition file, which is located at:

```
<ATG10dir>/Publishing/base/config/atg/epub/publishingSecurity.xml
```

Note that an item descriptor's `descriptor-acl` defines the ACL for the item descriptor. Its `creation-base-acl` defines the default ACL for a new item of the item descriptor type.

For an explanation of each access right specified below, see the *Secured Repositories* chapter in the *ATG Repository Guide*.

Item descriptor	descriptor-acl	creation-base-acl
`process`	EPub-Admin: read, write, create, delete EPub-Manager: read, write, create, delete EPub-User: read, write, create, delete ACC administrators-group: read, write, create, delete	undefined
`project`	EPub-Admin role: read, write, create, delete EPub-Manager role: read, write, create, delete EPub-User role: read, write, create, delete ACC administrators-group: read, write, create, delete	undefined

The `publishingSecurity.xml` file also defines the `acl-property` for the process and project item descriptors. The name of this property is `acl`.

You should modify item descriptor-level security of the PublishingRepository during development because it requires that you manually modify `publishingSecurity.xml` and restart the application that includes ATG Content Administration for the changes to take effect.

VersionRepository Security

If you configure secured repositories to sit on top of your versioned repositories, you can set item descriptor-level and, if necessary, item-level security as described in the *Secured Repositories* chapter in the *ATG Repository Guide*.

ATG Content Administration accesses a secured VersionRepository through the secured repository adapter component that overlays it. This component is an instance of the class :

```
atg.adapter.secure.GenericSecuredMutableVersionRepository
```

Configuring

To configure the security of a secured versioned repository, follow these steps:

1. Define the ACLs for the item descriptors as needed by modifying the secured repository definition file for the underlying repository. This lets you define access rights to all assets of a given type. For more information, see the *Secured Repositories* chapter in the *ATG Repository Guide*.
2. If necessary, also define the ACLs for individual items.

As you define the ACLs for the item descriptors and items in your versioned repositories, make sure that you coordinate them with the ACLs that you define for the projects and workflows that might use the items.

PublishingFileRepository Security

The versioned content repository `/atg/epub/file/PublishingFileRepository` stores an application's file asset metadata. ATG Content Administration accesses the PublishingFileRepository through the SecuredPublishingFileRepository component that overlays it:

```
/atg/epub/file/SecuredPublishingFileRepository
```

This component is an instance of the class :

```
atg.adapter.secure.GenericSecuredMutableVersionContentRepository
```

The default SecuredPublishingFileRepository component is set as follows:

```
$class=atg.adapter.secure.GenericSecuredMutableVersionContentRepository

repositoryName=PublishingFiles
repository=/atg/epub/file/PublishingFileRepository
configurationFile=/atg/epub/file/publishingFileSecurity.xml
securityConfiguration=/atg/dynamo/security/PublishingFileSecurityConfiguration
```

```
XMLToolsFactory=/atg/dynamo/service/xml/XMLToolsFactory
transactionManager=/atg/dynamo/transaction/TransactionManager
```

Item Descriptor Security

By default, security for all PublishingFileRepository item descriptors is defined as follows:

Principal	Access Privileges

ATG Content Administration roles:
`EPub-Super-Admin`
`EPub-Admin`
`EPub-Manager`
`EPub-User`
 	
ACC groups:
`administrators-group`
	Read
	Write
	Create
	Delete

ACC groups: `everyone-group`	Read

You can examine the secured repository definition file at this location:

```
<ATG10dir>/Publishing/base/config/atg/epub/file/publishingFileSecurity.xml
```

You can also access this file in the ATG Dynamo Server Admin Component Browser, via the `configurationFile` property of the SecuredPublishingFileRepository component.

Content Item Security

The SecuredPublishingFileRepository uses a custom security policy that determines user access to a content item as follows

1. Checks the ACL for the given item.
2. If the item's `acl` property is null or empty, checks the ACL for its parent folder:
   - If set, the parent folder's ACL is used to determine user access to the child item.
   - If null or empty, the system walks up the folder hierarchy until a folder with a defined ACL is found.

By default, ATG Content Administration defines an ACL for the repository's root folder; the ACL is defined as follows:

Principal	Access Privileges

ATG Content Administration roles:
`EPub-Super-Admin`
`EPub-Admin`
`EPub-Manager`
ACC groups:
`administrators-group`
	List
	Read
	Writ
	Destroy
	Read_ACL
	Write_ACL

ATG Content Administration roles:
`EPub-User`
ACC groups:
`everyone-group`
	List
	Read

Configuring the security of the PublishingFileRepository is similar to configuring security for any other VersionRepository that stores your application's assets. You can configure the following assets:

- Scenario and personalization assets
- Web assets

Scenario and Personalization Assets

If you manage scenario and personalization assets in the content development environment, users create and edit them in two interfaces:

- ATG Business Control Center: Users create, modify and delete personalization assets, and they can complete projects that contain both scenario and personalization assets. They can also use this interface to delete scenario assets.
- ATG Control Center (ACC): Users create, modify and delete scenario assets. It is also possible to manage Personalization assets in the ACC, but it is recommended to do so in the ATG Business Control Center.

Users who work with these assets require user accounts for both the ATG Business Control Center and the ACC; access privileges for the two accounts must be coordinated.

However, note the following as you configure the corresponding access privileges for the ACC:

- You can disable and enable access to the ACC Scenarios task area as needed. See the discussion on managing access control in the *ATG Programming Guide* for details.
- As described in the *ATG Personalization Programming Guide*, the ACC has its own default security policy and security mechanisms for the Scenarios task area. By default the administrators-group is granted full access to all scenarios, while the everyone-group is granted List and Read access only. This has the following implications for ATG Content Administration-managed assets:
  - If you need to restrict access to the Scenarios task areas to a subset of users, you should assign those users as members of the administrators-group. Then create

one or more custom groups for users who require access to other administrative ACC task areas.

- Carefully coordinate the security set on the scenario folders and scenarios via the ACC security features with the security used to give access to the same items in the ATG Business Control Center.

- For information on using the ACC's security features for scenario folders and scenarios, see the discussion on scenario security in the *ATG Personalization Programming Guide*.

Web Assets

If you intend to use the PublishingFileRepository to store Web file asset data (items of type `textFileAsset` and `binaryFileAsset`), you can configure the security of these assets using the same general process as for items in any other VersionRepository. Simply keep in mind the custom security policy and folder hierarchy of the PublishingFileRepository as you do so.

Disabling a Secured Repository

This section describes a simple process for disabling a secured repository. This process is useful if you do not require security for the out-of-the-box repositories provided with ATG Content Administration, such as the PublishingFileRepository, but you do not want to completely remove the default configuration.

To disable a secured repository:

1. In your `localconfig` directory, layer on a configuration file for the secured repository that looks like this:

   ```
   $class=atg.nucleus.GenericReference
   componentPath=path
   ```

   where *path* is the Nucleus path of the underlying repository

2. Reconfigure the VersionManagerService to manage the unsecured repository, not the secured one. See Configure the VersionManagerService for more information.

   Note: You cannot layer on a configuration file to make this change; modify the configuration file that is provided by default.

Checking Versioned Repostory Security

You can check whether a given versioned repository—atg.adapter.secure.GenericSecuredMutableVersionRepository and atg.adapter.secure.GenericSecuredMutableVersionContentRepository—is

7 Setting Up an ATG Content Administration Cluster

Clustering multiple asset management servers can help load-balance the use of resources for authoring and editing. In order to set up an ATG Content Administration cluster, you perform these tasks:

- Install cluster servers.
- Configure cluster servers.
- Manage distributed file assets.
- Configure deployment from a cluster.

Install Cluster Servers

You can install multiple asset management servers that run against a single installation of ATG. All servers run against the same ATG Content Administration database.

Caution: Never use the same ATG installation for more than one cluster.

To create a cluster of asset management servers:

1. Perform the initial installation and set up of ATG, including ATG Content Administration. Configure the initial asset management server as described in Setting Up an Asset Management Server.
2. Use the ATG Dynamo Server Admin to create additional instances of asset management servers. The process is described in detail in "Installing Multiple Dynamo Servers" in the *ATG Installation and Configuration Guide*. By default each additional server inherits the settings of the initial server.
3. Make sure that all servers point to the same database.
4. Follow the remaining multiple server configuration steps in this chapter.

Configure Cluster Servers

The process of configuring multiple asset management servers includes the following tasks:

1. Identify the workflow editor server.
2. Configure distributed caching for versioned repositories.

Identify the Workflow Editor Server

In a cluster, you must identify the workflow editor server in the Workflow Process Manager's XML configuration file, `workflowProcessManager.xml`. By default this file is located in:

```
<ATG10dir>/Publishing/base/config/atg/epub/workflow/process/
```

To override the default server configuration, create a file with the same name in the top-level directory—that is in the `/home/localconfig` directory that applies to the whole cluster:

```
<ATG10dir>/home/localconfig/atg/epub/workflow/process/
```

The following example shows the workflow editor server setting in this file:

```
<?xml version="1.0" encoding="UTF-8" ?>
<process-manager-configuration>

<process-editor-server>
  <server-name>dyn1:8850</server-name>
</process-editor-server>

</process-manager-configuration>
```

For detailed information on determining the server name and port number, refer to *Configuring the Scenario Manager* in the *ATG Personalization Programming Guide*.

The same server can manage scenarios and workflows. To use the same server for both, specify the same `<server-name>` setting in `scenarioManager.xml` and `workflowProcessManager.xml`. For more information, refer to *Configuring the Scenario Manager* in the *ATG Personalization Programming Guide*.

Configure Distributed Caching for Versioned Repositories

In order to maintain consistency across all versioned repositories that are distributed among cluster servers, you should configure repository items to use distributed caching. Set the `cache-mode` property for item descriptors to a distributed caching mode, one of the following:

- `distributed` (distributed TCP caching)
- `distributedJMS` (distributed JMS caching)

- `distributedHybrid` (distributed hybrid caching)

For more information about distributed caching options, see the *SQL Repository Caching* chapter in the *ATG Repository Guide*.

Manage Distributed File Assets

The servers that comprise an ATG Content Administration cluster can either run on the same host machine, or be distributed across different hosts.

If all servers run on the same host, you can set the location of file asset storage for all servers to a common location as follows:

- Create a `publishingFiles.xml` file with a new `pathPrefix` attribute that overrides the default.
- Put the file in your application's configuration path.

For detailed information, see Changing File Asset Storage Location.

If the cluster servers are distributed across different host machines, you must coordinate file asset storage and access. You can do so in several ways:

- Use a storage area network (SAN) and make the location of the `versionFileStore` exist on the SAN.
- Use `rsync` to synchronize all directories on each server.
- Use a network file system (NFS) and make the location of the versionFileStore exist on the SAN.
- If none of the aforementioned methods are available, synchronize file assets with ATG's `FileSynchronizationDeployServer` component, as described in the next section.

Synchronizing Distributed Files with FileSynchronizationDeployServer

If enabled, the `FileSynchronizationDeployServer` component synchronizes distributed files whenever a given file is required by any server on the cluster. In order to use this component, you must set properties on it and on `/atg/epub/Configuration`, as follows:

`/atg/epub/file/synchronization/FileSynchronizationDeployServer`:

Property	Setting
`enabled`	`true`
`port`	The desired listener port number, set by reference to `/atg/dynamo/Configuration.fileSynchronizationDeploymentPort`. To change the port setting, change the referenced property.

/atg/epub/Configuration:

Property	Setting
remoteHosts	In a comma-delimited list, specify the host names of other servers in this server's cluster.
remoteRMIPorts	In a comma-delimited list, specify the RMI port settings that are configured for the hosts specified in remoteHosts. List the ports in the same order as the corresponding hosts.
remotePorts	In a comma-delimited list, specify the file synchronization ports that are configured for the other servers in their FileSynchronizationDeployServer components. List the ports in the same order as the corresponding hosts in remoteHosts.

For example:

```
remoteHosts=\
   jupiter.acme-widgets.com,\
   saturn.acme-widgets.com,\
   uranus.acme-widgets.com

remoteRMIPorts=\
   8860,\
   8860,\
   8860

remotePorts=\
   8815,\
   8815,\
   8815
```

Configure Deployment from a Cluster

In order to set up deployment from an ATG Content Administration cluster:

- For each cluster server, complete the deployment procedures described in Setting Up Deployment.
- For each cluster server, set its serverName and drpPort properties in /atg/dynamo/service/ServerName.properties to values that are unique within the cluster.

For example, you might set server jupiter as follows:

```
serverName=jupiter:8850
drpPort=8850
```

Then you can set server saturn as follows:

```
serverName=saturn:18850
drpPort=18850
```

- Set up a ServerLockManager and ClientLockManagers for the asset management server cluster. A ClientLockManager must be set up on each ATG server in the cluster. For information about configuring a ServerLockManager and ClientLockManagers, refer to the SQL Repository Caching chapter in the *ATG Repository Guide*.

  Note: Do not run any ATG Content Administration modules or DAF Deployment on a standalone ServerLockManager. Doing so causes deployment deadlocks.

- In order to configure a cluster for switch deployment, set each server's remoteHosts, remoteRMIPorts, and remotePorts properties appropriately.

Configure a Cluster for Switch Deployment

In order to enable switch deployment from a multi-server cluster, each server must be configured with contact data about the other servers in the cluster. For each server in a cluster, configure these properties in `/atg/epub/Configuration`:

- `remoteHosts`: In a comma-delimited list, specify the host names of other servers in this server's cluster.
- `remoteRMIPorts`: In a comma-delimited list, specify the RMI port settings that are configured for the hosts specified in `remoteHosts`. List the ports in the same order as the corresponding hosts.
- `remotePorts`: Set this property only if you use the `FileSynchronizationDeployServer` to synchronize distributed file system assets (see Manage Distributed File Assets).

  In a comma-delimited list, specify the file synchronization ports that are configured for the other servers in their `FileSynchronizationDeployServer` components. List the ports in the same order as the corresponding hosts in `remoteHosts`.

For example, server `pluto.acme-widgets.com` might have the following settings for other servers in the same cluster:

```
remoteHosts=\
   jupiter.acme-widgets.com,\
   saturn.acme-widgets.com,\
   uranus.acme-widgets.com

remoteRMIPorts=\
   8860,\
   8860,\
   8860
```

8 Project Workflows

ATG Content Administration uses workflows to define the tasks that constitute a project's lifecycle, and to manage the project's progress from one task to another. Workflows typically let users perform these tasks:

- Author or revise project asset content
- Review content changes
- Approve the project for deployment
- Verify deployment

You can customize workflows to suit your business requirements. For example, the default workflow provides only one content review task; if desired, you can modify the workflow so it provides multiple content review cycles by different reviewers.

A project's workflow defines deployment targets; it also specifies whether assets are deployed directly to production targets, or are initially deployed to staging targets where they can be evaluated before final deployment to production. You decide which deployment model to use when you assemble the asset management server's Web application (see Installed Workflows).

Note: All project workflows must use the same deployment model: either production-only or staging/production.

Recommended Background Reading

This chapter describes the workflows that are provided with ATG Content Administration, and explains how you can modify them. It assumes you are familiar with the following documentation:

- *Using Workflows* in the *ATG Personalization Guide for Business Users* describes how to create workflows with the ATG Control Center.
- *Creating and Configuring Workflows* in the *ATG Personalization Programming Guide* explains how to configure backend support for workflows and how to extend workflow features through the workflow API.

Chapter Contents

This chapter contains the following sections:

- Workflow Action Elements

Installed Workflows

The ATG installation provides two workflows, which are described in this chapter:

- Production-only (default) supports deployment to a single target.
- Staging/production supports deployment to two targets: first a staging target, then a production target.

Only one of these is available, depending on how you assemble the asset management server application. In order to use the staging/production workflow, the EAR file that you deploy on the asset management server must be assembled with the `-layer staging` option.

Workflow Target Sites

The workflows provided by the ATG installation initially define one or two target sites with the following identifiers:

- `Production` is the sole target defined in the production-only workflow.
- `Staging` and `Production` targets are defined in the staging/production workflow.

In order to use a workflow with the predefined target site identifiers, you must define the deployment topology with site names that correspond exactly to these identifiers. For more information on defining target sites, see Define the Deployment Topology.

Production-Only

Unmodified, the production-only workflow deploys to a single production target and contains the following sequence of task elements:

1. Author
2. Content review
3. Approve for production deployment
4. Wait for production deployment completion
5. Verify production deployment

The following sections describe these elements and the sequence of elements associated with them, as set up in the ATG installation. For information about individual workflow elements, see Workflow Action Elements.

Author

Options	Action
Ready for review	Lock project assets from further edits; check whether the head versions of project assets match the current head versions on the main branch. Advance to the next task, Content Review.
Delete Project	See action element Delete Project.

Content Review

Options	Action
Approve Content	Advance workflow to next task, Approve for production deployment.
Reject	Reopen project; return workflow to Author.
Delete Project	See action element Delete Project.

Approve for Production Deployment

Options	Action
Approve and Deploy to Production	Approve and deploy project: advance workflow to next task, Wait for production deployment completion.
Approve for Production Deployment	Approve project, advance workflow to next task, Wait for production deployment completion.
Reject	Release asset locks; return workflow to Author task.

Wait for Production Deployment Completion

Two outcomes are possible:

- Deployment succeeds: advance workflow to next task Verify production deployment.
- Deployment fails: return workflow to previous task Approve for production deployment.

Verify Production Deployment

Options	Action
Accept Production Deployment	Validate project deployed on target, and take the following actions: 1. Check in project's workspace. 2. Complete project. 3. Complete process.
Revert Assets on Production Immediately	Revert assets Immediately on target. If revert operation is successful: release asset locks, reopen project, and return to Author. If revert operation fails: return to Verify production deployment.

Staging/Production

The unmodified staging/production workflow performs these tasks:

1. Author
2. Content review
3. Approve for staging deployment
4. Wait for staging deployment completion
5. Verify staging deployment
6. Approve for production deployment
7. Wait for production deployment completion
8. Verify production deployment

The following sections describe these tasks and the sequence of actions and events that follow them, as set up in the ATG installation. For information about individual workflow elements, see Workflow Action Elements.

Author

Options	Action
Ready for review	Lock project assets from further edits; check whether the head versions of project assets match the head versions on the main branch. Advance to the next task element.
Delete Project	See action element Delete Project.

Content Review

Options	Action
Approve Content	Advance workflow to next task, Approve for staging deployment.
Reject	Reopen project; return workflow to Author.
Delete Project	See action element Delete Project.

Approve for Staging Deployment

Options	Action
Approve and Deploy to Staging	Approve and deploy project: advance workflow to next task, Wait for staging deployment completion.
Approve for Staging Deployment	Approve project, advance workflow to next task, Wait for staging deployment completion.
Reject	Release asset locks; return workflow to Author task.

Wait for Staging Deployment Completion

Two outcomes are possible:

- Deployment succeeds; advance workflow to next task Verify staging deployment.
- Deployment fails: return workflow to previous task Approve for staging deployment.

Verify Staging Deployment

Options	Action
Accept Staging Deployment	Validate project deployed on target, and advance workflow to next task, Approve for production deployment.
Revert Assets on Staging Immediately	Revert assets Immediately on target. If revert operation is successful: release asset locks, reopen project, and return to Author. If revert operation fails: return to Verify staging deployment.

Approve for Production Deployment

Options	Action
Approve and Deploy to Production	Approve and deploy project: advance workflow to next task, Wait for production deployment completion.
Approve for Production Deployment	Approve project, advance workflow to next task, Wait for production deployment completion.
Reject	Release asset locks; return workflow to Author task.

Wait for Production Deployment Completion

Two outcomes are possible:

- Deployment succeeds; advance workflow to next task Verify production deployment.
- Deployment fails: return workflow to previous task Approve for production deployment.

Verify Production Deployment

Options	Action
Accept Production Deployment	Validate project deployed on target, and take the following actions: 1. Check in project's workspace. 2. Complete project. 3. Complete process.
Revert Assets on Production Immediately	Revert assets Immediately on Target. If revert operation is successful, return to Verify staging deployment. If revert operation fails, repeat Verify production deployment.

Asset Locking and Check-in

Workflow deployment elements trigger project asset locking as follows:

- If the project workflow includes staging and production targets, project assets are locked when a user approves deployment to staging.
- If the project workflow includes only a production target, assets are locked when a user approves deployment to production.

In both cases, assets remain locked until deployment to production is complete and verified. At that point, the workflow checks in project assets and unlocks the assets. Until then, other projects cannot deploy those assets. You can determine which assets are locked by other projects by opening the project's Lock Conflicts tab.

If projects routinely contain large numbers of assets, you can optimize asset locking in order to advance the project workflow more quickly. See Optimizing Workflow Performance in the Setting Up an Asset Management Server chapter.

Note: Project assets can be deployed to one-off targets at any stage of the project's workflow, whether locked or unlocked. For more on one-off targets, see One-Off Deployments in the Deployment Concepts chapter.

Creating Project Workflows

You are likely to supplement the installed ATG project workflow to suit your special requirements. You should create a workflow by copying the installed workflow and use that as your starting point. In general, you should adhere as closely as possible to the original workflow, especially with respect to elements that handle deployment. Creating project workflows from scratch is not recommended.

Before you make structural changes to an existing workflow, make sure it is not in use by open projects; doing so invalidates those projects and make it impossible to deploy their assets. Structural changes include adding or deleting elements. You can safely make cosmetic changes to a workflow such as changing its display name, and changing security access rights to a workflow and its elements.

Phasing Out a Workflow

When you introduce a new workflow to replace another one, you can safely phase out the old workflow by simply removing access rights to it. In this way, only the new workflow is accessible, and the deprecated workflow is phased out as soon as all current projects that use it are complete.

Procedure

Edit a workflow in the following steps:

1. In the ACC, locate the project workflow to use. Currently installed workflows are located in:

   Workflow > Publishing > ATG Content Administration
2. Right-click the project workflow name and choose Duplicate Workflow Definition.
3. Set the new workflow's location to the ATG Content Administration folder.
4. Give the duplicate workflow an appropriate name.
5. Click OK.
6. Open the new workflow for editing.
7. Assign a display name and description to the new workflow. The display name appears as an ATG Content Administration operation on the Home page of the ATG Business

Control Center. Users choose this operation in order to start a project that uses this workflow.

To edit the display name and description:

- Right-click the initial Workflow element.
- Choose Edit Details.

8. Change the workflow as desired. For detailed information on editing workflows, refer to the *ATG Personalization Guide for Business Users*. See also the next section, Workflow Action Elements.
9. If desired, set appropriate access rights on the workflow and individual task elements. Workflow access rights determine whether users can create a project with that workflow, and the tasks that they can manage.

   To set access rights for the workflow:

   - Right-click the initial Workflow element.
   - Choose Edit Details > Set Access Rights.

   To set access rights for a task element:

   - Right-click on the task element
   - Choose Edit Details > Set Access Rights.

   For detailed information about setting access rights to workflow and workflow elements, see Project and Workflow Security in the Managing User Access and Security chapter.
10. Save changes to the new workflow.

    Note: A workflow can be saved only if the deployment elements have defined targets.
11. Reassemble and redeploy the Web application containing ATG Content Administration. The next time you run the ATG Business Control Center, the new project workflow appears under the ATG Content Administration list of project types.

Workflow Action Elements

This section lists action elements that ATG Content Administration provides, including the default set of workflow elements. You add these elements to an ATG Content Administration workflow through the ACC workflow editor.

Workflow actions extend class `atg.epub.workflow.process.action.PublishingAction`. For general information on workflow architecture, see the *ATG Personalization Programming Guide*.

Approve and Deploy Project

Deploys to the specified target. This target must already be defined, as described later in this manual in Define the Deployment Topology.

Approve Project

Marks the project as approved for deployment on the specified target; deployment is triggered either by the RecurringDeploymentService or manually through the ATG Business Control Center Admin Console. The target must already be defined, as described later in this manual in Define the Deployment Topology.

Check Assets Are Up to Date

Compares the head version of project assets against the current head versions of those assets on the main branch. If these are not the same, the system generates an alert; the user must resolve version discrepancies before the workflow can be advanced.

For more on how ATG Content Administration versions assets, see Versioning Process earlier in this guide. For more information on asset conflict resolution, see the *ATG Content Administration Guide for Business Users*.

Check In Project's Workspace

Checks in all resources associated with the current project, thereby creating checked-in versions of assets from the working versions in the project.

Clone Project

Valid only for ATG Outreach, used exclusively by the Edit Asset process workflow as part of the internal procedure for managing projects: this element clones assets from the previous project associated with a process to the current project, creating working versions of those assets. You typically do not add this element to project workflows.

Complete Process

Used by a project workflow's parent process to indicate that the current process is complete.

Complete Project

Indicates that the current project is complete.

Create Process Data

Valid only for ATG Outreach, specifies process data type.

Create Project

Used by the workflow's parent process to start a project.

Delete Project

Terminates the workflow, and ends the project without checking in the workspace. Typically this action is available only before the project reaches the deployment stage.

Release Asset Locks

Called after a deployment is reverted from all targets. Before the workflow can return to a predeployment task such as Author, the project assets must be unlocked.

Reopen Project

After a successful revert from operation, reopens the project for further editing. Because this action enables users to modify project assets, it should always be followed by a Go To element that redirects the workflow to the Author task element.

Revert Assets Immediately on Target

Reverts deployment of project assets from the specified target.

Send Task Notification

Sends an email message to the recipients you specify – the owner of the current task, the permitted actors, the owner or permitted actors, or the last actor. In this case, Actors refers to anyone who has access rights to the task. This element is defined by the `emailNotifyTaskActors` entries in the file:

```
<ATG10dir>/Publishing/base/config/atg/epub/workflow/process/workflowProcessManager
.xml
```

Use the Send Notification action to send email messages outside the context of a task.

Validate Project Deployed on Target

Checks to ensure the workflow can be safely advanced to the next task. This element should be called after a project is deployed, to prevent the workflow from advancing to the next task before deployment is complete.

Note: ATG Content Administration does not support task priorities or dynamic routing of tasks.

9 Customizing Asset Display

The ATG Business Control Center uses the View Mapping system to display assets and asset properties. The View Mapping system is an object-based framework of dynamically generated pages, which are created at runtime according to the configuration in the View Mapping repository. Thus, you can customize asset and property display without hard-coding links or parameters in the pages themselves.

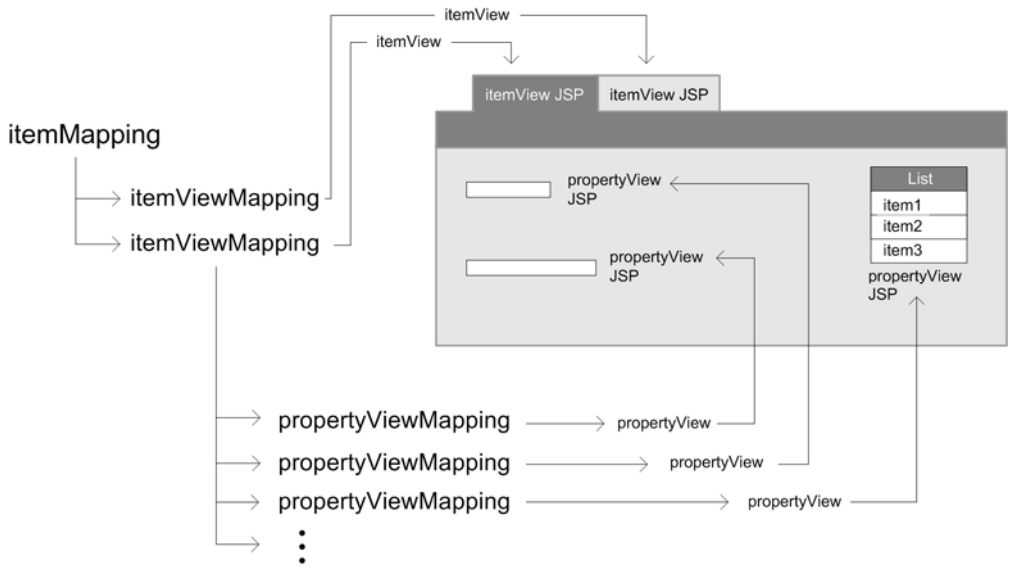
Note: The system and procedures described in this chapter apply only to portions of the ATG Business Control Center that are JSP-based. Interfaces that are Flex-based have different customization requirements. These are ATG Merchandising (with the exception of Price Lists and Commerce Search), Promotions, and Site Administration.

This chapter contains the following sections:

View Mapping System

The View Mapping system renders assets and their properties via itemMappings. Each itemMapping serves as a container within the View Mapping repository for the objects required to render an asset.

For example, the following graphic depicts how the View Mapping system might organize asset properties in two separate tabs:



The View Mapping system is three-tiered:

- The asset's itemMapping lists one or more itemViewMappings—two in this example.
- Each itemViewMapping points to an itemView, which in turn points to a JSP fragment that is used to generate a tab. Each itemViewMapping also lists one or more propertyViewMappings.
- Each propertyViewMapping points to a propertyView, which in turn points to a JSP fragment that is used to render the property.

When a user chooses an asset in the ATG Business Control Center, the following events occur:

1. The `getItemMapping` tag in the JSP queries the View Mapping system, passing it:
    - The user's current mode—for example, edit or view
    - The asset's item descriptor type
2. The View Mapping system returns the appropriate `itemMapping`, which serves as a container within the View Mapping repository for the objects needed to render an asset in a specific mode.
3. On obtaining the `itemMapping`, the system finds the appropriate underlying JSP fragments.

Note: In the context of the View Mapping system, *item* corresponds to *asset*.

View Mapping Attributes and Inheritance

View mapping objects can set attributes that are passed to the corresponding JSP fragment. These attributes are added to the appropriate object through the ACC as key/value pairs and are stored in the View Mapping repository. Because the attributes are stored in the repository rather than defined in each page, you can easily adapt the appearance of assets or properties and maintain a variety of display options that suit different users and contexts.

ATG Content Administration provides itemViews and propertyViews with predefined attributes whose values are applied when a particular view is displayed. For example, the propertyView `WYSIWYG HTML Editor` sets a number of variables that control the editor's appearance and behavior, such as `appletHeight`, `appletWidth`, `dictionary`, `inputFieldMaxlength`, `rows`, and `spellCheck`.

itemMapping attributes are passed to each itemViewMapping, whose attributes supersede its parent's settings. For example, an itemViewMapping can specify an itemView that overrides the default itemView used for that asset type. Each itemViewMapping also contains settings that define the appearance of the tab, such as its display name.

propertyViewMappings can provide property-level overrides. For example, if the property `assetX.prop1`, which is a `String`, requires a property editor different from the default simple `String` editor, the propertyViewMapping for `assetX.prop1` can specify a propertyView that overrides the default.

For more information on the attributes of each object, see View Mapping Repository.

itemView and propertyView JSP Fragments

The View Mapping system displays assets in the ATG Business Control Center by combining two types of JSP fragments:

- itemView JSP fragments
- propertyView JSP fragments

The View Mapping system determines how to combine these JSPs by examining the links in the View Mapping object hierarchy, the attribute values stored in the object hierarchy, and the item descriptor in the View Mapping repository definition file.

itemView JSP Fragments

An itemView object in the View Mapping system contains a pointer to an itemView JSP and identifies the itemView's mode, such as view or edit. An itemView appears in the ATG Business Control Center as a tabbed page.

propertyView JSP Fragments

propertyView JSP fragments control display of asset properties. Each property data type uses a different propertyView JSP. For example, a property of type `date` is displayed by default with a date picker JavaScript control; while a property of type `big string` is presented using the default HTML editor `propertyView WYSIWYG HTML Editor`. The HTML that displays each data type is generated from the corresponding propertyView JSP fragment.

For example, the default repository itemView for edit mode—`Standard RepositoryItem Editor`—uses the JSP `oneColumnEdit.jsp`, in this location:

```
<ATG10dir>/PubPortlet/PubPortlets.ear/portlets.war/html/views/item/gsa/
```

The oneColumnEdit.jsp page iterates through the properties specified in the asset's repository definition file. It generates a form field for each property by including the appropriate propertyView JSP fragment.

View Mapping Repository

The View Mapping repository is defined by the ViewMapping.xml file, in:

```
<ATG10dir>/BIZUI/config/config.jar
```

The repository stores the following items, which are defined by corresponding item descriptors in the repository definition file:

- itemMapping
- itemViewMapping
- itemView
- propertyViewMapping
- propertyView
- Map modes
- View Mapping Form Handlers

itemMapping

An itemMapping is a top-level container object where you configure the edit mode and display of an asset type. itemMappings provide default mapping information for general classes of asset types, such as repository items and virtual files.

Item Descriptor Definition

The itemMapping item descriptor is defined in ViewMapping.xml:

```
<item-descriptor name="itemMapping" id-space-names="viewmapping"
    default="true" display-property="description" query-cache-size="80">

    <table name="vmap_im" type="primary" id-column-names="id">
      <property name="id" data-type="string"/>
      <property name="name"/>
      <property name="description"/>
      <property name="itemPath" column-name="item_path"/>
      <property name="itemName" column-name="item_name"/>
      <property name="isReadOnly" column-name="is_readonly"
        data-type="boolean"/>
      <property name="formHandler" column-name="form_handler"
        item-type="formHandler"/>
```

```
    <property name="mode" column-name="mode_id" required="true"
      item-type="mapMode"/>
  </table>

  <!-- attributes property -->
  <table name="vmap_attrval_rel" type="multi"
    id-column-name="mapper_id" multi-column-name="name">
    <property name="attributes" column-name="attribute_id"
      data-type="map" component-item-type="attributeValue"
      cascade="update,delete"/>
  </table>

  <!-- viewMappings property, maps to array of ItemViewMapping -->
  <table name="vmap_im2ivm_rel" type="multi" id-column-name="item_id"
    multi-column-name="sequence_num">
    <property name="viewMappings" data-type="list"
      column-name="view_id" component-item-type="itemViewMapping"
      cascade="update,delete"/>
  </table>

</item-descriptor>
```

itemMapping Properties

itemMapping properties include the following:

Property	Description
`name`	The name of the item mapping. For all item mappings, specify the default value, which is an asterisk (*). **Note:** This property anticipates future releases of ATG Content Administration.
`description`	Optional description of the item mapping.
`itemPath`	The Nucleus path of the component that represents the asset's repository or VFS. For example, the `itemPath` for the default text file asset type is set to: `/atg/epub/file/WWWFileSystem`
`itemName`	The repository item descriptor name or the VFS file type of this asset. For example, the `itemName` value for the default text file asset type is `wwwTextFileAsset`.
`formHandler`	A reference to a View Mapping `formHandler` item used to handle input of asset properties.

Property	Description
mode	The map mode to use for this mapping. For more information, see Map Modes.
attributes	Optional key/value pairs that define an attribute of this item mapping.
viewMappings	A list of itemViewMapping references. Each itemViewMapping corresponds to a separate tab of asset information.

itemViewMapping

An itemViewMapping represents a tab in the user interface.

Item descriptor Definition

The `itemViewMapping` item descriptor is defined as follows in `ViewMapping.xml`:

```
<item-descriptor name="itemViewMapping" id-space-names="viewmapping"
    display-property="name" query-cache-size="40">
    <table name="vmap_ivm" type="primary" id-column-name="id">
      <property name="id" data-type="string"/>
      <property name="name"/>
      <property name="displayName" column-name="display_name"/>
      <property name="description"/>
      <property name="view" column-name="view_id" item-type="itemView"/>
    </table>

    <!-- attributeValues property -->
    <table name="vmap_attrval_rel" type="multi"
      id-column-name="mapper_id" multi-column-name="name">
      <property name="attributeValues" column-name="attribute_id"
        data-type="map" component-item-type="attributeValue"
        cascade="update,delete"/>
    </table>

    <!-- propertyViewMappings (PropertyViewMapping) property -->
    <table name="vmap_ivm2pvm_rel" type="multi" id-column-name="ivm_id"
      multi-column-name="name">
      <property name="propertyMappings" column-name="pvm_id"
        data-type="map" component-item-type="propertyViewMapping"/>
    </table>
</item-descriptor>
```

itemViewMapping Properties

itemViewMapping properties include the following:

Property	Description
name	The name for the view in the ACC. You can use the name value to switch between different views for the same set of asset properties; for example, you might have one view for assets displayed and edited in English and another view in Spanish.
displayName	The display name for the view in the ATG Business Control Center—for example, the value that appears in the banner of the asset properties tab.
description	Optional description of this item.
view	Optional reference to an itemView. If omitted, the itemViewMapping uses the default itemView.
attributeValues	Optional key/value pairs that define an attribute of this itemViewMapping.
propertyMappings	A map of asset properties to propertyViewMappings.

itemView

An itemView specifies a JSP fragment that renders a page of information for a particular asset type in a given mode.

Item descriptor Definition

The itemView item descriptor is defined in ViewMapping.xml as follows:

```
<item-descriptor name="itemView" display-property="name" query-cache-size="10">
  <table name="vmap_iv" type="primary" id-column-name="id">
    <property name="id" data-type="string"/>
    <property name="name"/>
    <property name="description"/>
    <property name="uri"/>
    <property name="applicationName" column-name="app_name"/>
    <property name="mode" column-name="mode_id" required="true"
      item-type="mapMode"/>
  </table>

  <!-- attributes property -->
  <table name="vmap_iv2ivad_rel" type="multi" id-column-name="view_id"
    multi-column-name="name">
    <property name="attributes"
      component-item-type="itemViewAttributeDefinition"
      column-name="attr_id" data-type="map" cascade="update,delete"/>
```

```
  </table>
</item-descriptor>
```

itemView Properties

itemView properties include the following:

Property	Description
`name`	The name of the itemView.
`description`	Optional description of the itemView.
`uri`	The URI of the JSP fragment that renders this view.
`applicationName`	Name of the Web application where this view resides. The value is the display name defined for the application in its `web.xml` file.
`mode`	The map mode to use for this view. For more information, see Map Modes.
`attributes`	Optional key/value pairs that define an attribute of this itemView.

Example

The following table shows property settings for the itemView `Standard RepositoryItem Editor`:

Property	Setting
`id`	`2`
`name`	`Standard RepositoryItem Editor`
`description`	`Repository Item Edit View - 1-column layout`
`uri`	`/html/views/item/gsa/oneColumnEdit.jsp`
`applicationName`	`Publishing Portlets`
`mode`	`edit`
`attributes`	`{title=Replacement view title, textAbove=Text above the form}`

propertyViewMapping

A propertyViewMapping identifies the propertyView used to render a specific property, or a component property if the property type is a collection. A propertyViewMapping can override propertyView attributes when the default settings are inappropriate.

Item Descriptor Definition

The `propertyViewMapping` item descriptor is defined as follows in `ViewMapping.xml`:

```
<item-descriptor name="propertyViewMapping" id-space-names="viewmapping"
    display-property="description" query-cache-size="300">
    <table name="vmap_pvm" type="primary" id-column-name="id">
      <property name="id" data-type="string"/>
      <property name="description"/>
      <property name="propertyView" column-name="pview_id"
        item-type="propertyView"/>
      <property name="componentPropertyView" column-name="cpview_id"
        item-type="propertyView"/>
    </table>

    <!-- attributeValues property -->
    <table name="vmap_attrval_rel" type="multi" id-column-name="mapper_id"
      multi-column-name="name">
      <property name="attributeValues" column-name="attribute_id"
        data-type="map" component-item-type="attributeValue"
        cascade="update,delete"/>
    </table>

    <!-- componentAttributeValues property -->
    <table name="vmap_cattrval_rel" type="multi"
      id-column-name="mapper_id" multi-column-name="name">
      <property name="componentAttributeValues" column-name="attribute_id"
        data-type="map" component-item-type="attributeValue"
        cascade="update,delete"/>
    </table>
  </item-descriptor>
```

propertyViewMapping Properties

propertyViewMapping properties include the following:

Property	Description
`description`	Display name for this item
`propertyView`	Optional reference to a propertyView item. If omitted, the propertyViewMapping uses the default propertyView.
`componentPropertyView`	Optional reference to a propertyView item for component properties, if this mapping refers to a collection type.
`attributeValues`	Optional key/value pairs that affect the `propertyView`.

Property	Description
componentAttributeValues	Optional key/value pairs that affect the componentPropertyView.

propertyView

A propertyView specifies the JSP fragment that renders a specific property in an itemView. It also identifies the property's mode and data type. For example, a propertyView can specify a JSP fragment that is used to edit a string or view a boolean value.

A propertyView defines an attribute map of values that are used in the JSP to control how the property is presented—for example, by specifying the number of rows and columns for displaying a text area.

Note: To find the JSP fragment that is used to render a specific property, display the page source and search for the label of the property you are interested in. Default views surround page fragments with comments such as the following:

```
<!-- Begin jsp URI /html/views/property/bigstring/htmlEdit.jsp -->

<!-- End jsp URI /html/views/property/bigstring/htmlEdit.jsp -->
```

Item Descriptor Definition

The `propertyView` item descriptor is defined as follows in `ViewMapping.xml`:

```
<item-descriptor name="propertyView" display-property="name"
query-cache-size="300">
    <table name="vmap_pv" type="primary" id-column-name="id">
      <property name="id" column-name="id" data-type="string"/>
      <property name="type" column-name="type" data-type="string"/>
      <property name="name" column-name="name" data-type="string"/>
      <property name="description" column-name="description"
        data-type="string"/>
      <property name="uri" column-name="uri" data-type="string"/>
      <property name="applicationName" column-name="app_name"
        data-type="string"/>
      <property name="isDefault" column-name="is_default"
        data-type="boolean"/>
      <property name="isReadOnly" column-name="is_readonly"
        data-type="boolean"/>
      <property name="isComponentPropertyView" column-name=
         "is_component" data-type="boolean"/>
      <property name="mode" column-name="mode_id" required="true"
        item-type="mapMode"/>
    </table>
```

```
    <!-- attributes property -->
    <table name="vmap_pv2pvad_rel" type="multi" id-column-name="view_id"
      multi-column-name="name">
      <property name="attributes"
        component-item-type="propertyViewAttributeDefinition"
        column-name="attr_id" data-type="map" cascade="update,delete"/>
    </table>
</item-descriptor>
```

propertyView Properties

propertyView properties include the following:

Property	Description
`type`	The data type to which this view applies.
`name`	The name for the view in the ACC.
`description`	Optional description of the propertyView.
`uri`	The URI of the JSP that renders this view.
`applicationName`	Name of the Web application where this view resides. The value is the display name defined for the application in its `web.xml` file.
`isDefault`	A boolean property, where true specifies to use this view as the default for this property, when in the specified mode.
`isReadOnly`	A boolean property, where true specifies to disallow editing of this view.
`isComponentPropertyView`	A boolean property, where true specifies this view is used for a component property (component of a collection).
`mode`	The map mode to use for this view. For more information, see Map Modes.
`attributes`	Optional key/value pairs that define an attribute of this propertyView.

Example: Setting propertyView Properties for an HTML Editor

The following table shows the `WYSIWIG HTML Editor` propertyView—the default HTML editor that is used in the ATG Business Control Center to edit `big string` properties:

Property	Setting
type	big string
name	WYSIWYG HTML Editor
description	WYSIWYG HTML editor for big string properties
uri	/html/views/property/bigstring/htmlEdit.jsp
isDefault	true
isReadOnly	true
isComponentPropertyView	false
applicationName	Publishing Portlets
mode	edit
attributes	key/value pairs that define this property editor's attributes; see the example that follows.

The following table shows the `attributes` property keys that are set to control the display and behavior of the propertyView `WYSIWYG HTML Editor`:

Key	Value
appletHeight	Height of EditLive applet, in pixels
appletWidth	Width of EditLive applet, in pixels
columns	Number of columns
defaultValue	Default value for field
dictionary	Spell check dictionary to use
inputFieldMaxLength	Maximum length of the data in the field
rows	Number of rows
simpleUI	Switch to show the applet in simple mode
spellCheck	Allow spell check on the field
textAboveField	Text to display above the field
textBelowField	Text to display below the field
title	Replacement field title

For more information on the WYSIWYG HTML editor, see Configuring the EditLive! HTML Editor.

Note: When specifying a property name in a `propertyView` item, use the property's actual name as specified in the repository, which can differ from its display name in the ATG Business Control Center. You can determine a property's name through the ATG Dynamo Server Admin: in the Component Browser, navigate to the repository that contains the asset type you are working with.

Hiding Properties

By default, the standard views display all asset properties. To prevent display of a property in the ATG Business Control Center, set up a propertyViewMapping for that property and map it to a Hidden propertyView. You can also use the tag `pws:categorize` to restrict the properties that appear on a page, individually or by category.

Sorting Properties

The order in which properties appear on a page is controlled by the tag `pws:Categorize`, which includes attributes for sorting properties by their `category` value in the repository definition file.

Map Modes

Map modes provide symbolic names used to categorize itemViews and propertyViews by function. The following map modes are supplied with ATG Content Administration:

- `browse`
- `conflict`
- `diff`
- `edit`
- `pick`
- `view`

Item Descriptor Definition

The `mapMode` item descriptor is defined as follows in `ViewMapping.xml`:

```
<item-descriptor name="mapMode" display-property="name" query-cache-size="10">
  <table name="vmap_mode" type="primary" id-column-name="id">
    <property name="id" data-type="string"/>
    <property name="name"/>
    <property name="description"/>
    <property name="fallbackMode"
      column-name="fallback_id" item-type="mapMode"/>
  </table>
</item-descriptor>
```

Map modes can include a fallback mode, which identifies the mode that should be used if a default propertyView is not found in the primary mode. This behavior can be used to minimize the number of

default propertyViews that an application requires. Fallback modes are searched recursively until a mode with an empty fallback mode or a mode that has already been encountered is reached.

You can add new map modes if required. The following example assumes you have a business requirement to display asset properties so that they fit in a limited amount of space. It shows the basic steps for setting up a new map mode to fulfill this requirement and specify its fallback mode—in this case, `view`.

1. Create the item for the new mode—in this example, `condensedView` –and set its fallback mode to `view`.
2. In the appropriate JSP, add a getItemMapping tag with a mode parameter set to `condensedView`:

   ```
   <biz:getItemMapping var="imap"
      mode="condensedView" item="${item}"/>
   ```

3. Create `propertyView` items that truncate potentially large data values, and set their mode to `condensedView`.

Note: You can also set the fallback mode within the `getItemMapping` tag. For example:

```
<biz:getItemMapping var="imap" mode="condensedView"
  fallbackMode="view"
  item="${item}"/>
```

View Mapping Form Handlers

An itemMapping can specify a Nucleus form handler component to handle input of asset data. The form handler item in the View Mapping repository also provides attributes used to modify the behavior of the form handler that the item represents.

Item Descriptor Definition

The `formHandler` item descriptor is defined as follows in `ViewMapping.xml`:

```
<item-descriptor name="formHandler" display-property="name"
  id-space-names="viewmapping" query-cache-size="20">
  <table name="vmap_fh" type="primary" id-column-name="id">
    <property name="id" data-type="string"/>
    <property name="name"/>
    <property name="description"/>
    <property name="path" column-name="component_path"/>
  </table>

  <!-- attributes property -->
  <table name="vmap_attrval_rel" type="multi"
    id-column-name="mapper_id" multi-column-name="name">
    <property name="attributes" column-name="attribute_id"
```

```
      data-type="map" component-item-type="attributeValue"
      cascade="update,delete"/>
  </table>
</item-descriptor>
```

formHandler Item Properties

formHandler item properties include the following:

Property	Description
`name`	The form handler's name in the ACC
`description`	Optional description of this formHandler
`path`	The Nucleus path of this formHandler component
`attributes`	Optional key/value pairs. This property sets the `attributes` property of the form handler class `atg.epub.servlet.AssetFormHandler`.

Form handlers are provided for all the default asset types in ATG Content Administration. The following table shows the form handler items in the view mapping repository and the form handler component to which they point.

ACC form handler items	Form handler component in /atg/epub/servlet
`DefaultFile`	`BinaryFileAssetFormHandler`
`DefaultRepository`	`RepositoryAssetFormHandler`
`DefaultRepositoryConflict`	`ProjectDiffFormHandler`
`DefaultRepositoryDiff`	`AssetDiffFormHandler`
`File Folder Form Handler`	`FolderAssetFormHandler`
`File Form Handler`	`FileRepositoryAssetFormHandler`
`ViewMapping:formHandler:100062`	`TextFileAssetFormHandler`
`ViewMapping:formHandler:100080`	`SegmentAssetFormHandler`

View Mapping system form handlers must implement this interface:

`atg.epub.servlet.AssetFormHandler`

In general, form handlers should subclass `RepositoryAssetFormHandler` or one of the file asset form handler classes in `atg.epub.servlet`:

- `BaseFileAssetFormHandler`
- `FolderAssetFormHandler`
- `GenericFileAssetFormHandler`
- `BinaryFileAssetFormHandler`
- `TextFileAssetFormHandler`

The *ATG API Reference* describes `atg.epub.servlet.AssetFormHandler`. For more about View Mapping form handlers, see Appendix C: Form Handlers.

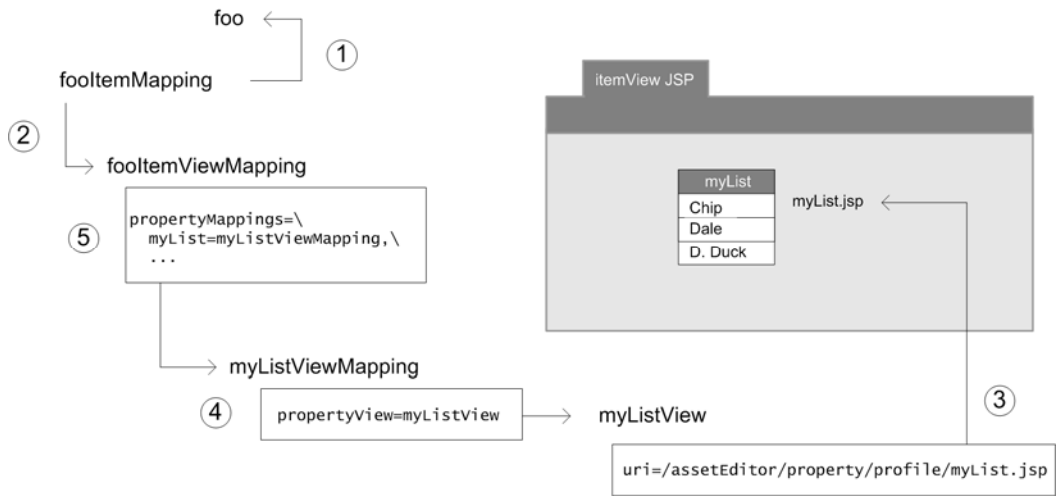
Overriding Default Asset Display Settings

View mapping objects are provided for all default asset types and properties in ATG Content Administration. Thus, you can display most asset or properties without configuring any view mappings; the View Mapping system uses the default items to represent assets.

To change the presentation of a default asset type—for example, to change a property so it uses a dropdown list instead of radio buttons as an edit control—you must configure an appropriate view mapping item.

The View Mapping system acts as a hierarchy of overrides, so you only need to configure the parts of a view that differ from the default. For example, to customize the display of the `myList` property in asset type `foo`:

1. Create an itemMapping for asset type `foo`.
2. Link the new itemMapping to a new or existing itemViewMapping.
3. Create a propertyView and corresponding JSP fragment for the `myList` property.
4. Create a propertyViewMapping whose propertyView property references the new propertyView `myList`.
5. In `foo`'s itemViewMapping, map the asset type's `myList` property to its propertyViewMapping via the itemViewMapping's property `propertyMappings`.



Thereafter, the new propertyView is used when you edit an asset of type `foo`. The View Mapping system renders the asset with the changed property setting, which overrides the default setting.

Caution: Always create view mapping items rather than edit the default ones, as many default items are used more than once in the ATG Business Control Center. Thus, changing an item can unintentionally affect multiple asset types or properties.

The basic process for creating items for the view mapping repository—for example, a propertyView or itemViewMapping repository item—is the same:

1. Access the ACC and display the Publishing > View Mapping task window.

   Note: The View Mapping menu displays only if your user profile belongs to a group that has the `Publishing: EPublishing Repository` option enabled in the People and Organizations > Control Center Groups window. For more information on enabling Control Center Group options, refer to the *ATG Programming Guide*.

2. In the List field, specify the type of item to create, then click List. Existing items of this type are listed. It can be helpful to review these as examples.
3. Click New Item. The New Item dialog box appears, with fields appropriate for a new item of the type to create.

Using Resource Bundles

In order to support internationalization, you can specify resource bundles at the following levels, listed in descending order of precedence:

1. propertyViewMapping
2. itemViewMapping
3. itemMapping

You specify a resource bundle by adding the `resourceBundle` attribute to the appropriate property:

For this View Mapping component:	Set resourceBundle in this property:
propertyViewMapping	`attributeValues` `componentAttributeValues`
itemViewMapping	`attributeValues`
itemMapping	`attributes`

Default Resource Bundle

If no resource bundle is explicitly specified at any level in the View Mapping hierarchy, the View Mapping system looks for the default resource bundle of the current asset's repository. The View Mapping system finds this bundle by translating the repository's Nucleus pathname into a `CLASSPATH` as follows:

`nucleus-path``/`*`repositoryName`*`=c`*`lasspath`*`.`*`repositoryName`*Resource

For example, given the repository name `/company/Catalog`, the View Mapping system looks for a resource bundle along the `CLASSPATH` as follows:

```
company.CatalogResource
```

getItemMapping

A JSP can call view mapping functionality by including a `getItemMapping` tag. This tag is defined in the `bizui` tag library:

```
<ATG10dir>/BIZUI/taglibs/bizui/tld/bizui-1_0.tld
```

In order to call `getItemMapping`, the JSP must include this `taglib` directive:

```
<%@ taglib prefix="biz" uri="http://www.atg.com/taglibs/bizui" %>
```

Parameters

`getItemMapping` has the following parameters:

Parameter	Description
`fallbackMode`	The fallback mode.
`item`	The versioned repository item or virtual file

Parameter	Description
itemName	The item descriptor name or VFS file type of the item (if the item parameter is not specified)
itemPath	The Nucleus component path of the item (if the item parameter is not specified)
mode	The map mode to use for this item. For more information, see Map Modes.
propertyList	A list of properties that are defined in the specified item or item descriptor. If this parameter is set, the View Mapping system processes only the listed properties.
readOnlyMode	Optionally specifies the mode to use for read-only properties.
showDefault	A boolean property that specifies whether to show properties that do not have explicit mappings. The default is true.
showExpert	A boolean property that specifies whether to show (true) or hide (false) expert properties.

For usage examples, see the pages in this directory:

```
<ATG10dir>/PubPortlet/PubPortlets.ear/portlets.war/html/ProjectsPortlet
```

For example, `assetEditPage.jsp` contains an instance of the `getItemMapping` tag used in the context of a top-level `itemMapping`.

Setting Up Linked Assets

You can configure an asset property to expect a link to another asset as its value. To do so, set up the repository definition so the parent asset property is defined as a repository item of the type to reference as the child asset. In the following example, the definition of property `xyz_ref` references an item of type `xyz_rep_item`:

```
<property column-name="xyz_ref_obj" name="xyz_ref"
  item-type="xyz_rep_item"
  repository="/AnyCorp/AnyMachine/MyAssetRepository"
  category="Basics" display-name="Reference"
  required="false" readable="true" writable="true"
  queryable="true" hidden="false" expert="false"
  cache-mode="inherit">
</property>
```

By default, the propertyView editor that is used in the ATG Business Control Center for properties of type repository item is `popUpPickerEdit.jsp`. The editor supplies buttons for creating and deleting the link as well as the mechanism for displaying the asset where the link points.

You can link any number of assets in this way: assets can have multiple child assets that link to their own child assets or back to their parents, and so on.

The type of asset to which a link can be created is constrained to the single repository item type you specify in the definition. Thus, business users can link to only one type of asset from each property. You can, however, specify multiple sub-types as the property type. In this case, business users can select any asset belonging to any specified sub-type as the target of the link. For more information on working with repository item sub-types, refer to *Item Descriptor Inheritance* in the *ATG Repository Guide*.

Property links to other assets can also be defined as any of the following types:

- List
- Map
- Set

The editor automatically displays controls that are appropriate for each type.

When business users create links to assets through the asset editor in the ATG Business Control Center, the target assets are automatically added to the current project and deployed like any other asset. For more information, see *Creating Linked Assets* in the *ATG Content Administration Guide for Business Users*.

Troubleshooting the ViewMapping System

The ViewMapping system can be analyzed and debugged at several levels:

- Rendered JSP page source
- Asset form handler log file
- ViewMappingFactory log file

Rendered JSP Page Source

You can view the HTML source of rendered itemView and propertyView JSP fragments. By doing so, you can determine which item mappings were used to render the asset. For example:

```
<!--
Mapping Information --------------------
description: Default Editable ItemMapping for Repository Items
itemPath: /atg/devtest/SecuredTestRepository
itemName: descriptor1
view: DefaultRepEdit, context root: /PubPortlets URI:
/html/views/item/gsa/oneColumnEdit.jsp
-->
```

For each rendered standard property, the HTML source includes comments that wrap the property. For example:

```
<!-- Begin JSP, context root /PubPortlets, URI
/html/views/property/enumerated/radioGroupEdit.jsp -->
<!-- End JSP, context root /PubPortlets, URI
/html/views/property/enumerated/radioGroupEdit.jsp -->
```

Note: Custom property views should follow this convention, but are not required to do so.

Asset Form Handler Log File

ATG Content Administration provides three form handlers to view and edit assets. You can turn on debugging for these form handler components by setting their `loggingDebug` properties to true in `localconfig`. The standard form handler component paths are:

- `/atg/epub/servlet/RepositoryAssetFormHandle`
- `/atg/epub/servlet/BinaryFileAssetFormHandler`
- `/atg/epub/servlet/TextFileAssetFormHandler`

You can also turn on debugging within the JSP `assetEditPage.jsp` by modifying it in two locations. To turn on debugging for the page rendering, modify the JSP as follows:

```
<!-- Import form handler specified in mapping -->
<dspel:importbean var="formHandler" bean="${imapformHandler.path}"/>
<c:set target="${formHandler}" property="loggingDebug" value="true"/>
```

To turn on debugging for the form submit:

```
<dspel:input type="hidden" value="${assetInfoPath}" priority="100"
bean="${imap.formHandler.path}.assetInfoPath"/>
<dspel:input type="hidden" value="${debug}" priority="100"
bean="${imap.formHandler.path}.loggingDebug"/>
```

ViewMappingFactory Log File

You can obtain detailed information on how the ViewMapping system renders items and properties through JSP fragments, by setting the DefaultViewMappingFactory component's `loggingDefault` property to `true`. In order to render an item and its properties for viewing, the DefaultViewMappingFactory performs these tasks:

1. Assembles custom itemViewMappings for the item to view, if any exist; otherwise, it maps to the default itemViews.
2. Attaches to the itemViewMappings any custom propertyViewMappings that might exist; otherwise, it uses the default propertyViewMappings.

The log file contains detailed information on this process.

Configuring the EditLive! HTML Editor

ATG Content Administration includes a propertyView that displays a Java applet you can use to edit HTML pages with the WYSIWYG HTML editor EditLive!. The output is an HTML fragment that can be included in a page. The propertyView is set up to appear automatically when users edit any property of type `big string`.

To configure the editor for custom asset properties, create appropriate view mapping items that include the WYSIWYG HTML Editor propertyView. The default attributes of this item are shown as an example in the section propertyView.

Because the output of the editor is an HTML fragment, it omits `<html>` and `<body>` tags. If the asset is an HTML content item that is propagated to the file system, and it is then accessed directly from a browser instead of being included in another page, you might have to add `<html>` or `<body>` tags as a post-edit or through the Code tab of the editor.

When displaying on a page the value of a property that uses the WYSIWYG editor, use the `valueishtml` directive to suppress display of HTML tags on the page:

```
<dsp:valueof valueishtml="true" param="element.data">
```

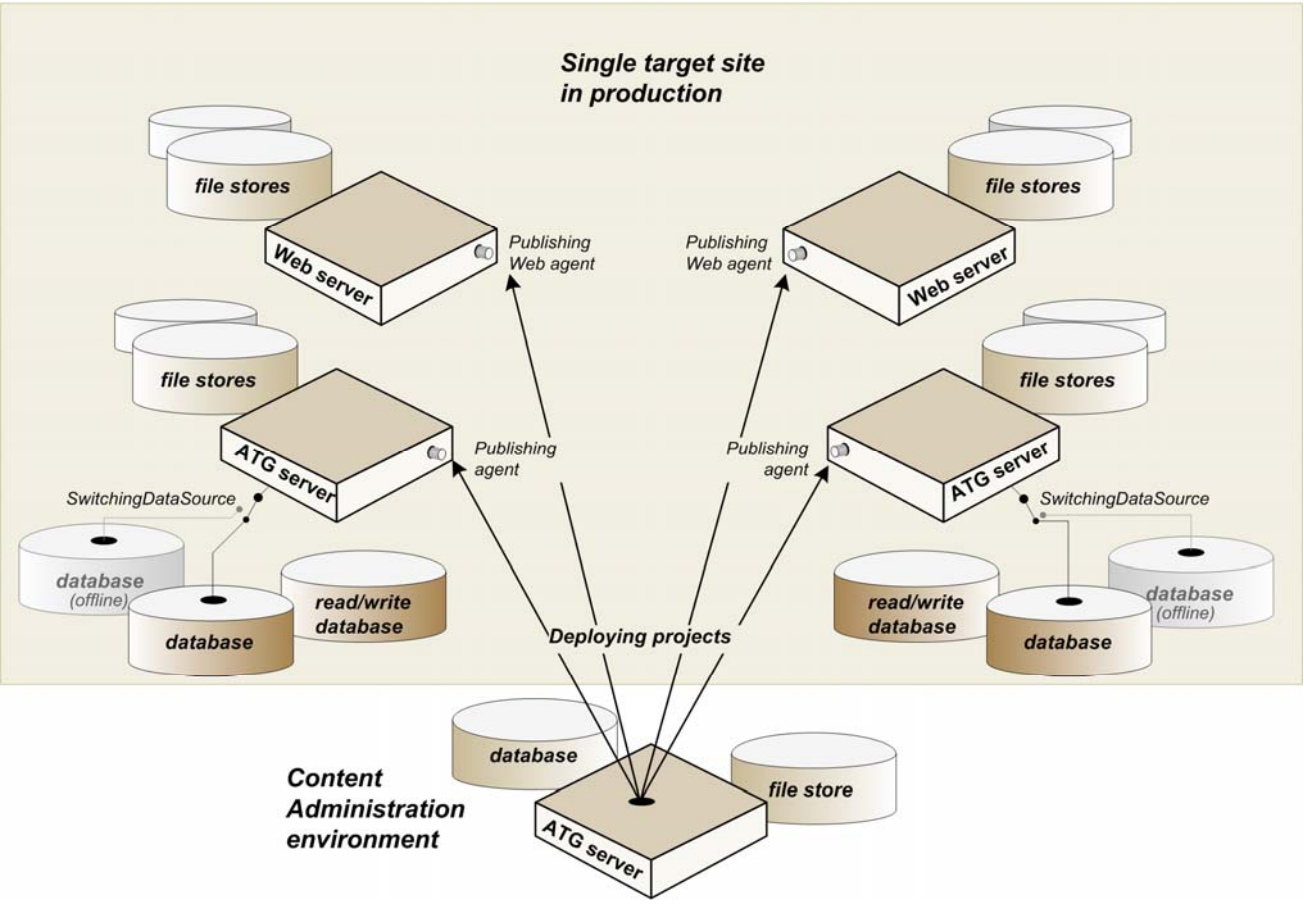
10 Deployment Concepts

This chapter introduces concepts and features that are central to ATG Content Administration deployment. It includes the following sections:

After reading this chapter, see the chapters that follow: Setting Up Deployment, for information about the steps required to configure a asset management server for deployment; and Deploying Project Assets, on different ways to execute a deployment.

Target Site Basics

In ATG Content Administration, a target site is a logical grouping of one or more servers that serve the same content. When you deploy assets, you deploy them to a specified target site. The following diagram illustrates a single target site in a production environment:



This diagram shows deployment of several projects to a single target site in a production environment. A production or staging environment can include one or several target sites (see Identify Deployment Target Sites). When you deploy projects to a site, every server in the site is updated with the asset versions from those projects. The diagram shows both server clusters in the site being updated by a deployment that is in progress.

This diagram shows a single target site; however, your configuration can include multiple sites. Moreover, each site can include multiple server clusters.

Note: In the context of ATG Content Administration deployment, the term *target site* or *site* denotes a logical grouping of one or more servers where projects are deployed. In this sense, your staging and production environments can include one site or several sites. However, a single- or multi-server ATG Content Administration environment cannot be used to manage the assets for multiple Web sites—for example, `www.mywebsite1.com` and `www.mywebsite2.com`.

Target site servers run one of the following agents:

- Publishing agent on the ATG servers
- Publishing Web agent on the Web servers

Publishing Agent

The Publishing agent is a service that runs on a target site's ATG servers and manages deployment of repository and file assets on them. The Publishing agent performs database caching and switch deployment operations on the target. It also maintains status information about the currently live datastore for switch deployments, the deployment state, and the deployed snapshot.

The Publishing agent includes the virtual file system (VFS) `ConfigFileSystem`, which is configured out-of-the-box to facilitate deployment of file assets to the ATG server, including scenarios, targeters, content and profile groups, segments, and slots. Like the corresponding VFS on the asset management server, this VFS is located in Nucleus at:

```
/atg/epub/file/ConfigFileSystem
```

For detailed information, see Repositories.

The Publishing agent is installed with the ATG platform.

Publishing Web Agent

The Publishing Web agent is a service that runs on target site's Web servers, and manages deployment of file assets on them. It includes the virtual file system (VFS) `WWWFileSystem`, which is configured out-of-the-box to facilitate deployment of Web content (static text and binary files) to the Web server. Like the corresponding VFS on the asset management server, this VFS is located in Nucleus at:

```
/atg/epub/file/WWWFileSystem
```

For detailed information, see Repositories.

The Publishing Web agent must be installed manually.

Deployment Process

ATG Content Administration uses the multi-threaded deployment features provided by the Dynamo Application Framework (DAF) to transfer data from the versioned source on the asset management server to the target site. DAF deployment uses the JDBC driver to copy repository assets to the target, and a TCP connection to transfer file assets.

ATG Content Administration integrates with DAF deployment through the ATG Content Administration deployment API, whose starting point is the DeploymentServer. The DeploymentServer represents the core controller of all ATG Content Administration deployment operations and provides API access to the targets and agents where assets are deployed.

The deployment process can be outlined as follows:

1. The ATG Content Administration deployment system assembles the data to deploy and provides that data to the DAF DeploymentManager.

2. The DeploymentManager that handles the deployment sends JMS messages to DeploymentManager instances on other ATG servers.
3. Each DeploymentManager instance spawns multiple threads to perform the deployment.
4. After the data transfer is complete, the DeploymentManager returns control to the ATG Content Administration deployment system.

Note: If the deployment process fails at any point, the DeploymentManager returns control to the ATG Content Administration deployment system. For more information on failure handling, see Recover from Deployment Failure.

Enabling Distributed Deployments

By default, deployment events are sent locally and only to the server that initiated the deployment. If you want to use distributed deployment instead, so deployment events are sent as JMS messages to any configured listener, set the DeploymentManager's `useDistributedDeployment` property to true. For more information about distributed deployment, see *Using DAF Deployment* in the *ATG Programming Guide*.

Post-Deployment Tasks

After a successful deployment, ATG Content Administration performs these tasks:

- Updates each agent's status.
- If the environment is configured for switch mode deployments, performs the appropriate switches on the agent servers' switchable repositories and VFSs.
- Invalidates repository caches where necessary.

DAF deployment architecture and its processes and phases are described in detail in the *Using DAF Deployment* chapter of the *ATG Programming Guide*.

Deployment Scope

The scope of asset deployment to a target site is determined by the type of deployment that you choose. Two deployment type options are available:

- Full deployment
- Incremental deployment

Your decision to perform a full or incremental deployment can depend on various factors, such as the number and type of assets to deploy, the frequency of deployment, and the reason for deployment.

Full Deployment

A full deployment copies all assets from the asset management server to a target site. A full deployment comprises two steps:

1. All assets are deleted from the target site.
2. All assets from projects that are currently and previously deployed to the target site are written to that site.

Note: A full deployment to a one-off target only includes assets from checked-in projects. Assets from deployed but still-active projects are excluded from the deployment.

On a site with many assets, full deployments can be time consuming and resource intensive. Initiate a full deployment in order to refresh all site assets, especially when they are modified directly or are corrupt.

You can limit the scope of a full deployment in two ways:

- The SelectiveDeleteVFSService lets you specify which file assets to delete from the local file system during full deployment. You can use this VFS in order to limit the scope of full deployment to JSP files only, and protect other file asset types from deletion.
- An item descriptor's `deployable` attribute can be set to false in order to exclude specific repository item types from deployment:

  ```
  <attribute name=deployable value="false"/>
  ```

  This attribute setting must be consistent in the versioned and target site repositories; otherwise, attempts to deploy assets of this type fail.

Incremental Deployment

An incremental deployment only updates the site with asset changes from the deployed projects; these changes include add, update, and/or remove operations. This type of deployment is more efficient than a full deployment, especially when few site assets changed since the previous deployment.

Deployment Modes

The mode of deployment determines whether you update the site directly or indirectly. Two deployment mode options are available:

- Online deployment deploys updated assets directly to the live site. Online deployment is appropriate only for development and testing.
- Switch deployment deploys updated assets to an off-line file directory or database, then swaps these with assets on the live site as a single transaction. Production sites should always use switch deployment.

For demonstration and evaluation purposes, both the Publishing agent and the Publishing Web agent are configured by default to manage online deployments.

Online Deployment

In an online deployment, a site's repository assets are updated in one or more transactions and activated (made live on the site) as each transaction is committed. File assets are updated on the site and activated as each file is written to the local file system.

Switch Deployment

A production target that is configured for switch deployment requires a pair of databases for the same set of repository assets: one that is active and the other inactive at any given time. Similarly, the same target requires a pair of active and inactive directories for the same set of file assets.

A switch deployment updates a target site's repository and file assets as follows:

- Repository assets are updated on the site's inactive database while the site's live database continues to run undisturbed. When updates to the inactive database are complete, the site's SwitchingDataSource switches its underlying data source to the updated database. The newly inactive database is updated to reflect the same deployment.
- File assets are updated on the site's inactive directory while the site's live directory runs. When updates to the inactive directory are complete, the site switches to it: the inactive directory becomes the live directory, and vice-versa. The inactive directory is then updated to reflect the same deployment.

Note: Switch deployment of JSPs to a Web application is not supported: the inactive directory where JSPs are first deployed can only contain the JSPs that are managed in and deployed from the ATG Content Administration environment. This excludes most Web applications, which include other resources such as servlets.

Online versus Switch Deployments

Switch deployment is generally favored for production environments, where it is important that all updates take place immediately and without error. Online deployment is liable to put a live Web site temporarily in an inconsistent state containing both new and stale data; and it can also disrupt performance. Moreover, failure of an online deployment can put the site in an unknown state and render it unavailable for an extended period of time while the problem is diagnosed. The solution—often a full deployment—can itself be time-consuming.

Switch deployment avoids the problems associated with online deployment:

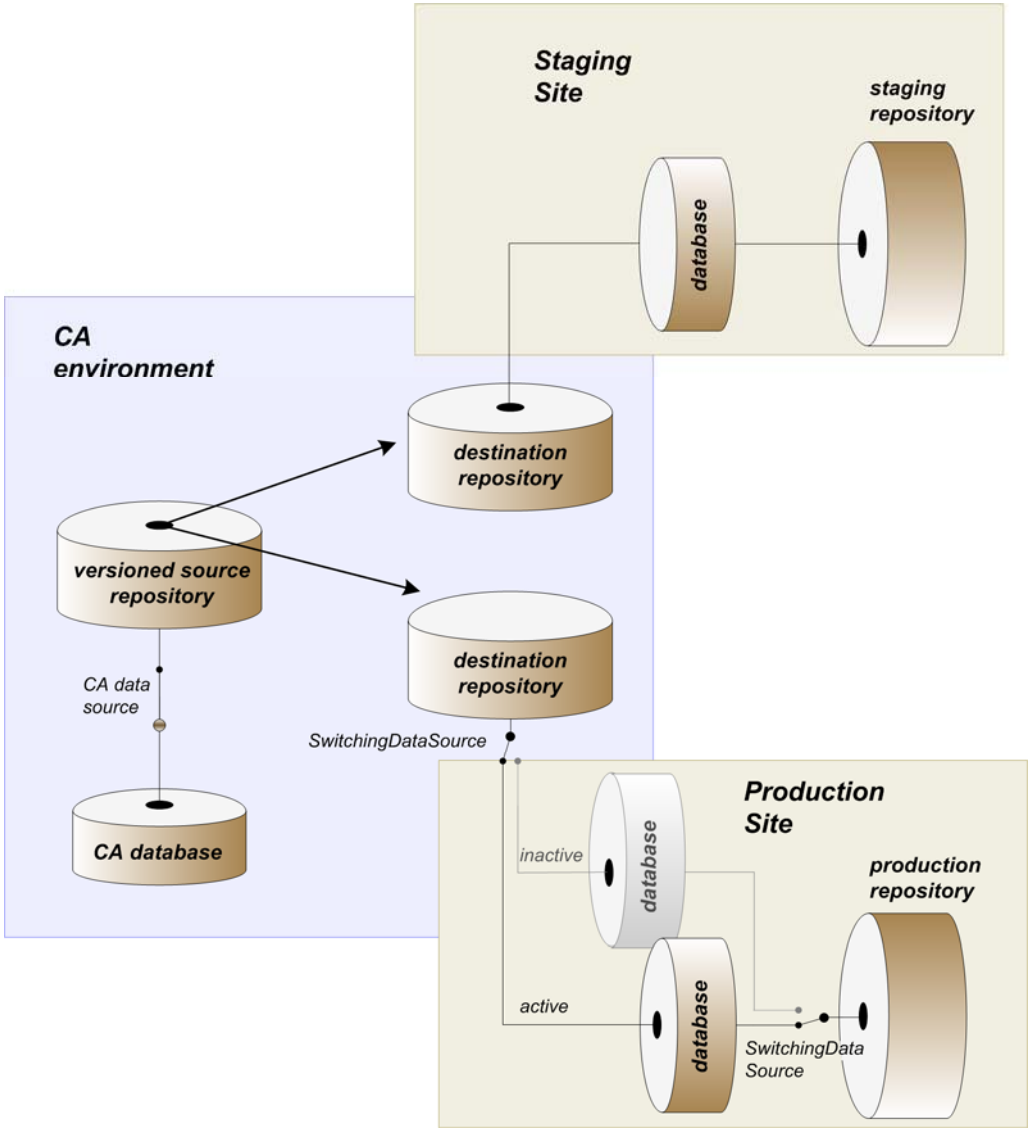
- All new data is available to the live site at the same time, no matter how much data is moved.
- Switch deployment failures only affect the offline server, so deployment failures have no effect on the live production site.

Online deployment is generally advisable for development and testing purposes, where easier setup and faster deployment is desirable, and the potential risks are not a major concern.

Destination Repositories

DAF deployment requires a *destination repository* on the asset management server for every target repository that receives deployed data. A destination repository is an unversioned GSARepository instance that points to the database of the matching target site repository. Only repository asset deployment requires a destination repository. File assets are deployed directly from the asset management server to the target sites over a TCP connection.

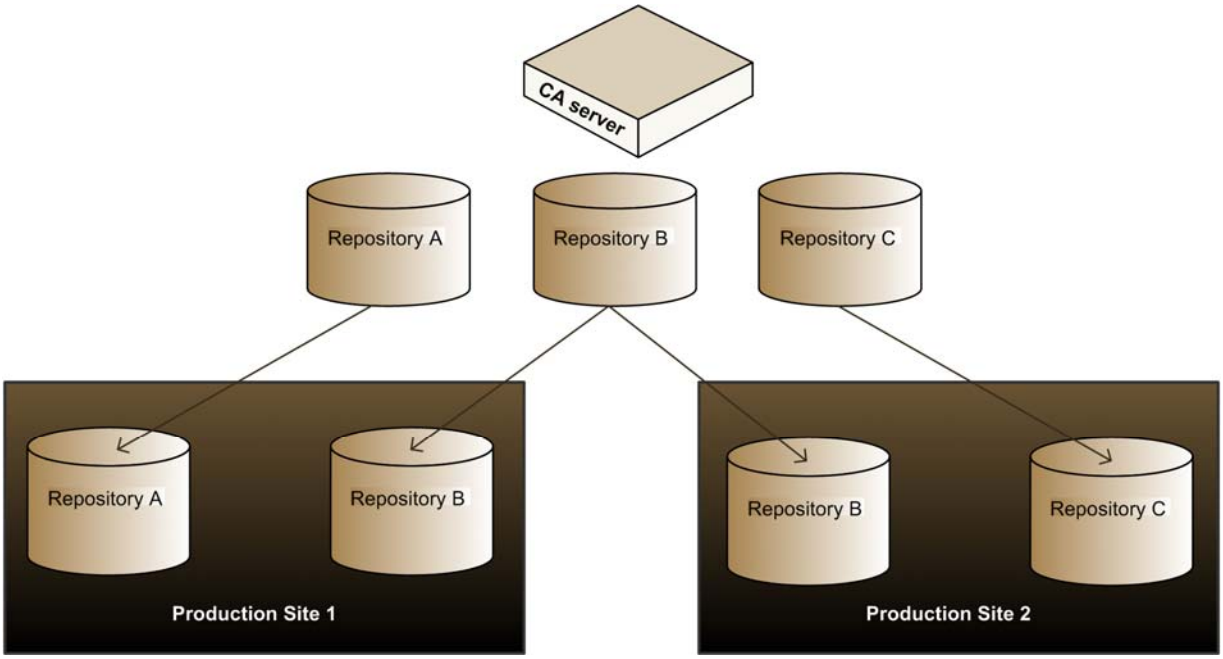
The following graphic shows one versioned source repository that deploys to two target sites, staging and production. The asset management server has two destination repositories, one for staging and another for production. The production site is configured for switch deployment: the production site repository and the corresponding destination repository each point to a SwitchingDataSource that directs them to the active data source.



As part of the deployment setup process for ATG Content Administration, you need to create and configure destination repositories (see Configure Deployment Data Sources and Destination Repositories). Later, when you set up deployment topology, you specify mappings between the source and destination repositories.

Deploying Unique Data to Multiple Sites

DAF Deployment architecture lets you deploy from a single asset management server to multiple sites, including multiple sites on a single target. Each site can receive a unique set of data. For example, site A can receive data from product catalog A while site B receives data from catalog B. You set up this type of configuration by omitting the appropriate repositories from the target sites. If the target repositories do not exist, ATG Content Administration does not deploy to them.



To set up this type of environment, create corresponding repositories on the target server or servers only for the repositories to be deployed from the ATG server. Data is only deployed to matching repositories.

In the illustration, production site 1 has no matching repository for source repository C, so this target site does not receive data from that repository. Similarly, repository A is not configured on production site 2, so this target does not receive data from repository A.

Note: Any repositories with data dependencies—for example, assets that have links to other assets—must be deployed to the same target.

Deploying From Multiple Asset Management Server Clusters

Multiple clusters of asset management servers can deploy to a single target site. In order to avoid conflicts among files that are deployed from different clusters, deployment agents use cluster identifiers to assign ownership to the deployed files. This enables agents to differentiate files from various clusters that share the same paths and names. During deployment, agents check file ownership before deciding whether to overwrite existing files. The site is configured with a file ownership strategy that determines how to resolve ownership conflicts among file system assets.

For detailed information on enabling deployment from multiple ATG Content Administration clusters to the same site, and configuring that site to handle deployment from multiple clusters correctly, see Configure Deployment from Multiple Asset Management Server Clusters.

Deployment Scheduling

A project's workflow determines that deployment schedules are managed in one of two ways:

- The workflow is configured to schedule project deployment immediately. Doing so bypasses the need for a deployment administrator or the RecurringDeploymentService to schedule deployments.
- A workflow element flags a project as approved for deployment. When this stage in the workflow occurs, the project appears in the Admin Console in the ATG Business Control Center. A deployment administrator can manually schedule deployment for the project, or set up the RecurringDeploymentService to schedule approved projects for automatic deployment in batches at a predetermined time—for example, every night at 3 AM.

A deployment that is scheduled by the workflow for immediate deployment only pertains to one project at a time. Deploying from the RecurringDeploymentService or the Admin Console allows simultaneous deployment of multiple projects.

For details on scheduling a deployment manually in the Admin Console, see Deploying from the Admin Console. For information about scheduling a deployment automatically, see Configuring the RecurringDeploymentService.

For information on scheduling a deployment manually or automatically via a project workflow, see Project Workflows.

Deployment Queue

However a deployment is scheduled, all deployments pass through a deployment queue. Each deployment target maintains a separate queue of deployments. Deployments go immediately into the queue, or the Admin Console assigns them a time to enter the queue. Queued deployments run in order of entry; however, you can reorder them through the Admin Console.

Fulfiller Service

The deployment server has a fulfiller service that periodically passes through each queue and performs tasks such as moving deployments whose deploy time has arrived into the appropriate queue. You can configure the frequency of these tasks through the `schedule` property in the `/atg/epub/deployment/DeploymentFulfiller` component.

The deployment fulfiller service is solely responsible for starting deployments. When a deployment reaches the head of the deployment queue, the fulfiller acts as follows:

1. Verifies the target site is in a deployable state—that is, the target site is: accessible, not in an error state, and not running a current deployment.
2. Moves the deployment out of the queue and into the position of the current deployment.
3. Starts the deployment and moves on.

Deployments that complete successfully delete themselves and nudge the fulfiller to check the queue for the next deployment. Deployments that fail wait in an error state and display as such in the Admin Console until a deployment administrator corrects the problem or abandons the deployment.

Interrupting Deployments

If necessary, you can suspend and resume target queues individually from the Admin Console. The deployment server never automatically resumes a suspended queue. However, projects can continue to schedule deployments to the suspended queue's site.

You suspend a queue when you need to bypass the normal deployment fulfiller cycle. After suspending the deployment queue for a specific target site, you can move the desired deployment to the head of the queue, and choose Run Now in the Admin Console. For more information, see Deploying from the Admin Console in the Deploying Project Assets chapter.

One-Off Deployments

Project workflows provide versioning and asset locking safeguards that are important for managing deployment to staging and production sites. However, you might also need to evaluate project content before it is ready for workflow deployment. For example, after authoring new content across multiple projects, you might wish to assess its impact on performance, or check graphic design elements before submitting the projects for review. In this case, you can deploy project data to a site that is defined as a one-off target—that is, a target that accepts deployments outside the workflow and versioning safeguards that ATG Content Administration otherwise imposes.

Note: You can also set up a preview server in order to view project assets on a simulated Web application; unlike a one-off target, a preview server is subject to ATG Content Administration controls. For more information on preview servers, see Setting Up Preview in the *ATG Business Control Center Administration and Development Guide*.

One-off deployments can be launched at any time from an ATG Content Administration project, regardless of that project's current workflow status. One-off deployments have no effect on that project's workflow or any target sites that are defined for that workflow. They do not lock project assets, so work on the project can continue unimpeded.

You designate a target site for one-off deployments when you define its site type (see Define the Target Site). In all other respects, you configure the site for ATG Content Administration deployment like any other target site. After you initialize the site, its designation as a one-off target cannot be changed.

Caution: A one-off target site definition must never map its source repositories to destination repositories that are used by a workflow target. One-off deployments to a workflow target prevent ATG Content Administration from managing content of that target, and can yield unpredictable results.

Constraints

The following constraints apply to one-off deployments, and differentiate them from workflow deployments:

- A one-off target site is not available for deployment assignments in a workflow. In order to make that site available for workflow deployments, you must delete the target and recreate it.
- A full deployment to a one-off target only includes assets from checked-in projects. Assets from deployed but still-active projects are excluded from the deployment.
- You cannot roll back deployment on a one-off target; the Admin Console's Projects tab for a one-off target site is read-only. If you perform a full deployment to a one-off target, the Projects list is emptied when deployment is complete.

11 Setting Up Deployment

You set up ATG Content Administration deployment in the following steps:

Note: This procedure assumes that ATG is installed on your staging and/or production target servers.

1. Plan deployment topology—that is, the target sites and agents that make up the environment where you deploy asset data.
2. Set up deployment agents.
3. Within each deployment target:
   - Configure the deployment mode for target servers: either configure switch deployment, or configure online deployment.
   - Manage asset security on target sites: adopt a strategy to protect the ATG Content Administration-managed assets on the target from user modification.
4. Configure the asset management server:
   - Configure deployment data sources and destination repositories used by the DAF deployment system.
   - Define the deployment topology for use by the DeploymentServer.
   - If your environment has several ATG Content Administration clusters, configure deployment from multiple asset management server clusters to the same site, if desired.
   - Initialize target sites.
5. If desired, on the asset management server and/or target servers, configure deployment event listeners that listen for deployment events and take appropriate action.
6. Schedule deletion of empty folders for folders whose assets moved to a renamed folder.
7. Optionally, cache checksums for file assets in order to optimize deployment of file assets. More generally, you can use other strategies to optimize frequent deployments, or deployments of very large numbers of assets. For more information, see *Configuring DAF Deployment for Performance* in the *ATG Programming Guide*.

Plan Deployment Topology

The DeploymentServer requires information about the production and staging targets where assets are to be deployed. In order to provide this information you define the deployment topology—that is, deployment targets and their individual servers where agents are installed. Before you do so, however, knowledge of the topology is required for several earlier steps in the deployment configuration process. For this reason, you should plan the deployment topology as the first step towards deployment setup.

You define a deployment topology in these steps:

1. Identify deployment target sites.
2. Identify deployment agents.
3. Plan deployment agent responsibilities—for example, determine whether a given agent deploys repository or file assets.

Identify Deployment Target Sites

A deployment target site is a logical grouping of one or more servers that serve the same content. When you deploy to a target site, every server in the target is updated to reflect the new set of assets.

If multiple data stores mirror each other, you can group them in a single target site in order to synchronize their content. Alternatively, you can put them in different targets, so one cluster remains functional in case the other has deployment problems. In either case, a workflow can specify only one target site per deployment. For example, if the workflow defines staging and production deployments, it can identify just one target site for each deployment.

Note: A repository where you deploy assets can have reference constraints to tables from another repository only if both repositories belong to the same target.

Identify Deployment Agents

A deployment topology requires you to identify the deployment agents that run on target servers. These agents include:

- **Publishing agents** that you install on the target's ATG servers. These agents perform deployment-related tasks such as cache management for repository and file assets on the ATG servers.
- **Publishing Web agents** that you install on the target's Web servers. These agents perform deployment-related tasks for Web content file assets on the Web servers.

A deployment agent can be included in only one target; you cannot include an agent in multiple targets.

For each agent, you must provide the following:

- A simple identifier or name
- The URI of the agent's RMI client

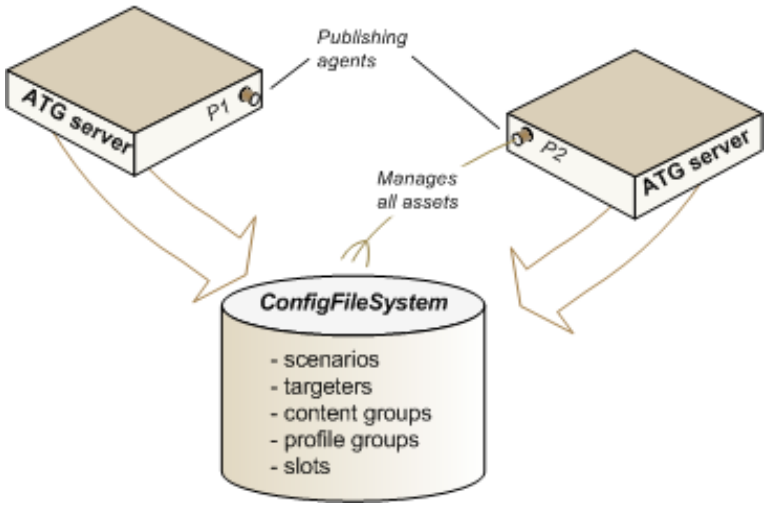
Plan Deployment Agent Responsibilities

A deployment topology requires you to determine which agents in a target are responsible for managing target assets. Only one agent should manage a shared data store. For each agent, determine which repository or type of asset that agent will manage. The two examples that follow demonstrate the deployment responsibilities to assign to agents in various target configurations:

- ATG servers share a single data store.
- ATG servers and Web servers use shared and local data stores.

ATG Servers Share a Single Data Store

The following target includes two ATG servers that share a single data store.



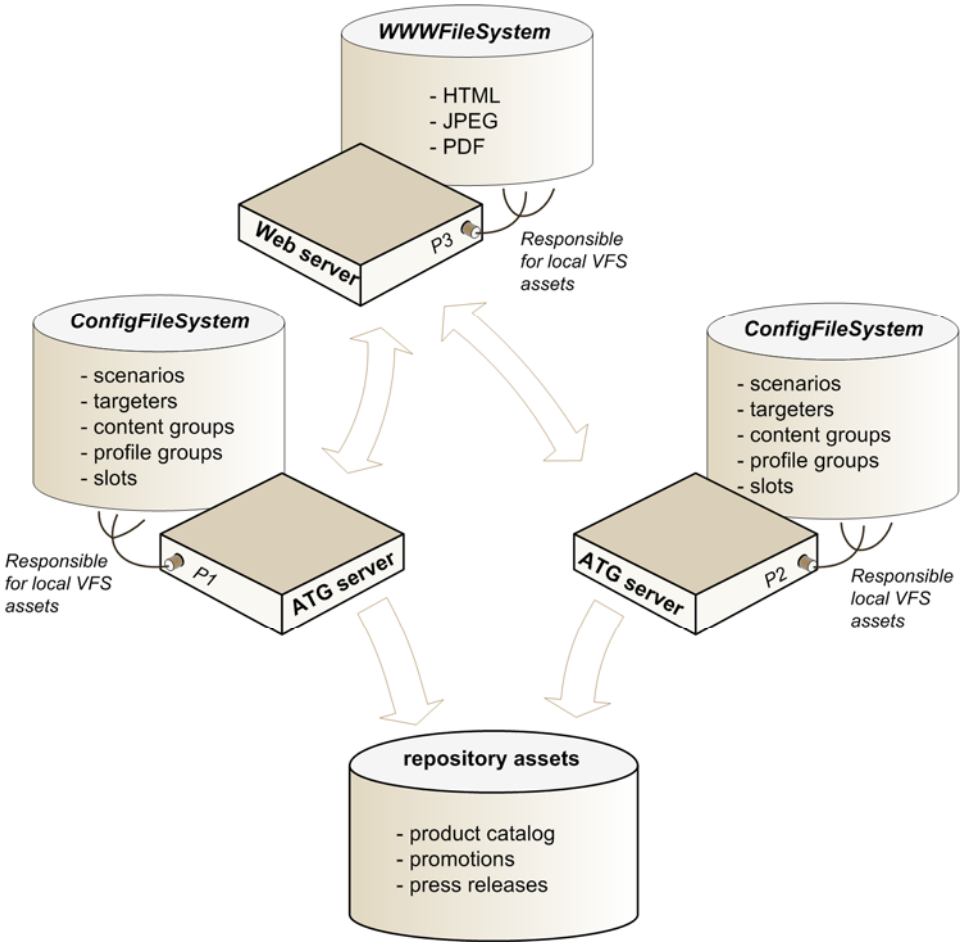
While both deployment agents share the data store, only one is given responsibility for managing all data store assets, by selecting all available destinations in the Admin Console deployment UI.

You can also assign deployment responsibilities by giving agent P2 responsibility for managing all repository assets. This is permissible because there are no VFS assets in the target.

If you expand this sample target to include a second cluster of ATG servers that uses a second data store, the responsibility for managing the second store must be given to an agent that uses it.

ATG Servers and Web servers Use Shared and Local Data Stores

This target includes two ATG servers that share a data store that stores repository assets, and a Web server where static content files are deployed from the asset management server. The Web server has its own local VFS that stores file assets. This local VFS is located in Nucleus at `/atg/epub/file/WWWFileSystem` and is installed with the Publishing Web agent.



The various deployment agents have the following deployment responsibilities:

- Publishing agents P1 and P2 manage their own local ConfigFileSystem VFSs.
- Web publishing agent P3 manages its own local WWWFileSystem. In the Admin Console deployment UI, specify `/atg/epub/file/ConfigFileSystem` as the included destination for deployment agents P1 and P2. Because both VFSs are local, no work is duplicated. For agent P3, specify `/atg/epub/file/WWWFileSystem` as the included destination.

Set Up Deployment Agents

The following sections show how to set up deployment agents:

- Installing the Publishing Web agent
- Changing the port used for file asset deployment
- Running deployment agents

Note: The Publishing agent is installed by default with the ATG platform, so installation steps for this agent are not included here.

Installing the Publishing Web Agent

In order to deploy Web content such as static text and binary files to staging and production Web servers, install and configure a Publishing Web agent on each Web server.

Note: You use the ATG Web Server Extensions installer to install the Publishing Web agent. You can also use it to install a `DistributorServer` for a Content Distributor system. For information on using the installer to do so, as well as general information on Content Distributor systems, see the *Content Distribution* chapter in the *ATG Programming Guide*.

Installing on Windows

To install the Publishing Web agent on a Web server running on Windows:

1. Make sure the Web server machine has a Java Virtual Machine installed.
2. Download the ATG Web Server Extensions executable file from the ATG Web site, `www.atg.com`.
3. Run the ATG Web Server Extensions executable.

   The installer displays the Welcome dialog box.
4. Click Next
5. Specify the installation directory for the Publishing Web agent. The default (and recommended) directory is `c:\ATG\ATGWeb10.0.1`
6. Click Next

   The installer displays the list of Web server extensions you can configure during the installation process. By default, the `DistributorServer` and the Publishing Web agent are both selected. You can uncheck the `DistributorServer` option for this installation.

   For information on configuring the `DistributorServer` for a Content Distributor system, see the *Content Distribution* chapter in the *ATG Programming Guide*.
7. Click Next
8. Specify the RMI port to be used by the Publishing Web agent. The default is 8860. If this port is already taken—for example, by the PublishingAgent of an ATG production server running on the same machine—you must reset this to a unique value. You can reset `RMIPort` in one of the following properties files:

   - `\config\atg\dynamo\server\RmiServer.properties`
   - `\home\localconfig\atg\dynamo\server\RmiServer.properties`

   Note: Later, when you define the deployment topology, you specify the same port number for the Web agent in the Admin Console deployment UI.
9. Click Next

10. Specify the local directory to be used by the Publishing Web agent. The default is `C:\ATG\ATGWeb10.0.1\home\doc`. You can specify the Web server's document root directory or any subdirectory within the root directory.
11. Click Next
12. Specify a name for the program folder that the installer will create within the Windows Programs folder. The default is `ATG 10.0.1\ATG Web Server Extensions`. The program folder will contain a shortcut for starting the Publishing Web agent.
13. Click Next

    The installer displays the installation directory you selected and indicates it is ready to install the Publishing Web agent.
14. Review the setup information:
    - Click Next to start the installation
    - Click Back to change any settings
15. After installation is complete, click Finish to close the installer.

Installing on UNIX

To install the Publishing Web agent on a Web server running on UNIX:

1. Make sure the Web server machine has a Java Virtual Machine installed.
2. Download from the ATG website `www.atg.com` the ATG Web Server Extensions executable.
3. Make sure you have execute permissions with this command or its equivalent:

   `chmod +x` *`filename`*
4. Run the installation program.
5. When prompted, enter the local directory to be used by the Publishing Web agent.

   You can specify the Web server's document root directory or any subdirectory within the root directory.
6. The installer displays the Web server extensions you can configure during the installation process: the `DistributorServer` and the Publishing Web agent. By default, both selected. You can deselect the `DistributorServer` option for this installation.

   For information about the `DistributorServer` and the Content Distributor system, see the *Content Distribution* chapter in the *ATG Programming Guide*.
7. Specify the RMI port to be used by the Publishing Web agent. The default is 8860. If this port is already taken—for example, by the PublishingAgent of an ATG production server that runs on the same machine—you must reset this to a unique value. You can reset a PublishingWebAgent's `RmiServer.RMIPort` property in one of the following locations:
   - `/config/atg/dynamo/server/RmiServer.properties`
   - `/home/localconfig/atg/dynamo/server/RmiServer.properties`

Note: Later, when you define the deployment topology, you specify the same port number for the Web agent in the Admin Console deployment UI.

Changing the Port Used for File Asset Deployment

By default, file asset deployment uses port 8810 to send assets from the asset management server to the target. Depending on your environment, you might need to open this port in the target server's firewall. Alternatively, you can change the port that the connection uses by editing the `port` property in the agent's `/atg/deployment/file/FileDeploymentServer`. If multiple agents run on the same VM, each one's `FileDeploymentServer` port must be unique. No corresponding change is required on the asset management server, which detects the port automatically through the RMI connection.

Running Deployment Agents

After setting up target site deployment agents, you can start them as described in the following sections:

- Running the Publishing agent
- Running the Publishing Web agent

Caution: Do not use any ATG Content Administration modules or the `DAF.Deployment` module with a standalone GSA lock manager server. Doing so causes the lock server to become a slave server of the deployment. It attempts to deploy files, and the deployment deadlocks as a result. Symptoms include the deployment cycling in a time out loop.

Running the Publishing Agent

To activate the Publishing agent on an ATG server, assemble an application that includes the following application module:

```
PublishingAgent
```

You also use non-versioned modules with your ATG Content Administration application on the target site. After assembling the application, deploy it to the appropriate server. For information on ATG modules and application assembly, see the *ATG Programming Guide*.

Running the Publishing Web Agent

To use the Publishing Web agent with a Web server:

1. Make sure your `JAVA_HOME` environment variable is set.
2. Change to the `/ATGWeb10.0.1/home` directory.
3. Start the Publishing Web agent with one of the following commands:

   Windows: `bin\startNucleus -m PublishingWebAgent`

   UNIX: `bin/startNucleus -m PublishingWebAgent`

If you also have a configured `DistributorServer` on the Web server (as part of a Content Distributor system), you can start up the `Distributor` and `PublishingWebAgent` modules at the same time, as in the following example:

```
bin/startNucleus -m PublishingWebAgent:Distributor
```

For information on Content Distributor systems, see the *Content Distribution* chapter in the *ATG Programming Guide*.

Configure Switch Deployment

Configuring a target site for deployments in switch mode involves the following general steps:

1. Configure target repositories for switch deployments.
2. Configure default target VFSs for switch deployments.
3. If necessary, configure custom target VFSs for switch deployments.
4. Configure switch deployment on the asset management server.

Optimization Options

The following options are available to optimize switch deployment:

- Selective Cache Invalidation. Retaining still-valid items in repository caches can improve a site's post-deployment performance.
- Configure background deletion of file system assets.
- Optionally, configure a shared ConfigFileSystem for multiple agents. This lets you deploy to a single VFS rather than to VFSs on multiple agents, which can improve deployment time.

Configure Target Repositories for Switch Deployments

Deployments performed in switch mode require two databases, where repository assets are updated on the target site's inactive database while its live database runs undisturbed. Both databases must begin with identical content. You can ensure this in one of two ways:

- Use tools provided by your database vendor to make the two databases identical.
- After you configure the target for switch deployments, perform a full deployment to the target.

Definition files of target site repositories must not contain operation tags such as `<add-item>`, `<update-item>`, and `<remove-item>`. These tags cause switch deployments to fail.

Perform the following steps on the target site ATG servers:

1. Configure data sources that connect to the two databases. Each data source must be of class `atg.nucleus.JNDIReference`, where its `JNDIName` property points to an application server data source. For detailed information about configuring data sources, see the *ATG Installation and Configuration Guide*.
2. Configure a SwitchingDataSource to switch between the two underlying data sources.

3. Configure destination and production-site repository components so their `datasource` property points to the appropriate SwitchingDataSource.
4. Optionally, configure selective cache invalidation on target repositories.
5. Add the SwitchingDataSource to the DeploymentAgent's list of switchable data stores. To do this, set the property `switchableDataStores` to the SwitchingDataSource's fully qualified Nucleus path, in:

   `<ATG10dir>/home/localconfig/atg/epub/DeploymentAgent.properties`

Configure a SwitchingDataSource

Switch deployment uses a SwitchingDataSource (`atg.service.jdbc.SwitchingDataSource`) instead of a regular data source such as `atg.service.jdbc.MonitoredDataSource`. During deployment, it typically switches between two data sources. `DataSource` method calls are passed through to the data source that is specified by the SwitchingDataSource's `currentDataSource` property.

Caution: If you have multiple independent ATG clusters that share a single `SDSRepository`, each cluster must use a unique set of `SwitchingDataSource` names. Otherwise, the clusters interfere with each other during the switching process.

Set the following properties of the SwitchingDataSource component:

Name	Description

`dataSources`	A `ServiceMap` of `DataSources` that maps the short names of data sources to their Nucleus component paths. For example:
	
	`dataSources=\`
	`DataSource1=/atg/dynamo/service/jdbc/FirstDataSource,\`
	`DataSource2=/atg/dynamo/service/jdbc/SecondDataSource`

`initialDataSourceName`	The short name of the data source to use as the `currentDataSource` on the first deployment. On subsequent runs, the initial `currentDataSource` is obtained from the state recorded in the `SDSRepository`.
`repository`	Set with a reference to `/atg/dynamo/service/jdbc/SDSRepository`. Specifies the switching data source repository `SDSRepository`, which monitors which database the switching data source points to at any time.

The following example shows the default settings of the switching datasource used by the ATG Commerce product catalog:

```
$class=atg.service.jdbc.SwitchingDataSource
#
# A map from data source names to data sources
#
dataSources=\
     DataSourceA=/atg/commerce/jdbc/ProductCatalogDataSourceA,\
     DataSourceB=/atg/commerce/jdbc/ProductCatalogDataSourceB

#
# The name of the data source that should be used on startup
#
initialDataSourceName=DataSourceA

repository=/atg/dynamo/service/jdbc/SDSRepository
```

Configure Default Target VFSs for Switch Deployments

The ATG distribution provides two VFSs that can be configured for switch deployments:

- `/atg/epub/file/ConfigFileSystem`
- `/atg/epub/file/WWWFileSystem`

ConfigFileSystem

The ATG distribution provides ConfigFileSystem with the Publishing agent. As installed, ConfigFileSystem is configured for online deployments. In order to deploy personalization and scenario assets to the target, you must configure ConfigFileSystem on each ATG server for switch deployments.

To configure each ConfigFileSystem VFS for switch deployments, perform these steps for each target ATG server:

1. Create an instance of `atg.vfs.switchable.SwitchableLocalFileSystem`. The installation provides a `ConfigFileSystem.properties` file that you can copy from this location.

   `<ATG10dir>/PublishingAgent/config/atg/epub/file/`

   Uncomment the settings that are specific to switch deployment, and modify the properties where necessary. For more information about these properties, see `SwitchableLocalFileSystem` in Appendix B: Virtual File Systems.

2. Place the new `ConfigFileSystem.properties` file in each server's `localconfig` directory:

   `../localconfig/atg/epub/file/`

3. For each server, add ConfigFileSystem to the list of switchable data stores configured in the DeploymentAgent:

`../localconfig/atg/epub/DeploymentAgent.properties`

Set `switchableDataStores` to the ConfigFileSystem's fully qualified Nucleus path:

`/atg/epub/file/ConfigFileSystem`

WWWFileSystem

The ATG distribution provides WWWFileSystem with the Publishing Web agent. As installed, it is configured for online deployments. If you install the Publishing Web agent on the target Web servers for deployment of deploy static text and binary files, you can configure the WWWFileSystem on each Web server for switch deployments.

To configure each WWWFileSystem VFS for switch deployments, perform these steps for each target Web server:

1. Create an instance of `atg.vfs.switchable.SwitchableLocalFileSystem`. The installation provides a `WWWFileSystem.properties` file that you can copy from this location.

   `<ATG10dir>/PublishingWebAgent/config/atg/epub/file/`

   Uncomment the settings that are specific to switch deployment, and modify the properties where necessary. For more information about these properties, see `SwitchableLocalFileSystem` in Appendix B: Virtual File Systems.

2. Place the new `WWWFileSystem.properties` file in each server's `localconfig` directory:

   `../localconfig/atg/epub/file/`

3. For each server, add WWWFileSystem to the list of switchable data stores configured in the DeploymentAgent:

   `../localconfig/atg/epub/DeploymentAgent.properties`

   Specify WWWFileSystem by its fully qualified Nucleus path:

   `/atg/epub/file/WWWFileSystem`

Configure VFSs on a New ATG server for Switch Deployment

When adding an ATG server or Web server to an existing target, configure its switchable VFSs like those on the existing ATG target servers. Verify that:

- It has the same underlying data sources.
- Its initial data store, specified in the `name1` property of the VFS, is the same as their current data store, so the new server's agent application starts up using the same data store.

Configure Custom Target VFSs for Switch Deployments

To create and configure a custom VFS for switch deployments, perform these steps on the target server:

1. Install the Publishing Web agent on the target server.
2. Create and configure the switchable VFS.

3. Configure the DeploymentAgent with the custom VFS.

Install the Publishing Web agent on the target server

If the custom VFS is not located on an ATG server, you must install the Publishing Web agent on the target server in order to enable deployment from the asset management server. For more information, see Installing the Publishing Web agent.

Note: The Publishing Web agent is configured with the VFS WWWFileSystem, as a convenience for users whose sites require that static content be deployed directly to the Web server, where it is served quickly. However, you can install the Publishing Web agent on any non-ATG server where you want to deploy files, whether or not that server is a Web server. If you do not require the WWWFileSystem on the server, either remove its configuration file or exclude it from your deployment topology (see Plan Deployment Agent Responsibilities).

Create and configure the switchable VFS

Configure a `SwitchableLocalFileSystem` instance and place the configuration file in each server's `localconfig` directory:

```
../localconfig/atg/epub/file/
```

You can use the `WWWFileSystem.properties` file that is installed with the Publishing Web agent as the template for your custom VFS. If your site does not use WWWFileSystem, you can create the custom VFS by renaming the `WWWFileSystem.properties` file and modifying its class and properties.

For example, the following properties file configures a switchable VFS `/mycompany/FTPFileSystem` that stores files on an FTP server. (This example continues the example used in Configure Support for Other File Assets.)

```
$class=atg.vfs.switchable.SwitchableLocalFileSystem

liveDirectory={atg.dynamo.server.home}/MyConfig/ftpFiles/live
stagingDirectory={atg.dynamo.server.home}/MyConfig/ftpFiles/staging
dataDirectory={atg.dynamo.server.home}/MyConfig/ftpFiles/data

name1=firstDataStore
name2=secondDataStore
```

For detailed information on `SwitchableLocalFileSystem` properties, see *Appendix B: Virtual File Systems*.

Configure the DeploymentAgent with the custom VFS

Add the custom VFS to the DeploymentAgent's list of switchable data stores by setting the agent's `switchableDataStores` property in:

```
../localconfig/atg/epub/DeploymentAgent.properties
```

Be sure to specify ConfigFileSystem by its fully qualified Nucleus path.

Configure VFSs on a New ATG Server for Switch Deployment

When adding an ATG server or Web server to an existing target, configure its switchable VFSs exactly like those on the existing target servers. Verify that:

- It has the same underlying data stores.
- Its initial data store, specified in the `name1` property of the VFS, is the same as their current data store, so the new server's agent application starts up using the same data store.

Configure Switch Deployment on the Asset Management Server

Follow these steps to set up switch deployment on the asset management server.

1. Set up the appropriate Nucleus components for a SwitchingDataSource. SwitchingDataSources must be configured on the asset management server for each production-site destination repository and must point to the target databases. See the diagram in Destination Repositories, which shows a switch deployment configuration.
2. In the Nucleus component that configures `atg.service.jdbc.SwitchingDataSource`, configure the `dataSources` property so it uses the same data source names configured on the agents in the appropriate target.
3. Set the property `SwitchingDataSource.initialDataSourceName` to the data source that is **not** the one specified in the `initialDataSourceName` on the target servers. This behavior ensures deployment of the inactive database on the target servers.

For example, the properties file `/atg/dynamo/service/jdbc/SwitchingDataSource.properties` for the asset management server might look like this:

```
$class=atg.service.jdbc.SwitchingDataSource
dataSources=\
  FirstDataSource=/atg/dynamo/service/jdbc/FirstDataSource,\
  SecondDataSource=/atg/dynamo/service/jdbc/SecondDataSource
initialDataSourceName=SecondDataSource
```

Configuring a Multi-Server Cluster

In order to enable switch deployment from a multi-server cluster, each server must be configured with contact data about the other servers in the cluster. For more information, see Configure a Cluster for Switch Deployment.

Selective Cache Invalidation

You can optimize a site's performance after a switch deployment by enabling selective cache invalidation on target repositories. A repository that is thus configured invalidates its item caches selectively during deployment, rather than invalidating the contents of those caches entirely.

If enabled:

Only those cached items that change as a result of the deployment are invalidated. All other cached items remain unchanged. Because selective invalidation increases deployment overhead, you might want to configure a threshold for the number of invalidated items. On exceeding that threshold—the sum of all changed items in a deployment—the selective invalidation process is skipped and the deployment invalidates all cached items in the target repositories.

If not enabled:

Item caches of all deployed repositories are invalidated. This can result in slow response time to initial requests, as fresh data is obtained directly from the database.

Constraints

Two general constraints apply to usage of selective cache invalidation:

- Selective cache invalidation only applies to item descriptors that use simple caching mode.
- The `atg.repository.RepositoryImpl` method `invalidateCaches()` clears all caches from the target repository, even if selective cache invalidation is enabled for that repository.

Configuration Steps

You configure selective cache invalidation on the production server and the asset management server, as follows:

- On each production site repository, set the GSARepository property `selectiveCacheInvalidationEnabled` to `true`. By default, this property is set to `false`.

  Note: You can use the `liveconfig` configuration layer to enable selective cache invalidation on desired repositories. As installed, the `liveconfig` configuration layer enables selective cache invalidation on certain ATG repositories such as `productCatalog`.

- On the asset management server, configure the item invalidation threshold by setting the `threshold` property to a positive integer in this component:

  `/atg/epub/sci/ServerSCIThresholdController`

  The default setting of `-1` allows an unlimited number of item invalidations.

Excluding Repositories and Item Descriptors

A production site publishing agent can be configured to exclude specific repositories and item descriptors from selective cache invalidation—in other words, require that item caches be fully invalidated on each deployment. To do so, set the property `fullInvalidationRepositoryPaths` in this component:

`/atg/epub/sci/AgentSCIThresholdController`

You set this property as follows:

```
fullInvalidationRepositoryPaths=\
   repository-path[=item-descriptor[;item-descriptor]...] \
   [,...]
```

- *repository-path* is the path to a repository.
- *item-descriptor* specifies which item caches in that repository to invalidate on each deployment.

If no item descriptors are supplied, all item caches in the repository are invalidated. The property can specify multiple comma-delimited repositories, and each repository can specify multiple semi-colon-delimited item descriptors.

For example:

```
fullInvalidationRepositoryPaths=\
   /atg/commerce/catalog/ProductCatalog=category;product
```

Background Deletion of File System Assets

During a full switch deployment, the `SwitchableLocalFileSystem` must delete all file assets from the target site's inactive directory before updated content can be deployed from ATG Content Administration. On a large site, the deletion process can significantly delay updates to the live site. You can optimize deployment by configuring the `SwitchableLocalFileSystem` to enable background deletion: when switch deployment begins, the VFS simply moves the current inactive directory to a temporary directory and proceeds with deployment. Separately, it launches another thread that is responsible for physically removing files from the temporary directory—which it typically defers until system resources are more plentiful.

Note: By default, full deployment of file system assets is optimized through checksum verification, which minimizes disk reads and writes (see Cache Checksums for File Assets). This is usually faster than full deployment with background deletion, where checksum verification is disabled (see Requirements and Constraints), so all files eventually must be physically removed from disk. In general, full deployment with background deletion is appropriate when you must purge a target site of all file system assets.

Properties

Three SwitchableLocalFileSystem properties enable background deletion and control its behavior:

- `deleteInBackground` enables background deletion if set to true. By default, this property is set to false.
- `deletionThreadPriorityDelta` lets you manually set the deletion thread's priority. In general, you should let the system set thread priority, and omit this property.
- `deletionThreadDelay` specifies in milliseconds how long the deletion thread waits before it begins to delete files from the temporary directory. If set to 0 (the default), deletion begins immediately.

Requirements and Constraints

The following requirements and constraints apply to background deletion of file system assets:

- Background deletion only occurs during full deployments.
- Checksum verification must be disabled, by setting the following file deployment property in the component `/atg/deployment/file/DeploymentConfiguration`:

  `noVerificationByChecksum=true`
- `ownerCacheEnabled` must be set to false in the VFS configuration of the production and staging servers (see Enable tracking of file ownership). As a result, you cannot deploy from multiple asset management servers.
- Background deletion prevents components from restarting properly, so it should not be used on the ConfigFileSystem.

Shared ConfigFileSystem for Multiple Agents

Optionally, you can configure an ATG Content Administration environment distributed across multiple VMs so it deploys to a single ConfigFileSystem. To do so, you create a master ConfigFileSystem on one VM and slave ConfigFileSystems on the others, so only one VM on a given machine is responsible for file deployments. This configuration can significantly increase performance because deployment data, which can be very large, is sent to one agent rather than all agents.

Note: A shared ConfigFileSystem is supported for switch mode deployments only.

To set up a shared ConfigFileSystem deployment model:

1. Designate one ATG instance on each machine as the master instance. Other ATG instances are considered slave targets and use the ConfigFileSystem on the master instance.
2. From the class `atg.vfs.switchable.SwitchableLocalFileSystem`, configure a master `ConfigFileSystem.properties` file:

   `../localconfig/atg/epub/file/ConfigFileSystem.properties`
3. From the class `atg.vfs.switchable.SlaveLocalFileSystemService`, configure identical slave `ConfigFileSystem.properties` files:

   `../localconfig/atg/epub/file/ConfigFileSystem.properties`
4. Configure the master ConfigFileSystem as a switchable data store:
   - On the master instance:
     `../localconfig/atg/epub/DeploymentAgent.properties`
   - Set the property `switchableDataStores`:

     `switchableDataStores+=/atg/epub/file/ConfigFileSystem`
5. Perform a full deployment to synchronize the new data directories with the asset management server.

Master ConfigFileSystem.properties

The following sample shows a `ConfigFileSystem.properties` file on the master instance:

Note: Paths are machine-specific.

```
#$class=atg.vfs.journal.JournalingFileSystemService
#virtualFileSystem=ConfigFileSystemLocal
#journalDirectory={serverHomeDirResource?resourceURI=PublishingAgent/
   deploymentconfig/data/config}
#clearJournalOnUpdate=true
#updateListeners+=/atg/epub/monitor/PersonalizationConfigurationMonitor

################################
#Use this configuration for a switch deployment

$class=atg.vfs.switchable.SwitchableLocalFileSystem
#
liveDirectory={serverHomeDirResource?resourceURI=PublishingAgent/
   deploymentconfig/live/config}
stagingDirectory={serverHomeDirResource?resourceURI=PublishingAgent/
   deploymentconfig/staging/config}
dataDirectory={serverHomeDirResource?resourceURI=PublishingAgent/
   deploymentconfig/data/config}
journaling=true
clearJournalOnUpdate=true
backupJournal=true
name1=VFSName1
name2=VFSName2
```

Slave ConfigFileSystem.properties

The following sample shows a `ConfigFileSystem.properties` file for slave instances:

Note: Paths are machine-specific.

```
$class=atg.vfs.switchable.SlaveLocalFileSystemService

# references to SwitchableLocalFileSystem
rootDirectory=c:\\atg\\atg10.0.1\\home\\servers\\target-
primary\\PublishingAgent\\deploymentconfig\\live\\config

journalDirectory=c:\\atg\\atg10.0.1\\home\\servers\\target-
primary\\PublishingAgent\\deploymentconfig\\data\\config\\journalBackup

statusFile=c:\\atg\\atg10.0.1\\home\\servers\\target-
primary\\PublishingAgent\\deploymentconfig\\data\\config\\status.dat
```

```
# schedule for checking switch status to see if an update event is required

scheduler=/atg/dynamo/service/Scheduler
schedule=every 20 seconds
```

Adding an ATG Server

When adding an ATG server to an existing target, configure its repositories and SwitchingDataSource like those on the existing ATG target servers. Verify that:

- It has the same underlying data sources.
- Its initial data source is the same as their current data source, so the new server's agent application starts up using the same data source.

Configure Online Deployment

In order to configure a target site for online deployment, you must:

- Configure repositories for online deployments.
- Configure custom VFSs for online deployments.

The ConfigFileSystem that is provided with the Publishing agent is configured by default for online deployment; it is a configured instance of class `atg.vfs.journal.JournalingFileSystemService`, a VFS implementation that provides the journaling functionality required to notify the appropriate services of personalization and scenario updates.

The WWWFileSystem that is provided with the Publishing Web agent is also configured out-of-the-box for online deployment; it is a configured instance of class `atg.service.vfs.LocalVFSService`.

Configure Repositories for Online Deployments

Configuring target site repositories for online deployment requires no special ATG Content Administration-specific configuration changes. Simply configure the SQL repository component as described in the *ATG Repository Guide*.

Configure Custom VFSs for Online Deployments

To create and configure a custom VFS for online deployments, perform this procedure on the target server:

1. If the custom VFS is not on an ATG server, you must install the Publishing Web agent on the target server, to enable communication with the asset management server during deployment, (see Installing the Publishing Web Agent).

   Note: The Publishing Web agent is configured with an out-of-the-box VFS, the WWWFileSystem, as a convenience for users whose sites require that static content be

deployed directly to the Web server, where it is served quickly. However, you can install the Publishing Web agent on any non-ATG server where you want to deploy files, whether or not that server is a Web server. In this case, if you do not require the WWWFileSystem on the server, either remove its configuration file or exclude it from your deployment topology (see Plan Deployment Agent Responsibilities).

2. Create the local VFS by configuring a `LocalVFSService` instance and placing the configuration file in your `localconfig` directory.

   Note: You can use the `WWWFileSystem.properties` file that is installed with the Publishing Web agent as the template for your custom VFS. If you do not require the WWWFileSystem on the given server, you can create a custom VFS simply by renaming the `WWWFileSystem.properties` file and modifying its `localDirectory` property.

   For example, the following is the configuration file for a local VFS `/mycompany/FTPFileSystem` that stores files on an FTP server. (This continues the example used in the Configure Support for Other File Assets section of the earlier chapter Setting Up an Asset Management Server.)

   ```
   $class=atg.service.vfs.LocalVFSService
   localDirectory={atg.dynamo.server.home}/MyConfig/ftpFiles
   ```

   In the `localDirectory` property, specify the directory to which the files should be deployed from the asset management server. Also make sure the directory does not store files used by other services; it should contain only files deployed from the corresponding custom VFS on the asset management server.

Manage Asset Security on Target Sites

Assets that are created and managed in the content development environment should not be modified directly on the target site. Therefore, it is important to secure target site assets to prevent changes that are outside the control of the content development environment.

Preventing uncontrolled changes is especially important with respect to ATG Content Administration deployments:

- The changes that users make on the target site are liable to be overwritten by subsequent deployments. In the case of a full deployment, all changes are always overwritten, inasmuch as the deployment begins by deleting all target assets.
- Incremental deployments assume that the set of assets that is active on a target actually represents the assets on that target.

In an incremental deployment, ATG Content Administration identifies the current set of assets on the target, examines it against the project to deploy, and deploys the new set by deploying only the asset changes to the target site. It is this element of the incremental deployment procedure that makes it faster than a full deployment.

However, if a user modifies the assets on the target directly, the target's knowledge of its current set of assets is no longer valid because it no longer represents the target's actual data. Therefore, subsequent incremental deployments are compromised and cannot result in a new current set of assets that

accurately represents the target's data. In this situation, a full deployment to the target site which first deletes all target assets is necessary in order to restore a valid set of assets.

Note: If you use secured repositories in the content development environment to control access of ATG Content Administration users to specific asset types and individual assets, be aware that ATG Content Administration does not deploy the ACLs for those assets when they are deployed to a target.

For all these reasons, it is critical that you secure ATG Content Administration-managed assets on your staging and production targets so users cannot modify them. Several recommended strategies follow, which are suitable to various content development, testing, and production requirements.

Modifying User Access Privileges in the ACC

If you require access to the ACC in the target environment, you should do one of the following:

- If you use secured repositories in the target environment, manually modify their definition files so the appropriate ACC user groups are restricted to List and Read access to repository items. Users who belong to those groups can only view the items—for example, for content validation purposes. For information on managing secured repository definition files, see the *ATG Repository Guide*.
- If you do not use secured repositories in the target environment, and you do not require access to them, simply remove UI access privileges to those repositories from the appropriate ACC user groups. For information on modifying the UI access privileges for ACC user groups, see the *ATG Programming Guide*.

  Alternatively, if you require access to the repositories, configure secured repositories to sit on top of your repositories and restrict the appropriate ACC user groups to List and Read access to the repository items, as described in the *ATG Repository Guide*.

Restricting Access to Personalization and Scenario Assets

If you manage personalization and/or scenario assets with ATG Content Administration, do one of the following:

- If you do not require access to the assets in the target environment, disable access to the Targeting and/or Scenarios task areas in the ACC by removing those UI access privileges from the appropriate ACC user groups. For information on how to do this, see the discussion on managing access control in the *ATG Programming Guide*.
- If you do require access to the assets in the target environment, prevent users from modifying scenarios by granting only List and Read access to all scenario folders to the appropriate ACC user groups. This ensures that all users who belong to those groups can view (but not add to or edit) the folders and their contents. For information on how to do this, see the discussion on setting up security for scenarios in the *ATG Personalization Programming Guide*.

Note: This functionality is not available for other ACC-editable personalization and scenario assets that you can manage with ATG Content Administration, namely, targeters, content and profile groups, and slots.

Configure Deployment Data Sources and Destination Repositories

For deployment, the asset management server must have a data source configured to point to each target server's database. This section describes how to configure the data source and create destination repositories, which are repositories on the asset management server that point to target site databases. For more about the relationship between data source and the destination repositories, see Destination Repositories.

Before you start this configuration, determine the number of repositories on each deployment target. The example that follows assumes two targets: staging and production. Each target has two repositories, `productCatalog` and `priceLists`, which correspond to two versioned repositories on the asset management server. Thus configured, the environment requires ten repositories, as shown in the following table.

Repository name	Type	Server
`productCatalog`	versioned	ATG Content Administration
`priceLists`	versioned	ATG Content Administration
`productCatalog_staging`	destination	ATG Content Administration
`priceLists_staging`	destination	ATG Content Administration
`productCatalog_production`	destination	ATG Content Administration
`priceLists_production`	destination	ATG Content Administration
`productCatalog`	target	staging
`priceLists`	target	staging
`productCatalog`	target	production
`priceLists`	target	production

Given this configuration, complete the following steps:

1. Create a destination repository data source.
2. Create and configure a destination repository.
3. Update the Destination Repositories list.

Create a Destination Repository Data Source

For each destination repository, create a data source to point to the target database:

1. On the asset management server, copy and rename `JTDataSource.properties`—for example, `JTDataSource_staging.properties`.

The copy must be in the same location as the original files.

Alternatively, extract and use the following distribution files:

- `JTDataSource_staging.properties`, in:
  `<ATG10dir>\DafEar\base\configlayers\stagingandprod\config.jar`
- `JTDataSource_production.properties`, in:
  `<ATG10dir>\DAS\config\config.jar`

2. Point the new `JTDataSource` properties file to the target site's database. For details, see *Configuring Databases and Database Access* in the *ATG Installation and Configuration Guide*.

Note: Configuration of the destination repository's datasource might require special handling if the target site database is set up as an Oracle RAC cluster with multiple nodes. For more information, see the *ATG Installation and Configuration Guide*.

Create and Configure a Destination Repository

For each target repository, you must create a destination GSARepository. For example, given two targets, staging and production, and two repositories on each target, ProductCatalog and PriceLists, four destination repositories are required:, two for staging and two for production, as configured by these properties files:

```
/atg/commerce/catalog/ProductCatalog_staging.properties
/atg/commerce/pricing/priceLists/PriceLists_staging.properties

/atg/commerce/catalog/ProductCatalog_production.properties
/atg/commerce/pricing/priceLists/PriceLists_production.properties
```

Note: The distribution for ATG Merchandising provides ProductCatalog and PriceLists repositories and their corresponding destination repositories, for production-only and production/staging deployments. If you create your own repositories, you must also create their destination repositories.

Foreign Repository Mappings

Items in one destination repository can link to items in another through their `repository` attribute. Because the destination repositories are renamed copies of the corresponding production repositories, you must provide a way to resolve external references. For example, items in the destination repository `ProductCatalog_production` might reference items in the original repository `PriceList`, as follows:

```
<property name="priceList"
     column-name="price_list"
     item-type="priceList"
     repository="/atg/myApp/PriceLists" >
```

To ensure that cross-references resolve correctly, you can create a RepositoryMapper component from the class `atg.repository.RepositoryMapper`, which extends the `GenericService` class. This component's `RepositoryMappings` property provides the mappings that are required by a foreign repository, as follows:

```
repositoryMappings=\
   /Nucleus-path/original-repository=/Nucleus-path/new-repository,\
   /Nucleus-path/original-repository=/Nucleus-path/new-repository,\
   ...
```

where *Nucleus-path* must be the repository's absolute Nucleus path.

Thus, given the previous example, you set the `RepositoryMappings` property for a RepositoryMapper as follows:

```
repositoryMappings=\
   /atg/myApp/ProductCatalog=/atg/myApp/ProductCatalog_production,\
   /atg/myApp/PriceLists=/atg/myApp/PriceLists_production
```

Each repository that requires mapping for its items must set a RepositoryMapper in its `foreignRepositoryMapper` property. Given the previous example, `ProductCatalog_production.properties` and `PriceLists_production.properties` must set their `foreignRepositoryMapper` property to a RepositoryMapper as follows:

`foreignRepositoryMapper=`*`/Nucleus-path/repository-mapper`*

Note: The ATG platform distribution provides a RepositoryMapper that contains required mappings; you can add your own mappings to these by creating, in the management server's `localconfig` layer, `/atg/repository/ProductionRepositoryMapper.properties`, and setting `repositoryMappings` with the increment/assignment operator += as follows:

```
repositoryMappings+=mapping[,]...
```

Procedure

To create a destination repository, perform these steps:

1. Copy an existing unversioned repository properties file into the `localconfig` directory of the asset management server and rename it.
2. In the properties file:
   - Set the value of the `repositoryName` property to the name of the destination repository.
   - Set the `dataSource` property to point to this repository's data source. Each destination repository must be set to a data source component that is specific to the destination. For example, all destination repositories for the production server might set their `dataSource` property to `/atg/dynamo/service/jdbc/JTDataSource_production`, while all destination repositories for the staging server might specify `/atg/dynamo/service/jdbc/JTDataSource_staging`.

- Set the `lockManager` property to a ClientLockManager component. This ClientLockManager is defined to communicate with the corresponding ServerLockManager on the production server. While the ClientLockManager component is configured on the asset management server, it is logically part of the production server cluster and must be configured against that server's components.

  For information about configuring a ClientLockManager, refer to the SQL Repository Caching chapter in the *ATG Repository Guide*.

- Set `foreignRepositoryMapper` to the RepositoryMapper that you use to map destination repository names, as described earlier.

For example:

```
repositoryName=ProductCatalog_production
dataSource=/atg/dynamo/service/jdbc/JTDataSource_production
lockManager=/atg/dynamo/service/ClientLockManager_production
foreignRepositoryMapper=/atg/repository/ProductionRepositoryMapper
```

Update the Destination Repositories List

On the asset management server, add the destination repositories to the property `additionalAssetSources`, in `/localconfig/atg/dynamo/service/AssetResolver.properties`. This updates the Destination Repositories list in the Admin Console deployment UI, so it displays the new repositories.

For example:

```
additionalAssetSources+=\
  /atg/myApp/ProductCatalog_staging,\
  /atg/myApp/PriceLists_staging,\
  /atg/myApp/ProductCatalog_production,\
  /atg/myApp/PriceLists_production
```

Define the Deployment Topology

After you plan deployment topology, you can implement it through the Admin Console in the ATG Business Control Center. The Admin Console lets you set up the target sites, agents, and agent responsibilities that define an ATG Content Administration deployment topology:

1. Assemble a Web application that includes asset management server according to the instructions in the *ATG Programming Guide*. Deploy the application to the appropriate server.
2. Access the ATG Business Control Center.
3. Log in as a user with access rights to the Admin Console (see Accessing the Admin Console).

4. In the Home page, under ATG Content Administration click Admin Console.
5. From the Deployment Administration page, click Configuration.
6. From the Configuration page, click Add Site. You then configure the new site on its Details and Agents tabs, where you:
    - Define the target site
    - Configure target site deployment agents

Note: You can also define a deployment topology by editing deploymentTopology.xml, as described later in this section.

Define the Target Site

You define a target site on its Details tab, where you supply data for the following fields, then click Save Changes. After you define deployment target sites, you can return to the workflows that your environment uses and specify these targets in the appropriate task elements.

You can also use the Details tab to delete a target site.

Caution: Before deleting a target site, make sure no active project uses a workflow that includes that target.

Site Name

Supply the target site's name—for example, Staging, or Production. See Identify Deployment Target Sites for more information.

Note: The installed workflows that ATG provides predefine one or two target sites with the following identifiers:

- `Production` is the sole target defined in the production-only workflow.
- `Staging` and `Production` targets are defined in the staging/production workflow.

In order to use a workflow with these predefined target site identifiers, you must define the deployment topology with site names that correspond exactly to these identifiers. You must also set the following property to `false`:

```
/atg/epub/Configuration.queryWorkflowTargetByID
```

Site Type

Choose one of the following options:

- **Workflow target**: A target that is defined in a project workflow. Project assets are deployed to this target only after they are approved for deployment
- **One-off target**: A target that is available for deployment from any project at any workflow stage. Development sites that are used to test code or performance are typically suitable for this designation. For more information, see One-Off Deployments in the Deployment Concepts chapter.

After the target site is initialized, its site type designation cannot be changed.

Site Initialization Options

Choose one of the following options:

- **Flag Agents Only**: Select if the data on the target site was imported for use as the starting data for the versioned database on the asset management server. If so, a full deployment is unnecessary when this target is initialized, because the same data is being sent back to the target site. Flag Agents Only initializes the target only with data that changed since the original data was imported.
- **Do a Full Deployment**: Select if target site data differs from versioned data on the asset management server. This option ensures that data is the same on both. Be aware that a full deployment can be time-consuming.

Site Description

Enter text you can use to identify this target. The description appears in the Deployment Administration Overview list for this target. You can edit the description later if required.

Source Repository and Destination Repository

You use the Source Repository and Destination Repository to set up mappings between this target's source and destination repositories. For each mapping, follow these steps:

1. Choose a versioned source repository whose assets you wish to deploy to this target.

   This list is populated with all versioned repositories in the application's configuration path. If the items you need do not appear, make sure all appropriate modules are included in your application.
2. Choose the matching destination repository on the asset management server. This is the repository that is configured to point to the target server's database; for more information, see Configure Deployment Data Sources and Destination Repositories.
3. Click Add. The mapping is added to the Source Repository list.

To remove a mapping from this list, click X.

Configure Target Site Deployment Agents

After you set the details of a target site, click on its Agents tab to configure its deployment agents. For each agent, click Add Agent to Site and supply agent data, then click Save Changes. Each agent requires the following data:

Name

The agent's name—for example, Repository Agent. See Identify Deployment Agents for more information

Transport URL

The agent's transport URL.

Include File Systems

Shows the VFS destinations to be updated by this agent. You populate this list from the Available File Systems list. For more information, see Plan Deployment Agent Responsibilities.

Note: Be sure to assign each asset type (repository or VFS) or asset destination (identified by its name in Nucleus) that resides in a shared data store to only one agent that uses the data store.

Essential

A checkbox that specifies whether this agent is required for deployment. During the deployment process, the DeploymentServer requires all essential agents in the target to be online and functional. If you do not set this checkbox for an agent, you identify it as an unessential agent—that is, it is not required for deployment. In this case, the unessential agent can be off-line and deployment can proceed if the following conditions are also true:

- The `/atg/epub/DeploymentServer` property `allowMissingNonEssentialAgents` is set to true. By default, this property is set to false. If set to false, unessential agents are treated like essential agents and must be online and functional during deployment.
- All agents that are flagged as essential are online and functional.

Note: This setting applies only to agents that are configured to deploy repository assets. An agent that is configured to deploy VFS assets is always regarded as essential, whether or not it is explicitly flagged as such.

During a deployment's Activate Data phase, the DeploymentServer tries to contact all essential and unessential agents before it starts any task that might affect those agents—for example, flush repository caches and switch data stores. If an unessential agent is offline at that time, the DeploymentServer logs error messages before continuing the deployment. In this case, you might need to perform the following maintenance tasks on the agent after deployment is complete:

- Manually flush its repositories' caches.
- If the agent is configured for deployments in switch mode, manually switch the agent's switchable data stores so its current live stores are the same as other agents.

  Note: This task is generally optional. A switch deployment always updates the newly inactive store (which is still used by the unessential agent) after updating and activating the newly live store. Also, the `DeploymentServer` automatically synchronizes all online agents during the next deployment's Activate Data phase.

Editing deploymentTopology.xml

You can manually define deployment topology from an XML file, which uses the following DTD:

`http://www.atg.com/dtds/publishing_deployment/publishing_deployment_1.0.dtd`

The ATG distribution provides a template for creation of this file:

`<atg10dir>/Publishing/base/src/config/atg/epub/deploymentTopology.xml`

This section provides information about defining a deployment topology in XML and importing it into your ATG Content Administration environment. After you perform these tasks, you can initialize targets from the Admin Console's Configuration page, as described later in this chapter in Initialize Target Sites.

Sample Deployment Topology XML

The following example shows part of a `deploymentTopology.xml` file:

```
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE publishing-deployment-topology SYSTEM
    "http://www.atg.com/dtds/publishing_deployment/publishing_deployment_1.0.dtd">

<publishing-deployment-topology>
  <target>
    <target-name>Production</target-name>
    <agent>
      <agent-name>PubAgent</agent-name>
      <include-asset-destination>
        /atg/epub/file/ConfigFileSystem
      </include-asset-destination>
      <include-asset-destination>
        /atg/epub/file/WebAppRefFileSystem
      </include-asset-destination>
      <transport>
        <transport-type>RMI</transport-type>
        <rmi-uri>
          rmi://production1.yourcompany.com:8860/atg/epub/AgentTransport
        </rmi-uri>
      </transport>
    </agent>

    <agent>
      <agent-name>WebAgent1</agent-name>
      <include-asset-destination>
        /atg/epub/file/WWWFileSystem
      </include-asset-destination>
      <transport>
        <transport-type>RMI</transport-type>
        <rmi-uri>
          rmi://production2.yourcompany.com:8860/atg/epub/AgentTransport
        </rmi-uri>
      </transport>
    </agent>

<!--  Order matters. Repository mappings must come after agent tags.
-->

  <repository-mapping>
```

```
    <source-repository>/atg/MyRepository</source-repository>
    <destination-repository>/atg/MyRepository_production</destination-repository>
  </repository-mapping>

  <repository-mapping>
    <source-repository>/atg/MyRepository2</source-repository>
    <destination-repository>/atg/MyRepository2_production</destination-repository>
  </repository-mapping>

  <repository-mapping>
    <source-repository>/atg/commerce/catalog/ProductCatalog</source-repository>
    <destination-repository>
       /atg/commerce/catalog/ProductCatalog_staging
    </destination-repository>
  </repository-mapping>

  <repository-mapping>
    <source-repository>
      /atg/commerce/pricing/priceLists/PriceLists
    </source-repository>
    <destination-repository>
      /atg/commerce/pricing/priceLists/PriceLists_staging
    </destination-repository>
  </repository-mapping>

  </target>

  <target>
    <target-name>Staging</target-name>
    <!--  Agents in the Staging target defined here
    ...
    -->
  </target>

</publishing-deployment-topology>
```

The `<repository-mapping>` tags specify the source and destination repositories for the deployment.

Defining Targets for One-Off Deployment

By default, a target is defined for use in a project workflow. In order to define a target for one-off deployments, nest this tag in the `<target>` definition:

```
<target-deploy-type>ONE-OFF</target-deploy-type>
```

If this tag is not set, the target is available only for deployment within a project workflow. If desired, you can explicitly set a target for workflow deployment as follows:

```
<target-deploy-type>WORKFLOW</target-deploy-type>
```

Including and Excluding Asset Destinations

You specify an agent's deployment responsibilities through two tags:

- <include-asset-destination> includes this asset destination for an agent.
- <exclude-asset-destination> excludes this asset destination for an agent.

Tags <include-asset-destination> and <exclude-asset-destination> are set to the deployed repository or VFS's absolute Nucleus component name on the asset manager server. Multiple instances of the <include-asset-destination> and <exclude-asset-destination> include and exclude multiple asset destinations, respectively.

The <exclude-asset-destination> tag is typically used to exclude a versioned content repository that is listed as a destination for the agent's target—for example, in cases where that agent does not need to access the given repository. By excluding the repository, you avoid the overhead that the agent otherwise incurs by invalidating the repository cache during a switch.

Note: When manually supplying values in a deploymentTopology.xml file for tags <include-asset-destination> and <exclude-asset-destination>, specify the names of the repositories exactly as they are listed in the VersionManager.

Mapping VFSs

You can configure deployment so a VFS of one name on the asset management server deploys to a VFS of another name on the agent server. You do this through a <vfs-mapping> tag that embeds tags <source-vfs> and <destination-vfs>:

- <source-vfs> specifies a VFS previously specified in an <incude-asset-destination> tag.
- <destination-vfs> specifies a VFS on the agent server.

If these tags are omitted, the VFS on the agent and asset management servers are assumed to have the same name.

For example, a deployment topology might include the following VFS for deployment:

```
<include-asset-destination>/atg/epub/file/WWWFileSystem
</include-asset-destination>
```

You can map this VFS so it deploys on the agent server to the VFS WWWFileSystemOnAgent, as in the following example:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE publishing-deployment-topology SYSTEM
   "http://www.atg.com/dtds/publishing_deployment/publishing_deployment_1.0.dtd">

<publishing-deployment-topology>
<target>
<target-name>AgentMappingTest</target-name>
    <agent>
```

```
    <agent-name>TestAgent</agent-name>

    <include-asset-destination>/atg/epub/file/WWWFileSystem
    </include-asset-destination>

    <vfs-mapping>
      <source-vfs>/atg/epub/file/WWWFileSystem</source-vfs>
      <destination-vfs>/atg/epub/file/WWWFileSystemOnAgent</destination-vfs>
    </vfs-mapping>

    <transport>
    <transport-type>RMI</transport-type>
    <rmi-uri>rmi://localhost:8860/atg/epub/AgentTransport</rmi-uri>
    </transport>
    </agent>
</target>
</publishing-deployment-topology>
```

Importing a Deployment Topology File

You import a `deploymentTopology.xml` file as follows:

To import a `deploymentTopology.xml` file:

1. In the ATG Business Control Center, navigate to the Admin Console.
2. From the Deployment Administration page, click Configuration
3. From the Configuration page, click Import from XML
4. Find the desired `deploymentTopology.xml` file and select it for import.
5. On the Configuration screen, click on the new target site and display its Details page.
6. Select Flag Agents Only (see Site Initialization Options) and click Save Changes.

Configure Deployment from Multiple Asset Management Server Clusters

Multiple clusters of asset management servers can deploy to a single target site. In order to implement this deployment topology, each cluster must have a unique identifier, which enables target site deployment agents to differentiate the deploying clusters. To establish a cluster's identity, you configure each of its servers as follows:

- Set the cluster name.
- Define the cluster hosts.

Other requirements that are specific to repository assets and file assets are discussed later in this section.

Set the Cluster Name

You can set the server's cluster name in two ways:

- In `/atg/dynamo/service/ClusterName`, set the `clusterName` property. Each cluster name must be unique; within a cluster, all servers must set their `clusterName` property to the same value.
- Save the cluster name in the ATG Content Administration database by setting two properties in `/atg/dynamo/service/ClusterName`: set `clusterName` as described above, and set `useClusterTable` to `true`. On initial application startup, the server saves the `clusterName` setting to the database. When starting up later, the server checks the database name and compares it to the `clusterName` ; if `clusterName` is empty, it uses the stored database value. If it finds a mismatch, it uses the property setting and writes this back to the database.

  By setting `useClusterTable` to `true` on all cluster servers, you can set the cluster name on just one server, in order to propagate it to the others via the database.

Define the Cluster Hosts

Each server must be configured with contact data about the other servers in the cluster. For each server in a cluster, configure these properties in `/atg/epub/Configuration`:

- `remoteHosts`: In a comma-delimited list, specify the host names of other servers in this server's cluster.
- `remoteRMIPorts`: In a comma-delimited list, specify the RMI port settings that are configured for the hosts specified in `remoteHosts`. List the ports in the same order as the corresponding hosts.
- `remotePorts`: Set this property only if you use the `FileSynchronizationDeployServer` to synchronize distributed file system assets (see Manage Distributed File Assets).

  In a comma-delimited list, specify the file synchronization ports that are configured for the other servers in their `FileSynchronizationDeployServer` components. List the ports in the same order as the corresponding hosts in `remoteHosts`.

For example:

```
remoteHosts=\
   jupiter.acme-widgets.com,\
   saturn.acme-widgets.com,\
   uranus.acme-widgets.com

remoteRMIPorts=\
   8860,\
   8860,\
   8860
```

Repository Assets

When setting up deployment from multiple asset management server clusters, make sure that each cluster deploys from a unique set of repositories. After you initialize a target site with the repository of one ATG Content Administration cluster, you should only deploy repository assets from that cluster.

Caution: Deployment of the same repositories from different ATG Content Administration clusters to the same target is not supported.

File Assets

Because different ATG Content Administration clusters can deploy to the same VFS repository, it is possible for file system assets on these clusters to share the same paths and names. Using cluster identifiers, deployment agents on the target site can keep track of deployed file ownership; this enables them to differentiate between files from different clusters whose paths are otherwise identical.

In order to enable deployment of files assets from multiple ATG Content Administration clusters, you must configure a target site in two ways:

- Enable tracking of file ownership.
- Set a strategy for handling ownership conflicts.

Enable Tracking of File Ownership

In order to enable file ownership in a virtual file system, `ownerCacheEnabled` must be set to `true` in the target site's VFS configuration. Depending on your configuration, this property is set in one or more of the following VFS configuration files:

- `/atg/epub/file/WebAppRefFileSystem`
- `/atg/epub/file/WWWFileSystem`
- `/atg/epub/file/ConfigFileSystem`

Note: In all configuration properties files, uncomment the `ownerCacheEnabled` property.

When `ownerCacheEnabled` is set to `true`, the target site deployment agent caches file ownership data. For online deployment, the owner cache file path is explicitly set in the VFS configuration's property `ownerCacheDataFile`. For switch deployments, SwitchableLocalFileSystem components do not have this property; instead, they write the owner cache data to the following locations:

- `ConfigFileSystem`:
  `/home/PublishingAgent/deploymentconfig/data/config`
- `WWWFileSystem`:
  `/home/PublishingWebAgent/deploymentdocroot/data`

The owner cache contents can help answer deployment questions related to file ownership. The following example shows a fragment from the owner cache file for the WWWFileSystem component. It shows three files that are stored in the local WWWFileSystem and their path. The cache file also identifies each file's cluster owner as CA1 or CA2.:

```
\Products\Doc\copyright_X-Boite.txt:CA1
\About_Us\execProfiles.txt:CA2
\Products\Doc\whitePaper_X-Boite.pdf:CA1
```

Given these file owners, an attempt by cluster CA1 to deploy `About_Us\execProfiles.txt` triggers an ownership conflict, which is resolved according to the `ownerStrategy` that is configured for the target site (see below).

Set a Strategy for Handling Ownership Conflicts

You can choose one of several strategies to handle potential file conflicts during deployment, by setting three properties on the target component `/atg/deployment/file/DeploymentConfiguration`:

- `ownerStrategy`
- `ownerSpecificWinnerId`
- `ownerWarning`

ownerStrategy controls how to handle file ownership conflicts. You can set this property to one of the following values:

ownerStrategy setting	Behavior
`RESOLVE_ERROR` (default)	On any file conflict, throw an error and stop deployment. This setting is appropriate when you wish to exercise complete manual oversight over all possible file ownership conflicts.
`RESOLVE_SPECIFIC`	Give precedence to one cluster over all others in the event of a conflict. If you use this setting, you must specify the cluster by also setting `ownerSpecificWinnerId`. Attempts by other clusters to deploy this file are ignored and yield a warning if `ownerWarning` is `true`. This setting is appropriate for automating ownership conflict resolution.
`RESOLVE_CLOBBER`	Use the most recent file, regardless of its owner. The overwritten file is owned by the latest deploying cluster. If `ownerWarning` is `true`, changes to file ownership yield a warning.
`RESOLVE_NO_CLOBBER`	Allow deployment of a file only from its latest cluster owner. Attempts by other clusters to deploy this file are ignored and yield a warning if `ownerWarning` is `true`.

Attempts to deploy conflicting file assets are ignored unless you set `ownerStrategy` to `RESOLVE_ERROR`, where all file deployments fail in the event of any conflict, or `RESOLVE_CLOBBER`, where all deployments succeed. `RESOLVE_ERROR` and `RESOLVE_SPECIFIC` provide the greatest levels of control for resolving file ownership conflicts.

ownerSpecificWinnerId specifies the identifier of the cluster to take precedence if `ownerStrategy` is set to `RESOLVE_SPECIFIC`. If `ownerStrategy` is set to another value, this property is ignored.

Set this property to the same string value used to set the cluster name property `clusterName`.

ownerWarning specifies whether to print warnings on ownership conflicts to the deployment agent. The default setting is `true`.

Managing Multi-Cluster Deployment Data

You can use the Dynamo Component Browser to view the details of a target site's DeploymentAgent component: the agent's deployment state, deployment mode, current live data store, and so on. The view includes Cluster Status Detail sections for each cluster that deploys to that agent. Each section lists the repositories deployed from the cluster, the time stamp of the latest deployment, and other cluster-specific information.

The DeploymentAgent component is found in Nucleus on the target site: `/atg/epub/DeploymentAgent`. You can access the Dynamo Component Browser as follows:

`http://`*`hostname`*`:`*`port`*`/dyn/admin/nucleus/atg/epub/DeploymentAgent/`

For more information about the Dynamo Component Browser and default port used by your application server, see the *ATG Installation and Configuration Guide*.

Deleting cluster data

On occasion, you might need to remove ATG Content Administration cluster data from a target site's deployment agents. Generally, you do this after deleting the deployment topology of an ATG Content Administration cluster. For example, you might remove the existing deployment topology of cluster CA1, then create a topology for cluster CA2 that maps the same repositories as CA1. Before making the CA2 changes live, you must remove the stale CA1 deployment data from the target site's deployment agents—both Publishing agents and Publishing Web agents; otherwise, the agents return errors when you try to deploy from CA2.

You remove cluster data from deployment agents as follows:

- **Publishing agent:** From the Dynamo Component Browser, find the cluster to remove in `/atg/epub/DeploymentAgent` and click its Delete button.
- **Publishing Web agent:** On the target site:
  - Navigate to: `/home/PublishingWebAgent/data`.
  - Delete the file `cluster-stat-`*`oldCluster`*, where *`oldCluster`* is the name of the cluster to remove, the same string value used to set the cluster name property `clusterName`.

Initialize Target Sites

After you configure target sites, you must initialize the target sites for use with the DeploymentServer.

Note: If you use an MS SQL database, review the next section, Initializing Targets on MS SQL with Clustered Primary Keys, before starting the procedure below.

This procedure assumes the asset management server is already running.

1. Start an application on each target server that includes the appropriate agents. For information on starting agents, see Running Deployment Agents earlier in this chapter.
2. Access the ATG Business Control Center and log in as a user with access rights to the Admin Console.
3. In the Home page, expand the ATG Content Administration option in the Operations list, and click Admin Console.
4. Click Configuration.
5. Click Make Changes Live.
6. Select Flag Agents Only or Do a Full Deployment as appropriate (see Site Initialization Options), and click Make Changes Live.
7. Display the Overview page, and review the information that appears for the target sites you initialized.

You can also confirm target initialization by viewing the service information for the `atg/epub/DeploymentServer` component via the Component Browser in the ATG Dynamo Server Admin interface.

If you need to restart an initialized target site application, you must also restart the asset management server applications that deploy to it.

Initializing Targets on MS SQL with Clustered Primary Keys

If your ATG Content Administration environment uses an MS SQL Server database that employs clustered primary keys for repository tables, locking errors might occur when you initialize your targets:

```
Warning /atg/deployment/DeploymentManager Transaction (Process ID 164) was
deadlocked on lock resources with another process and has been chosen as
the deadlock victim. Rerun the transaction.
```

These errors can occur if `READ_COMMITTED_SNAPSHOT` is not enabled for the database, as described in the *ATG Installation and Configuration Guide*. Check all recommended settings.

If deadlock persists, reduce the number of threads by setting the following property values in the `/atg/deployment/DeploymentManager` component:

```
phaseCompletePollingInterval=30000
maxThreads=5
transactionBatchSize=5
```

If the errors still occur, try the following settings:

```
phaseCompletePollingInterval=60000
maxThreads=1
transactionBatchSize=1
```

Reducing the number of threads can adversely affect performance. If this occurs, contact ATG Customer Support Services for further help.

Adding Agents to an Initialized Target

After you initialize a target, you can add a deployment agent to that site as follows:

1. Configure the deployment agent as described earlier (see Configure Target Site Deployment Agents).
2. Return to the target site's Configuration page and click Make Changes Live.
3. Perform the steps associated with one of the following conditions:

If the new agent is responsible for...	Perform these steps:
File assets (ConfigFileSystem or WWWFileSystem)	Perform a full deployment on the target site.
File assets whose file directories are on a sharable network resource used by other agents in the ATG Content Administration cluster	Check to be sure that the file system points to the network resource.
Repository assets	**1.** Open the Component Browser page in ATG Dynamo Server Admin and navigate to `/atg/epub/DeploymentServer` **2.** Scroll to the target site. In the Force Snapshot ID field, enter the Current Installed Agent Snapshot value. **3.** Click Init.

4. If using switch deployment, verify that the new agent switchable datastore settings are consistent with the agents for the same target site, as follows:
   - From the Admin Console, click Overview.
   - Navigate to the target site of the new agent.

- Click on the Agents tab.
- From the Agents tab, click Switchable Datastores

The Agents tab lists the target datastores that each agent is configured to switch, and identifies which datastores are currently live.

Configure Deployment Event Listeners

You can manually monitor the status of deployments through the ATG Business Control Center's Admin Console. The Details section displays the current status of each deployment target. Alternatively, you can use event listeners that automatically monitor and manage deployments. These listeners listen for the deployment events that are fired upon the different state changes in a deployment. On receiving these events, the listener can respond with the appropriate action—for example, by sending email. See the following subsections for more information:

- Understanding Deployment Events
- Creating and Configuring a DeploymentEventListener

Note: This section assumes you are familiar with events and event listeners, as covered in the *Core Dynamo Services* chapter of the *ATG Programming Guide*.

Understanding Deployment Events

As described in Deployment Process, each time the deployment or an agent begins or ends a deployment phase (or fails during a phase), a DeploymentEvent is fired to indicate the change in state. These DeploymentEvent objects are fired by the following event sources:

- DeploymentServer when a state change occurs in the deployment.
- DeploymentAgent when a state change occurs in the target server agent.

The DeploymentEvent object includes the old state, the new state that caused the event to fire, the deployed project, and other information. For a full list of properties, see `atg.deployment.common.event.DeploymentEvent` in the *ATG API Reference*.

Creating and Configuring a DeploymentEventListener

You can automate many aspects of deployment monitoring and management by configuring event listeners on the asset management server and the target servers, to listen for deployment events that include a specific state code.

To create and configure a DeploymentEventListener:

1. Define a listener service class that implements the interface DeploymentEventListener and the method `deploymentEvent(DeploymentEvent pEvent)`.
2. Configure an instance in Nucleus on the appropriate server and register it with the DeploymentServer or DeploymentAgent as appropriate:

- Configure the listener on the asset management server and register it with the `/atg/epub/DeploymentServer`, so it listens for deployment events fired when the deployment changes its state.
- Configure the listener on the target server and register it with the local `/atg/epub/DeploymentAgent`, so it listens for deployment events fired when an agent changes its state.

For an example, examine the source code for the `DeploymentEmailer listener` class, which is located at:

`<ATG10dir>/Publishing/samples/Java`

DeploymentEmailer Listener

ATG Content Administration includes this DeploymentEventListener class:

`atg.deployment.common.event.DeploymentEmailer`

You can use this listener to send email notifications when a deployment succeeds, is interrupted, or fails.

By default, instances of the `DeploymentEmailer` class listen for deployment events that are fired when a deployment enters one of the following states:

- `DEPLOYMENT_COMPLETE`
- `EVENT_INTERRUPT`
- Any error state such as `ERROR` and `ERROR_PREPARE`

On receiving such an event, the listener sends an email message to a specified recipient.

To set up a `DeploymentEmailer` listener, first configure an instance of the `DeploymentEmailer` class on the asset management server. For example:

```
$class=atg.deployment.common.event.DeploymentEmailer

SMTPEmailSender=/atg/dynamo/service/SMTPEmail
fromAddress=personA@myCompany.com
toAddress=personB@myCompany.com
useShortMessage=true
```

The following table describes the properties to configure:

Property	Description
SMTPEmailSender	The SMTPEmailSender service through which to send the email. ATG provides a standard component of type SMTPEmailSender at: /atg/dynamo/service/SMTPEmail For more information on email senders and listeners, see the *ATG Programming Guide*.
fromAddress	The sender's email address.
toAddress	The destination email address.
useShortMessage	Specifies the level of message detail: True: The listener sends a short, pager-sized message. False (default): The listener sends a longer, more verbose message.
active	Specifies whether to enable the listener: True (default): The listener is enabled to send email notifications. False: The listener is disabled.

After the DeploymentEmailer listener is configured, register it with the /atg/epub/DeploymentServer by adding it to the list of listeners in the deploymentEventListeners property.

If you require different functionality for the event listener, create a subclass of DeploymentEmailer, override its deploymentEvent() method with your own implementation, and configure an instance of this subclass.

Schedule Deletion of Empty Folders

When you deploy a renamed folder to a target site, the assets in the original folder are moved to the new one; however, the original folder—now empty—remains on the target site. Empty folders are periodically removed by the following VFS components:

- ConfigEmptyDirDeleter monitors the ConfigFileSystem in the Publishing agent.
- WWWEmptyDirDeleter monitors the WWWFileSystem in the Publishing Web agent.

These components wake up at regular intervals and delete any empty folders that are older than the configured age. You can configure both components in their respective properties files, in /atg/epub/file:

Property	Description
`enabled`	A boolean, must be set to true (the default) in order to schedule empty directory removal.
folderAge	Specifies in milliseconds how old an empty folder must be before it is eligible for removal. The default is 604800000 (one week).
`schedule`	Schedules empty folder removal by specifying a PeriodicSchedule and/or RelativeSchedule, or a CalendarSchedule. Default setting: `every 1 hour in 1 hour` For detailed information on setting schedules, see the section Scheduler Services in the *ATG Programming Guide*.

If you create a custom VFS, you should create and configure its own `ConfigEmptyDirDeleter` or WWWEmptyDirDeleter component for the appropriate agent. To configure this component, simply copy the appropriate properties file from the installed ATG distribution into the appropriate folder, and modify as needed. For example:

```
$class=atg.vfs.EmptyFolderCleanupService

virtualFileSystem=custom-VFS-name

scheduler=/atg/dynamo/service/Scheduler
schedule=every 24 hours in 4 hours

enabled=true
folderAge=17280000
```

You must also specify this component as one of the initial services in `atg/epub/file/Initial.properties`. For example:

```
initialServices+=\
  ConfigFileSystem,\
  ConfigEmptyDirDeleter
```

Cache Checksums for File Assets

ATG includes the following checksum caching features, which you can use to improve performance for file asset deployments.

Note: For information on configuring deployment to optimize performance, especially for environments that require frequent deployments or deploy very large numbers of assets, refer to *Configuring DAF Deployment for Performance* in the *ATG Programming Guide*.

Checksum Caching on the Asset Management Server

In ATG, 64-bit file data checksums are calculated whenever a file is saved. Importing files with `importRepository` also triggers checksum calculations.

The checksums are stored in the PublishingFileRepository. To disable this feature, remove the `checksum` property from `publishingFiles.xml`.

Checksum Caching on Production Servers or Agents

The LocalVFSService and SwitchableLocalFileSystem components include checksum caching features. Calculated checksums are stored in memory, and they are occasionally written to or retrieved from a single file. JournalingFileSystemService components use this feature if it is enabled on the LocalVFSService to which they point.

This feature is useful only for file systems that have large numbers of files, and it is disabled by default, though enabling it does not adversely affect file systems with small numbers of files. It is highly recommended for the WWWFileSystem component on the agent if significant numbers of assets (more than 1000) are deployed to that component. It is enabled by default in the WWWFileSystem component on the PublishingWebAgent.

The following properties control this feature:

- `checksumCacheEnabled`: Enables/disables checksum caching.
- `checksumCacheEncoding`: Specifies the encoding used to save the cache data —by default, UTF-8. If your environment uses file paths with non-UTF-8 characters, set this property to the encoding set that you use.
- `checksumCacheDataFile`: The SwitchableLocalFileSystem automatically saves the checksum cache data files in its data directory, but to enable checksum caching on a LocalVFSService, set the value of this property to the location where a file can be saved.

Checksum Verification Deployment Mode

By default, full file asset deployments check the timestamp, file size, and checksum of a file on the agent before pushing a new version of the file from the asset management server. If the files match, the new file is not sent. Note that the timestamp check is not for the exact timestamp but for a more recent file on the server.

Occasionally, you might want to disable checksum verification in order to purge all file assets from a target site. You can disable checksum verification by setting the following property in the `/atg/deployment/file/DeploymentConfiguration` component on the asset management server:

```
noVerificationByChecksum=true
```

Note: This property must be set to true in order to configure background deletion of file system assets.

Local Copy During Switch Deployment

By default, full switch deployments for file assets copy the live directory of a SwitchableLocalFileSystem to the inactive directory on the second apply phase. The copy follows the verification rules described in Checksum Verification Deployment Mode if checksum verification is enabled (`noVerificationByChecksum=false`).

There are certain site configurations where turning this feature off might improve performance, specifically if there is a simultaneous repository deployment that takes a similar amount of time. The feature can be disabled by setting the following property in the `/atg/deployment/file/DeploymentConfiguration` component on the asset management server:

```
noChecksumAfterCopy=true
```

12 Deploying Project Assets

After a project is approved for deployment, deployment of its assets to the target site can be launched automatically or manually. Automatic deployment can be configured in two ways:

- Configure the RecurringDeploymentService so all pending deployments are launched according to the specified schedule, as described in this chapter.
- Configure workflows to enable immediate project deployment. For information about configuring workflows, see the *ATG Personalization Programming Guide*.

You can also manually launch deployments through the ATG Business Control Center Admin Console.

This chapter covers the following deployment topics:

Configuring the RecurringDeploymentService

You can set up ATG Content Administration so it automatically deploys all projects listed in the To Do tab of the Admin Console in the ATG Business Control Center.

To enable this functionality:

1. Set up your content development and target environments for ATG Content Administration deployments (review the steps described earlier in Setting Up Deployment).
2. Configure the RecurringDeploymentService to schedule pending projects for deployment:

   ```
   /atg/epub/deployment/RecurringDeploymentService
   ```

The following table describes each property to configure:

Property	Description
`candidateTargetNames`	Specifies targets to consider for scheduled deployments. If this property is not set, the RecurringDeploymentService deploys to all target sites that are set up for deployment. The RecurringDeploymentService logs error messages for invalid targets.
`deploymentServer`	The path of the DeploymentServer, by default `/atg/epub/DeploymentServer`.
`enabled`	Specifies whether the service is enabled. The default is false. Be sure to set all other required properties before enabling this service.
`forceFull`	If set to true, specifies that the RecurringDeploymentService performs a full deployment. By default, this property is set to false (performs incremental deployment).
`personaPrefix`	The substring in an ACL that is used to identify the user in the `UserAuthority`. The default is `Profile$login$`.
`schedule`	Specifies the schedule for deploying pending projects. The schedule is supplied to the Scheduler service specified in the Scheduler property. In order to set up a recurring schedule, specify either a `PeriodicSchedule` or `CalendarSchedule`. For more information on defining these types of schedules, see the discussion on Scheduler services in the *ATG Programming Guide*. The default setting is calendar `* * * * * 0`, which specifies to run the job every hour on the hour.
`scheduler`	The path of the Scheduler service to manage deployment scheduling; by default, set to `/atg/dynamo/service/Scheduler`.
`transactionManager`	The path of the Transaction Manager to use; by default, set to `/atg/dynamo/transaction/TransactionManager`.
`userAuthority`	The `userAuthority` that resolves the user specified in the username property.
`username`	Specifies the user to authenticate. The default is publishing. Set this property to a user name with full access to the `/atg/epub/PublishingRepository` in order to ensure that the user against whom queries are executed has the required access privileges to all projects.
`versionManager`	The path of the VersionManager to use; by default, set to `/atg/epub/version/VersionManagerService`.

At application startup, the RecurringDeploymentService schedules a new deployment job with the `/atg/dynamo/service/Scheduler` (class `atg.service.scheduler.Scheduler`). The Scheduler component is a service that monitors all scheduled jobs, and calls as needed on the appropriate services

to execute them. The ATG Dynamo Server Admin lists all jobs that the Scheduler component currently monitors.

According to the schedule defined in the property `RecurringDeploymentService.schedule`, the Scheduler calls on the RecurringDeploymentService to perform its scheduled task. When called, the RecurringDeploymentService retrieves all pending projects in the system and deploys them to all target sites specified in the `candidateTargetNames` property. If this property is not set, the RecurringDeploymentService deploys to all targets that are configured for deployment. You can monitor and manage scheduled tasks like other deployment activities.

Extending the RecurringDeploymentService

The default implementation of the RecurringDeploymentService schedulable service acts on all projects that are approved for deployment. You can extend this service to include only a subset of approved projects. For example, you might want to include only projects that contain changes to the product catalog.

To extend the RecurringDeploymentService in this manner, create a subclass of the `atg.epub.deployment.RecurringDeploymentService` class and override the following method:

```
public Collection getProjectsToDeploy(Collection pAllPendingProjects)
```

By default, `getProjectsToDeploy()` returns the collection of all projects approved for deployment. The custom implementation should filter the collection as needed.

After you create the subclass, configure an instance of the class in Nucleus by placing the configuration file in the configuration path:

```
/atg/epub/deployment/RecurringDeploymentService
```

Note: If project assets that are included in the filtered collection reference assets in other projects that are approved for deployment, the system automatically includes those projects in the deployment.

Deploying from the Admin Console

The ATG Business Control Center Admin Console lets you manage and troubleshoot deployments manually. The Admin Console can perform the following tasks:

- View deployment details, including errors.
- Stop deployments.
- Set deployment parameters such as time of deployment and its scope.
- Manage the deployment queue—for example, move a project to the front of the queue.
- Switch a Target Site's Datastores on a site that is configured for switch deployment.
- Roll back deployments—that is, undoing a specific deployment by reverting a target site to a previous state.

- View deployment agents status: status, errors, and configuration information.

Accessing the Admin Console

To access the Admin Console:

1. Make sure the ATG asset management server is running.
2. Log into the ATG Business Control Center as a user who has access rights to the Admin Console generic activity. By default, the `EPub-Admin` or `EPub-Super-Admin` role provides this access (username and login `publishing/publishing`). For information on generic activity access, see Access to Generic Activities.

   You also need an ATG Portal role. See ATG Content Administration Users for more information.
3. In the Home page, expand the ATG Content Administration option in the Operations list, and click Admin Console. Two options appear:
   - **Configuration** lets you configure target sites for deployment (see Define the Deployment Topology).
   - **Overview** displays summary information about the target sites that are set up for deployment and their status. You can obtain more detailed information about each target by clicking on the link under Site Name. This displays target site information in a series of tabs: Details, To Do, Plan, Projects, and Agents

   Note: The Overview pane displays sites according to the order that is set in the Configuration pane through the Site Priority controls.

View Deployment Details

The Details tab for a target site shows the following information:

- The name of the project included in the most recent deployment (the project being currently deployed)
- The number of projects waiting to be deployed
- The name of the project included in the last successful deployment (before the one listed as most recent)
- The name of the project that is next in the queue of projects waiting to be deployed

If a project is currently being deployed, detailed information about it appears in the panel at the bottom of the screen. For example, you can see the type of deployment (full or incremental), the time and date that the deployment started, and the name of the deploying server.

The Details tab also displays information about the accessibility of the target site. If the site is inaccessible or if deployment is failing, warnings appear in this tab. After you resolve any deployment issues, you are presented with options that let you resume deployment or roll back deployment to the last set of assets that were successfully deployed to the target site.

Stop Deployments

To stop all ongoing deployments manually: from the Details tab, click Halt Deployments.

The Details tab also contains a Critical Reset button, which you can use to delete the last deployment to the given target (the current deployment). The operation clears the deployment state one by one from each agent, deleting the deployment data directories from the ATG asset management server and the target. All agents are reset to IDLE. Typically you use the Critical Reset operation only in situations where errors occurred that are irrecoverable by stopping the deployment, fixing the error, and either resuming or rolling back the deployment.

Set Deployment Parameters

Use the To Do tab to manage any deployments that are not automatically started by a workflow deployment element or the RecurringDeploymentService. From this tab, select the projects to deploy and click Deploy Selected Projects. Options appear that let you perform the following tasks:

- Deploy the projects immediately or schedule them for a given date and time.

  Note: If the On Date and At Time fields are empty, the projects are scheduled for immediate deployment.
- Perform a full or incremental deployment of project assets.
- Impose strict constraints for updating target site data.

Click Deploy after you select the appropriate settings for this project.

Setting Strict Update Constraints

Before deploying you can set one or both of the following checkboxes:

- Strict File Update Operations
- Strict Repository Update Operations

If these options are set, the deployment succeeds only if all project assets of the specified type—file and/or repository assets—have matching data on the target site. For example, a deployment might set Strict File Update Operations; if the deployment specifies to remove a file asset and that asset is missing on the target site, the deployment fails and returns errors.

These options can be useful for sites that set rigorous standards for data integrity. If these options are left unset, data discrepancies generate warnings, but do not prevent the deployment from going forward.

Manage the Deployment Queue

Use the Plan tab to manage the queue of projects that are approved for deployment (see Deployment Scheduling). The Plan tab also shows projects that are scheduled for later deployment.

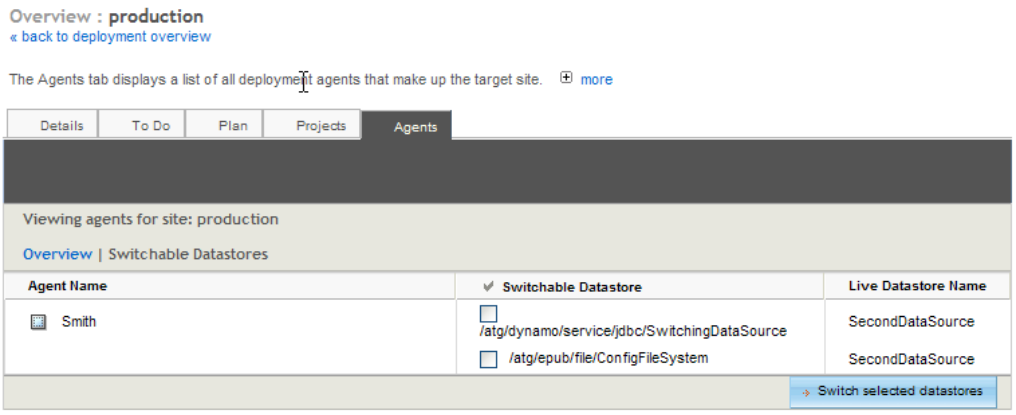
To move a project to the head of the deployment queue, select the project and select Run Now in the Action column.

Switch a Target Site's Datastores

If a site is configured for switch deployment, you can perform the switch through the target's agents as follows:

1. From the Admin Console, click Overview.
2. Navigate to the target site where you wish to switch datastores.
3. Click on the Agents tab.
4. From the Agents tab, click Switchable Datastores

The Agents tab lists the target datastores that each agent is configured to switch, and identifies which datastores are currently live:



To switch datastores:

1. Check one or more SwitchingDataSources and SwitchableLocalFileSystem to switch the corresponding datastores.
2. Click Switch Selected Datastores.

For information about configuring a site for switch deployment, see Configure Switch Deployment in the Setting Up Deployment chapter.

Roll Back Deployments

The Projects tab shows a chronological listing of all deployed projects. Those deployed most recently appear at the top of the list.

You can revert the target site to a particular deployment state by selecting a project from the list and clicking Rollback to Selected. This behavior starts a deployment that rolls back all projects deployed after the one you selected.

When a target site is rolled back to an earlier project, those projects whose assets are no longer on the site are shown in grey.

Roll back Constraints

Two constraints apply to roll back deployment:

- You cannot roll back deployment on a target site while an active deployment is in progress on that site. If the roll back operation is urgent, you must first revert all active deployments. In general, a roll back deployment can succeed only if the target is unchanged in the interval between when you schedule the roll back and when it actually executes.
- You cannot roll back deployment on a one-off target; the Projects tab for a one-off target site is read-only. If you perform a full deployment to a one-off target, the Projects list is emptied when deployment is complete.

View Deployment Agents Status

Use the Agents tab to view status, error, and configuration information about the agents set up for this target site.

Troubleshooting Deployment

Deployments occasionally require manual intervention, either because the deployment failed or stalled in an indeterminate state. This section describes how to perform the following tasks:

- Recover from deployment failure.
- Release a stalled deployment.

Recover from Deployment Failure

When a deployment fails—for example, a server crashes or network connectivity lapses—you can often recover as follows:

1. In the ATG Business Control Center Admin Console, navigate to the Details tab for the deployment's target site and stop the deployment.
2. Click Resume.

   The deployment operation picks up where it stopped and continues to completion. You can resume full and incremental deployments, and you can resume a deployment multiple times in the event of successive failures.
3. If attempts to resume deployment fail:
   - Stop the deployment.
   - Click Rollback. The target site deployment agents undo all work completed thus far in the failed deployment and restore the previous set of assets as the active set on the target. Only incremental deployments are candidates for roll back; a full deployment starts by deleting all target site data, which cannot be restored.
   - Deploy the project again.

Note: You can roll back only once. If a failure occurs while rolling back a target site, the target site is in an indeterminate state. In this situation you must perform a full deployment of the previous set of assets (created by one or more projects) to reestablish it as the active set on the target site.

After deployment recovery is complete, the target site is in a known, stable state and you can resume deploying to it as usual.

For information on full and incremental deployments, see Deployment Scope.

Resuming Deployment from a Failed Asset Management Server

In a clustered environment, an asset management server might fail during a deployment that it initiated. In that case, you cannot access the deployment from other asset management servers in the same cluster, When viewed from other servers, the Details tab for the deployment target does not display the usual deployment options, such as resume, stop, and roll back. Instead, it displays the following error message:

```
An RMI error encountered calling remote current deployment
'deployment-id' to target 'Production': getStatus() may or may not have been
passed to the running deployment
```

After the initiating server restarts, all available actions that pertain to the deployment become available again, and are accessible from any asset management server in the ATG Content Administration cluster.

Release a Stalled Deployment

Resolution of project deployment—successful and unsuccessful—depends on delivery of a Success or Failure status message to the project's workflow. Under rare circumstances, this message is never delivered., and the project stalls indefinitely in the workflow's Wait for staging deployment completion or Wait for production deployment completion stage. When this happens, the ATG Business Control Center Admin Console shows no deployment information for this project as active or pending.

You can resolve this problem in the ATG Dynamo Server Admin Component Browser as follows:

1. Navigate to this component:

   `/atg/epub/messaging/PublishingMessageSource`

2. Construct a message to send to the stalled deployment, by setting the following fields:
   - **Select Project**: Choose the project whose deployment is stalled.
   - **Select Target**: Choose the deployment's target site.
   - **Select Success or Failure State**: Choose the message to send, Success or Failure. A Success message advances the workflow; a Failure message reverts the workflow to its previous stage, Approve for production deployment, or Approve for staging deployment. In general, it is safest to send a Failure message and retry the deployment.
3. Click Send Deployment Complete Message.

Automating Recovery from Transient Errors

Occasionally, a full deployment can stop in an incomplete state due to a non-fatal error—for example, an SQL deadlock or a connection timeout. In cases like these, you can manually resume the deployment through the ATG Business Control Center Admin Console. Alternatively, you can automate the recovery process by setting the DeploymentManager property `transientErrorRetryCount`. This property sets the number of times a deployment thread tries to redeploy a transaction batch after a recoverable error. This property's default value is 0, which disallows retries.

`transientErrorRetryCount` is primarily aimed at facilitating recovery from SQL deadlock errors, though not exclusively. Deadlocks are data dependent and some databases handle them better than others. You should set this property to a reasonable value for your environment—typically, between 1 and 5.

13 Purging Asset Versions

Over time, the versioning system in ATG Content Administration can accumulate a large number of asset versions and completed projects. As asset versions accumulate, they can strain storage capacity and system performance. It is generally a good idea to periodically purge versioned repositories of outdated projects and asset versions.

AN ATG Content Administration purge encompasses all versioned repositories and file systems. The purge operates on asset versions and projects whose check-in date is earlier than the specified purge cut-off date.

The length of a purge depends on the number of repository and file assets that need to be purged. A purge that encompasses a large number of file assets can be lengthy. This is especially likely if no purge has been executed for a long time. In that case, you can minimize system overhead and user inconvenience by scheduling multiple purges, where each purge specifies a more recent cut-off date than the one before it.

Note: Each purge operation executes in a transaction. If a purge encompasses a large number of assets, you might need to raise your application server's transaction timeout setting—for example, reset the JBoss `TransactionTimeout` attribute in `<JBdir>/server/atg/conf/jboss-service.xml`.

Scheduled and On-Demand Purge

You can initiate purges in two ways:

- Scheduled purges automatically execute at fixed dates or regular intervals.
- On-demand purges can be initiated at any time through the ATG Dynamo Server Admin.

General Safeguards

Before you start a purge, back up all affected datastores and file systems. If you run scheduled purges, you can write code that uses the PurgingService event listener to trigger the required backup operations.

After executing a purge, you might need to repair database indexes, especially if a large amount of versioned data is removed. This is often the case the first time you purge versioned data. Other database-specific considerations might apply. For example, given an Oracle database, a purge that includes millions of asset versions might require a large undo tablespace; DB2 might require large buffer pools. For more information, consult your database system documentation.

Restricted Operations

In order to ensure data consistency, a purge automatically blocks operations that are associated with asset changes and project deployment. Before starting a purge, users should be advised to avoid using the ATG Business Control Center until the purge ends; otherwise, errors might occur if they try to access assets affected by the purge.

During a purge, the following ATG Content Administration operations are blocked and generate error messages:

- Create assets.
- Check in assets—for example, advance a project to verify deployment.
- Check out assets—for example, add assets to a project.
- Resolve asset conflicts.
- Remove assets from a project.
- Delete a project.

You can advance a project to deploy; however, the deployment does not start until the purge is complete.

Protected Versions

An asset version is protected from being purged regardless of the purge cut-off date if any of the following conditions is true:

- It is the head version of that asset.
- It is the predecessor version of an asset that is checked out to a project. Predecessor versions must be retained in order to detect and resolve potential version conflicts.
- It is referenced by another asset that cannot be purged—for example, because the second asset is checked out to a project.

Purge has no effect on the working versions of assets that are checked out to a project.

After the purge completes, the PurgingService verifies that no protected versions were inadvertently deleted. Each purge executes under transaction control, so any verification failure causes the entire purge to roll back. In the unlikely event of purge rollback, you can contact ATG Customer Support for help diagnosing the problem before retrying the purge operation.

Scheduled Purges

The PurgingService component `/atg/epub/purge/PurgingService` can be configured to start purges at regular intervals or on specific dates. By default, the `scheduler` property is set to `/atg/dynamo/service/Scheduler`. As with other ATG services, you can set the `schedule` property as follows:

- PeriodicSchedule and RelativeSchedule, which specify to run the purge at regular intervals, with a start time relative to the current time. For example, the following `schedule` setting specifies to run the PurgingService every 60 days, starting in two hours:

  ```
  schedule=every 60 days in 2 hours
  ```

- CalendarSchedule that schedules purges at specific points on the calendar. For example, the following `schedule` setting specifies to run a purge on the last Sunday of each month at 1:01am:

  ```
  schedule=calendar * . 1 last 1 1
  ```

You set the purge cut-off date by supplying an integer value to the `maxAgeDays` property—by default, set to 365. Thus, a value of 15 specifies that each scheduled purge removes versions that are 15 days old or older.

The exact purge cut-off timestamp is determined by the property `roundPurgeCutoffToPreviousMidnight`. If set to true (default), the value supplied to `maxAgeDays` is rounded to the previous midnight. If set to false, the purge cut-off timestamp is not rounded.

For detailed information on setting schedules, see the Scheduler Services section in the *ATG Programming Guide*.

On-demand Purges

An on-demand purge provides a summary metrics report that outlines the potential impact of purge criteria. Before starting the purge, you can evaluate the number of asset versions and projects involved, and the overhead likely to be incurred. If a purge cannot execute, the summary metrics report provides detailed information on the causes of failure and points to possible remedies.

You can initiate a purge at any time through the Component Browser in the ATG Dynamo Server Admin:

```
/atg/epub/purge/PurgingService/
```

From this page, you launch an on-demand purge in the following steps:

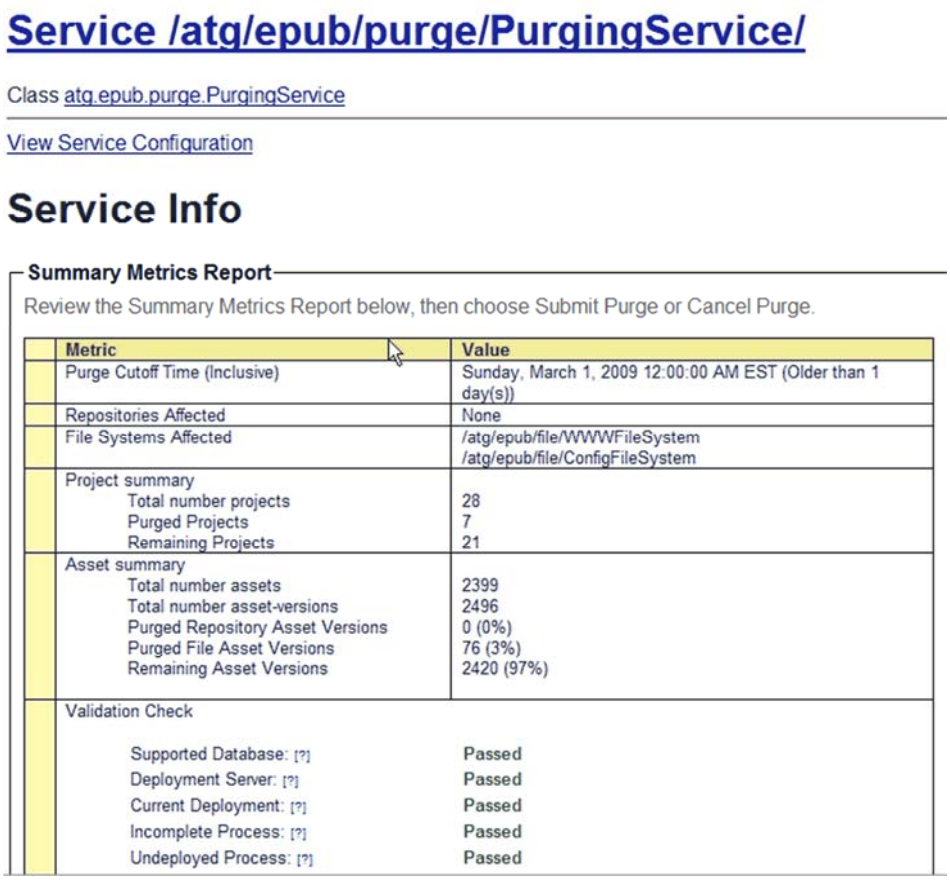
1. Specify the purge cut-off date by entering the corresponding number of days in the purge cut-off field. For example:

   Purge projects and unused versions of assets older than 180 **days.**

2. Click Show Metrics.
3. Review the data in the generated Metrics Report.
4. If no validation check errors occur, execute the purge by clicking Submit Purge.

The exact purge cut-off timestamp is determined by the property `roundPurgeCutoffToPreviousMidnight`. If set to true (default), the value supplied to the purge cut-off field is rounded to the previous midnight. If set to false, the purge cut-off timestamp is not rounded.

A Summary Metrics report is generated before the purge begins, and another one after the purge is complete. The report outlines the operation's scope—for example, the number of projects and asset versions removed, and the number of projects and asset versions that remain. It lists results from the validation checks that the purging service uses to evaluate the versioned data to purge and, more generally, the current state of the ATG Content Administration system. You can proceed with the purge only if all validation checks succeed. For example:



Service /atg/epub/purge/PurgingService/

Class atg.epub.purge.PurgingService

View Service Configuration

Service Info

Summary Metrics Report

Review the Summary Metrics Report below, then choose Submit Purge or Cancel Purge.

Metric	Value
Purge Cutoff Time (Inclusive)	Sunday, March 1, 2009 12:00:00 AM EST (Older than 1 day(s))
Repositories Affected	None
File Systems Affected	/atg/epub/file/WWWFileSystem /atg/epub/file/ConfigFileSystem
Project summary	
Total number projects	28
Purged Projects	7
Remaining Projects	21
Asset summary	
Total number assets	2399
Total number asset-versions	2496
Purged Repository Asset Versions	0 (0%)
Purged File Asset Versions	76 (3%)
Remaining Asset Versions	2420 (97%)
Validation Check	
Supported Database: [?]	Passed
Deployment Server: [?]	Passed
Current Deployment: [?]	Passed
Incomplete Process: [?]	Passed
Undeployed Process: [?]	Passed

Cached Summary Metrics

By default, an on-demand purge generates a summary metric report as a background process and caches the results. This ensures that the SQL query completes execution and the resulting report can be accessed regardless of the browser state. Thus, you can navigate away from the PurgingService page before the report is complete or after viewing it; the cached report remains available for a specified period of time—by default, 24 hours. On returning to the PurgingService page, you can access the cached report via a link on the PurgingService page, and submit the purge for execution if desired. Cached report links are listed in reverse order of creation, where the most recent report is listed first.

Two properties in the component `/atg/epub/purge/SummaryReportCache` control caching behavior:

- `maximumCacheEntries` specifies how many summary metrics reports can be cached at any one time. The default setting is 5. If the number of generated reports exceeds

this property, the purging service uses a least-recently-used (LRU) algorithm to determine which report to flush. If usage is equal, it flushes the oldest report. To disable caching, set this property to `0`.

- `maximumEntryLifetime` specifies in milliseconds how long a report can remain cached before it is invalidated. The default setting is `86400000` (24 hours).

After a purge executes, all summary metrics report caches are flushed.

Purging Unused Target Branches

Any purge, regardless of the cut-off date, removes all unused target branches and their assets from the versioning system. Thus, you can initiate a purge that removes only those branches by setting an unrealistically high cut-off date—for example, 14,000.

Summary Report Precision

Earlier (pre-9.0) versions of ATG Content Administration support deployment from child branches of the versioning system. If you used these versions of ATG Content Administration, it is likely that your versioning system retains a number of asset versions from child branches that were created for earlier projects.

The PurgingService property `preciseSummaryReports` determines whether a summary metrics report takes into account all branch asset versions when it calculates the number of asset versions to be purged. By default, this property is set to false. Given this setting, when the purging service generates the pre-purge summary report, it ignores asset versions on the main branch that have derived assets on child branches. By doing so, pre-purge summary report statistics are liable to differ slightly from actual purge results. Setting this property to true guarantees precision in pre-purge calculations; however, it can dramatically increase the overhead incurred by report generation.

Validation Checks

Before starting a purge, the PurgingService runs a series of validation checks to determine whether it can safely remove versioning data within the specified cut-off date. Purge execution requires the following conditions to be true:

- The PurgingService only supports database systems that are also supported for production use. The PurgingService does not support development-only database systems such as Solid and MySQL.
- The DeploymentServer is active and the deployment topology is up-to-date.
- No deployments are in progress. Because a deployment and a purge both affect snapshot data, consistency of data can be ensured only if all deployments are complete before the purge begins.

- All processes whose projects were checked in before the purge cut-off date are complete. Processes whose projects have not deployed are regarded as incomplete and should not be purged.
- For all initialized targets, the deployed snapshot date follows the purge cut-off date. In order to avoid deployment errors, a purge must never include the deployed snapshot.
- No projects to be purged are scheduled for deployment.

If a purge attempt fails one or more validation checks, you can often pass these checks by setting an earlier purge cut-off date. For example, if a purge fails to execute because purge candidates include an undeployed project, supply an earlier cut-off date that removes the project from the purge and allows the purge to proceed.

Appendix A: Database Schema

The ATG Content Administration database schema includes the following types of tables:

- Core ATG Content Administration Tables
- File Repository Tables
- Media Tables
- Versioning Tables
- User Profile Tables
- Workflow Tables
- View Mapping Tables

Note: You can access the ER diagrams for the ATG Content Administration database schema from the ATG documentation index page.

Core ATG Content Administration Tables

The following sections describe the database tables that are specific to ATG Content Administration core functionality.

epub_history

Contains information about history elements for projects.

Column	Data Type	Constraint
`history_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a history element	
`profile`	VARCHAR(40)	null
	The ID for the profile which created this history element	
`timestamp`	TIMESTAMP	null

Column	Data Type	Constraint
	The time the action in this history element occurred	
action	VARCHAR(255)	null
	The name of an action which this history element represents	
action_type	VARCHAR(255)	null
	The type of action which this history element represents	
description	WVARCHAR(4096)	null
	An associated description for this history element	

epub_his_act_parm

Contains information about action parameters for the history's action.

Column	Data Type	Constraint
history_id	VARCHAR(40)	not null
(primary key)	The ID of the history	
action_param	VARCHAR(255)	null
	The action parameters for the history's action	
sequence_num	NUMERIC(19)	not null
(primary key)	The sequence number which determines the ordering of the history element in relation to the other history elements in the same project	

epub_taskinfo

Contains information about the tasks associated with projects.

Column	Data Type	Constraint
taskinfo_id	VARCHAR(40)	not null
(primary key)	The ID of this task information item	

Column	Data Type	Constraint
version	BIGINT	not null
	The version of the task information item	
process_id	VARCHAR(40)	not null
	The ID of the process to which this item pertains	
process_name	WVARCHAR(255)	not null
	The name of the process whose task this item describes	
segment_name	WVARCHAR(255)	not null
	The name of the process segment whose task this item describes	
task_element_id	VARCHAR(255)	not null
	The element ID of the task this item describes	
acl	VARCHAR(2048)	null
	The Access Control List (ACL) assigned to the task at runtime	
priority	BIGINT	null
	The priority of the task	
owner_name	VARCHAR(255)	null
	The persona name of the owner of the task, or null if no owner is known.	
last_actor_name	VARCHAR(255)	null
	The persona name of the last known actor on the task.	
last_action_date	TIMESTAMP	null
	The date of the last known action on the task.	
last_outcome_id	VARCHAR(255)	null
	The element ID of the last applied outcome for the task.	

epub_agent_trnprt

Contains information about the transport of a target agent.

Column	Data Type	Constraint
transport_id	VARCHAR(40)	not null
	The unique ID for a transport.	
version	BIGINT	not null
	The version of the transport.	
transport_type	TINYINT	not null
	The type of transport. The only option currently available is RMI.	
jndi_name	VARCHAR(255)	null
	The name used by JNDI for the target's agent's transport.	
rmi_uri	VARCHAR(255)	null
	The URI to the RMI for the target agent's transport.	

epub_agent

Contains information about target agents

Column	Data Type	Constraint
agent_id	VARCHAR(40)	not null
(primary key)	The ID of the target agent.	
version	BIGINT	not null
	The version of the target agent.	
creation_time	TIMESTAMP	null
	The time that the target agent definition was created.	
display_name	WVARCHAR(255)	null
	The name of the target agent.	
description	VARCHAR(1024)	null
	A description of the target agent.	
main_agent_id	VARCHAR(40)	null

Column	Data Type	Constraint
	The main agent ID associated with the target. The agent ID identified in the `agent_id` column is a placeholder for the agent ID held in this column.	
`transport`	VARCHAR(40)	not null
	The ID of the transport that is described in the `epub_agent_trnprt` table.	

epub_target

Contains information about target sites.

Column	Data Type	Constraint
`target_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
`branch_name`	VARCHAR(255)	null
	The branch associated with the target.	
`snapshot_name`	VARCHAR(255)	null
	The name of the target's initial snapshot.	
`init_branch_name`	VARCHAR(255)	null
	The branch from which the target should be initialized.	
`version`	BIGINT	not null
	The version of the target.	
`creation_time`	TIMESTAMP	null
	The time the target definition was created.	
`main_target_id`	VARCHAR(40)	null
	The main target ID associated with the target definition. The target ID identified in the `target_id` column is a placeholder for the target ID held in this column.	
`display_name`	WVARCHER(255)	null
	The name of the target.	

Column	Data Type	Constraint
description	VARCHAR(1024)	null
	The description of the target.	
halted	TINYINT	null
	Indicates whether the target is accepting deployments.	
flag_agents	TINYINT	null
	Indicates whether the target accepts import (1) or full (0) deployment during initialization.	

epub_tr_dest

Stores deployment configuration information for the `targetDef` item type in the PublishingRepository.

This table is used by the `targetDef` item's `destinations` map property, whose key is a path to a source repository and whose value is a path to a destination repository.

Column	Data Type	Constraint
target_id	VARCHAR(40)	
(primary key)	The unique ID for a target.	
target_source	VARCHAR(100)	not null
(primary key)	The path to the source repository.	
target_destination	VARCHAR(100)	not null
	The path to the destination repository.	

epub_topology

Contains information about each deployment topology.

Column	Data Type	Constraint
topology_id	VARCHAR(40)	not null
(primary key)	The unique ID for a topology.	

Column	Data Type	Constraint
version	BIGINT	not null
	The version of the target associated with the topology.	
primary_tl	TINYINT	null
	Indicates whether this topology is the primary topology for the target.	

epub_tr_agents

Contains information about the agents for a target.

Column	Data Type	Constraint
target_id	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
agent_id	VARCHAR(40)	not null
(primary key)	The unique ID for an agent.	

epub_princ_asset

Contains information about the principal assets for a target agent.

Column	Data Type	Constraint
agent_id	VARCHAR(40)	not null
(primary key)	The unique ID for an agent.	
principal_assets	VARCHAR(40)	not null
(primary key)	The list of principal assets for the agent.	

epub_includ_asset

Contains information about the assets with included destinations associated with a target agent.

Column	Data Type	Constraint
agent_id	VARCHAR(40)	not null
(primary key)	The unique ID for a target agent.	
include_assets	VARCHAR(255)	not null
(primary key)	The assets that have included destinations for this target agent.	

epub_exclud_asset

Contains information about the assets that have excluded destinations associated with a target agent.

Column	Data Type	Constraint
agent_id	VARCHAR(40)	not null
(primary key)	The unique ID for a target agent.	
exclude_assets	VARCHAR(255)	not null
(primary key)	The assets that have excluded destinations for this target agent.	

epub_project

Contains information about projects.

Column	Data Type	Constraint
project_id	VARCHAR(40)	not null
(primary key)	Each project has a unique ID.	
version	BIGINT	not null
	The version of the project object.	
acl	VARCHAR(2048)	null
	The access control list for the project object.	
display_name	WVARCHAR(255)	null
	The name of the project object.	

Column	Data Type	Constraint
description	WVARCHAR(1024)	null
	A description of the project object.	
creator	VARCHAR(40)	null
	The profile ID of the user which created the project.	
workspace	VARCHAR(255)	not null
	The name of the workspace where this project exists.	
workflow_id	VARCHAR(40)	null
	The ID of the workflow that this project is using.	
checked_in	TINYINT	null
	Indicates whether this project is checked in (1) or checked out (0).	
editable	TINYINT	null
	Indicates whether this project is editable or read only.	
status	BIGINT	null
	The status of the project. Can be one of Active, Completed, Suspended, Error	
status_detail	WVARCHAR(255)	null
	Details about the status of the project.	
checkin_date	TIMESTAMP	null
	The date the project was checked in.	
creation_date	TIMESTAMP	null
	The creation date of the project.	
completion_date	TIMESTAMP	null
	The completion date of the project.	

epub_prj_targt_ws

Contains information about the workspaces associated with a project's target.

Column	Data Type	Constraint
project_id	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	
target_id	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
workspace_id	VARCHAR(40)	null
	The unique ID for a workspace.	

epub_pr_tg_status

Contains information about the status of a project associated with a target.

Column	Data Type	Constraint
project_id	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	
target_id	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
status_code	TINYINT	null
	The status code for the project. A project has one status per target. Options include: - approved (all tasks are complete and the project is configured for deployment) - accepted (approved and scheduled for deployment) - deployed (success deployment to a target site) - hidden (won't be deployed based on user preference)	

epub_prj_tg_snsht

Contains information about the snapshots for a project's target.

Column	Data Type	Constraint
project_id	VARCHAR(40)	not null

Column	Data Type	Constraint
(primary key)	The unique ID for a project.	
`target_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
`snapshot_id`	VARCHAR(40)	null
	The unique ID for a snapshot.	

epub_pr_tg_st_ts

Contains information about the creation time for a target project's snapshot.

Column	Data Type	Constraint
`project_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	
`target_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
`snapsht_creat_tm`	TIMESTAMP	null
	The time a snapshot was created for the project's target.	

epub_pr_tg_ap_ts

Contains information about the time a project's target was approved.

Column	Data Type	Constraint
`project_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	
`target_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a target.	
`approval_time`	TIMESTAMP	null
	The time the project's target was approved.	

epub_pr_history

Contains information about the history of projects.

Column	Data Type	Constraint
`project_id`	VARCHAR(40)	not null
(primary key)	The ID of the project to which the history element is associated.	
`sequence_num`	BIGINT	not null
(primary key)	The sequence number which determines the ordering of the history element in relation to the other history elements in the same project.	
`history`	VARCHAR(40)	not null
	The ID of the related history element in the project whose ID is in the `project_id` column.	

epub_process

Contains information about processes.

Column	Data Type	Constraint
`process_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a process.	
`version`	NUMERIC(19)	not null
	The version of the process.	
`acl`	VARCHAR(2048)	null
	The Access Control List for the process.	
`display_name`	WVARCHAR(255)	null
	The name of the process.	
`description`	WVARCHAR (1024)	null
	A description for the process.	

Column	Data Type	Constraint
creator	VARCHAR(40)	null
	The profile ID of the user who created the process.	
project	VARCHAR(40)	null
	The ID of the current project for this process.	
process_data	VARCHAR(40)	null
	A data object associated with the process.	
workflow_id	VARCHAR(40)	null
	The unique ID for the workflow that is managing this process.	
auto_deploy	TINYINT	null
	Indicates whether the project associated with this process was used for an import (1) or not (0).	
status	NUMERIC(19)	null
	The status of the process. Options include: Edit, Edit-Running, Running, Deployed, and Completed.	
status_detail	WVARCHAR(255)	null
	Additional information about the status.	
creation_date	TIMESTAMP	null
	The date the process was created.	
completion_date	TIMESTAMP	null
	The date the process was completed.	

epub_proc_prv_prj

Contains information about a process's completed projects.

Column	Data Type	Constraint
process_id	VARCHAR(40)	not null
(primary key)	The unique ID for a process.	
sequence_num	NUMERIC(19)	not null

(primary key)	The number that determines where a project ID is ordered in the context of all project IDs.	
`project_id`	VARCHAR(40)	not null
	The unique ID for a project.	

epub_proc_history

Contains information about a process's history.

Column	Data Type	Constraint
`process_id`	VARCHAR(40)	not null
(primary key)	The ID of the process to which the history element is associated.	
`sequence_num`	NUMERIC(19)	not null
(primary key)	The sequence number which determines the ordering of the history element in relation to the other history elements in the same process.	
`history`	VARCHAR(40)	not null
	The ID of the related history element in the process whose ID is in the `process_id` column.	

epub_proc_taskinfo

Contains information about a process's tasks.

Column	Data Type	Constraint
`id`	VARCHAR(40)	not null
	The unique ID for a process.	
`taskinfo_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a task.	

epub_deployment

Contains information about deployment scheduling.

Column	Data Type	Constraint
deployment_id	VARCHAR(40)	not null
(primary key)	The unique ID for a pending deployment.	
version	BIGINT	not null
	The version of the deployment.	
target_id	VARCHAR(255)	null
	The unique ID for the target associated with the deployment.	
deploy_time	TIMESTAMP	not null
	The scheduled time for the snapshot deployment.	
creation_time	TIMESTAMP	not null
	The time when the schedule was created.	
creator	VARCHAR(40)	null
	The name of the user who created the schedule.	
uri	VARCHAR(100)	null
	The RMI URI for the current deployment.	
next_dep_id	VARCHAR(40)	null
	The ID for the schedule that is next in the queue.	
previous_dep_id	VARCHAR(40)	null
	The ID for the previous schedule in the queue.	
force_full	TINYINT	null
	Indicates whether to execute a full deployment (1) or not (0).	
dep_type	TINYINT	null
	The type of deployment.	
status	NUMERIC(3)	null
	The status of the deployment. Options include `Error` and `Success`.	

Column	Data Type	Constraint
message_code	VARCHAR(255)	null
	The resource bundle key used for the error occurred during execution.	

epub_deploy_proj

Contains information about a project scheduled for deployment.

Column	Data Type	Constraint
deployment_id	VARCHAR(40)	not null
(primary key)	The unique ID for a pending deployment.	
project_id	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	
sequence_num	BIGINT	not null
	The sequence number used to determine where the project is ordered in the context of the deployment.	

epub_dep_err_parm

Contains information about the error parameters for an error message.

Column	Data Type	Constraint
deployment_id	VARCHAR(40)	not null
(primary key)	The unique ID for a pending deployment.	
error_param	VARCHAR(255)	null
	The error parameters defined for the deployment's error messages.	
sequence_num	BIGINT	not null
(primary key)	The sequence number used to determine where the error parameter is ordered in the context of the deployment.	

epub_dep_log

Contains information about all executed deployments.

Column	Data Type	Constraint
log_id	VARCHAR(40)	not null
(primary key)	The unique ID for a deployment log.	
dep_id	VARCHAR(40)	not null
	The unique ID for a deployment.	
target_name	VARCHAR(255)	not null
	The name of the target associated with the deployment.	
log_time	TIMESTAMP	not null
	The time the log was created.	
begin_time	TIMESTAMP	not null
	The time the deployment began execution.	
end_time	TIMESTAMP	not null
	The time the deployment finished execution.	
actor_id	VARCHAR(40)	null
	The ID for the user who performed the operation.	
type	INTEGER	not null
	Indicates whether the deployment is full (1) or incremental (2).	
dep_mode	INTEGER	not null
	The mode used for the deployment. Options include switch and online.	
status	INTEGER	not null
	The status of the deployment. Options include `Error`, `Cancel`, and `Success`.	
deli_proj_ids	VARCHAR(255)	null
	The IDs for deployed projects.	
delimiter	VARCHAR(5)	not null

Column	Data Type	Constraint
	The delimiter used to separate project IDs in the `deli_proj_ids` column.	

epub_process_data

Contains information about process data.

Column	Data Type	Constraint
`asset_version`	NUMERIC(19)	not null
(primary key)	The 1-based counter that specifies the version of the asset.	
`branch_id`	VARCHAR(40)	not null
	The ID for the branch where the asset exists.	
`is_head`	TINYINT	not null
	Indicates whether the asset is the first asset (1) in the branch.	
`version_deleted`	NUMERIC(1)	null
	Indicates this version of the asset is deleted (1) from the branch.	
`version_editable`	NUMERIC(1)	null
	Indicates whether this version of the asset is editable (1) or read-only (0).	
`pred_version`	NUMERIC(19)	null
	The ID for the previous version of the asset.	
`checkin_date`	TIMESTAMP	null
	The date when the item was checked in and this version was created.	
`process_data_id`	VARCHAR(40)	not null
(primary key)	The unique ID for the process data item.	
`type`	NUMERIC(19)	not null
	The item descriptor associated with the process data item.	

File Repository Tables

This section describes the database tables that store information about items in the `/atg/epub/file/PublishingFileRepository`, which stores your application's versioned file assets.

epub_file_folder

Stores information about file folder items.

Column	Data Type	Constraint
`asset_version`	NUMERIC(19)	not null
(primary key)	The 1-based counter that specifies the version of the asset.	
`workspace_id`	VARCHAR(40)	not null
	The ID of the workspace where the asset version was initially created.	
`branch_id`	VARCHAR(40)	not null
	The ID for the branch containing the asset.	
`is_head`	TINYINT	not null
	Indicates whether the asset is the first asset (1) in the branch.	
`version_deleted`	NUMERIC(1)	null
	Indicates whether the asset version is a deleted version.	
`version_editable`	NUMERIC(1)	null
	Indicates whether the asset version is an editable version. That is, the version is a working version in a workspace, where modifications to it can be made.	
`pred_version`	NUMERIC(19)	null
	The asset version upon which this version was based. For example, if you create version 2 by checking out version 1, version 2's predecessor version is version 1.	
`checkin_date`	TIMESTAMP	null

Column	Data Type	Constraint
	The date the item was checked in.	
folder_id	VARCHAR(40)	not null
(primary key)	The ID of the file folder.	
acl	VARCHAR(2048)	null
	The ACL (Access Control List) for the file folder.	
folder_name	WVARCHAR(255)	not null
	The name of the file folder.	
parent_folder	VARCHAR(40)	null
	The parent folder that contains this file folder.	

epub_file_asset

Stores information about file assets.

Column	Data Type	Constraint
asset_version	NUMERIC(19)	not null
(primary key)	1-based counter that specifies the version of the asset	
workspace_id	VARCHAR(40)	not null
	The ID of the workspace where the asset version was initially created.	
branch_id	VARCHAR(40)	not null
	The ID for the branch containing the asset	
is_head	TINYINT	not null
	Indicates whether the asset is the first asset (1) in the branch.	
version_deleted	NUMERIC(1)	null
	Indicates whether the asset version is a deleted version.	
version_editable	NUMERIC(1)	null

Column	Data Type	Constraint
	Indicates whether the asset version is an editable version. That is, the version is a working version in a workspace, where modifications to it can be made.	
pred_version	NUMERIC(19)	null
	The asset version upon which this version was based. For example, if you create version 2 by checking out version 1, version 2's predecessor version is version 1.	
checkin_date	TIMESTAMP	null
	The date the asset was checked in.	
file_asset_id	VARCHAR(40)	not null
(primary key)	The ID of the file asset.	
type	NUMERIC(19)	not null
	The asset type of the file asset.	
acl	VARCHAR(2048)	null
	The ACL (Access Control List) for the file.	
filename	WVARCHAR(255)	not null
	The name of the file asset.	
last_modified	TIMESTAMP	null
	The date and time the file asset was last modified.	
size_bytes	NUMERIC(19)	null
	The size of the asset, in bytes.	
checksum	BIGINT	null
	The 64-bit checksum value for the asset, calculated when the file asset is saved or imported.	
parent_folder	VARCHAR(40)	null
	The parent folder that contains the file asset.	

epub_text_file

Stores information about text file assets.

Column	Data Type	Constraint
asset_version	NUMERIC(19)	not null
(primary key)	The 1-based counter that specifies the version of the asset.	
text_file_id	VARCHAR(40)	not null
(primary key)	The ID of the text file asset.	
text_content	LONG VARCHAR	null
	The contents of the text file asset.	

epub_binary_file

Stores information about binary file assets.

Column	Data Type	Constraint
asset_version	NUMERIC(19)	not null
(primary key)	The 1-based counter that specifies the version of the asset.	
binary_file_id	VARCHAR(40)	not null
(primary key)	The ID of the binary file asset.	
binary_content	LONG VARBINARY	null
	The contents of the binary file asset.	

Media Tables

This section describes the database tables related to storing media in the ATG Content Administration system.

epub_folder

Contains information about ATG Content Administration folders.

Column	Data Type	Constraint
folder_id	VARCHAR(40)	not null
(primary key)	Each folder has a unique ID.	
version	NUMERIC(19)	not null
	The version of the folder object.	
creation_date	TIMESTAMP	null
	The creation date of the folder.	
start_date	TIMESTAMP	null
	The start date which the folder becomes active.	
end_date	TIMESTAMP	null
	The end date which the folder is no longer active.	
name	WVARCHAR(255)	not null
	The name of the folder object.	
description	WVARCHAR(255)	null
	A description of the folder object.	
path	WVARCHAR(255)	not null
	A relative path of where the folder is located in a hierarchy.	
parent_folder_id	VARCHAR(40)	null
	The parent of this folder.	

epub_media

Contains information about media.

Column	Data Type	Constraint
media_id	VARCHAR(40)	not null
(primary key)	Each media item has a unique ID.	
version	NUMERIC(19)	not null
	The version of the media item.	

Column	Data Type	Constraint
creation_date	TIMESTAMP	null
	The creation date of the media item.	
start_date	TIMESTAMP	null
	The start date which the media item becomes active.	
end_date	TIMESTAMP	null
	the end date which the media item is no longer active.	
name	WVARCHAR(255)	not null
	The name of the media item.	
description	WVARCHAR(255)	null
	A description of the media item.	
path	WVARCHAR(255)	not null
	a relative path of where the media item is located in a hierarchy.	
parent_folder_id	VARCHAR(40)	not null
	The parent of this media item.	
media_type	NUMERIC(19)	null
	The type of the media item.	

epub_media_ext

Contains information about media external data.

Column	Data Type	Constraint
media_id	VARCHAR(40)	not null
(primary key)	Each media item has a unique ID.	
url	WVARCHAR(255)	not null
	The URL of the media item.	

epub_media_bin

Contains information about media binary data.

Column	Data Type	Constraint
media_id	VARCHAR(40)	not null
(primary key)	Each media item has a unique ID.	
length	NUMERIC(19)	not null
	The length of the binary data.	
last_modified	TIMESTAMP	not null
	The last modified date of the binary data.	
data	VARBINARY(1048576)	not null
	The binary data for the media item.	

epub_media_txt

Contains information about media text data.

Column	Data Type	Constraint
media_id	VARCHAR(40)	not null
(primary key)	Each media item has a unique ID.	
length	NUMERIC(19)	not null
	The length of the text data.	
last_modified	TIMESTAMP	not null
	The last modified date of the text data.	
data	LONG WVARCHAR	not null
	The text data for the media item.	

Versioning Tables

The following sections describe the database tables that store versioning information used by ATG Content Administration.

avm_devline

Contains base type information for all branches, snapshots and workspaces.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID of the development line.	
type	NUMERIC(19)	not null
	The type of development line: 0 for branch, 1 for snapshot, 2 for workspace	
name	VARCHAR(255)	null
	Development line name.	
parent	VARCHAR(40)	null
	Parent development line from which this one was created.	
date_created	TIMESTAMP	null
	Date and time of creation.	

avm_workspace

Contains information about workspace development lines.

Column	Data Type	Constraint
ws_id	VARCHAR(40)	not null
(primary key)	Unique id of the workspace.	
checked_in	NUMERIC(1)	not null
	Whether the workspace is checked in.	
ci_time	TIMESTAMP	null

Column	Data Type	Constraint
	When the was workspace checked in.	
userid	VARCHAR(255)	null
	Who checked in the workspace.	
locked	NUMERIC(1)	not null
	Indicates whether the assets in the workspace are locked (1) or not (0).	
change_was	VARCHAR(4096)	null
	The change made using the workspace.	
asset_uri_cache	LONG VARBINARY	null
ref_ast_uri_cache	LONG VARBINARY	null

avm_asset_lock

Contains information about assets that are locked for deployment.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for the lock.	
repository_name	VARCHAR(255)	not null
	The name of the repository for which the asset is a part.	
descriptor_name	VARCHAR(255)	not null
	The item descriptor name associated with the asset.	
repository_id	VARCHAR(255)	not null
	The asset's RepositoryID.	
workspace_id	VARCHAR(40)	not null
	The unique ID for the workspace.	

User Profile Tables

The following sections describe tables that are used for the user profile extensions needed by ATG Content Administration.

epub_user

Contains information about users within ATG Content Administration.

Column	Data Type	Constraint
`user_id`	VARCHAR(40)	not null
(primary key)	Each user has a unique ID.	
`title`	VARCHAR(255)	null
	The title of the user.	
`expert`	NUMERIC(1)	null
	Indicates whether to display expert properties.	
`def_listing`	NUMERIC(19)	null
	The `defaultlisting` to use in batch page display.	
`def_ip_listing`	NUMERIC(19)	null
	The `defaultlisting` to use in batch in page property display.	
`allow_applets`	NUMERIC(1)	null
	Indicates whether to allow use of applets in browsers.	

epub_prj_hist

Contains information about the completed projects a particular user worked on.

Column	Data Type	Constraint
`user_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a user.	
`sequence_num`	NUMERIC(19)	not null

	The sequence number that places the project ID in an order within the context of the user's projects.	
`project_id`	VARCHAR(40)	not null
(primary key)	The unique ID for a project.	

Workflow Tables

The following sections describe the database tables related to the workflow processes. Collective workflow process instances represent all projects going through the workflow process.

epub_coll_workflow

Contains information about collective workflow instances.

Column	Data Type	Constraint
`id`	VARCHAR(40)	not null
(primary key)	Unique ID.	
`workflow_name`	VARCHAR(255)	null
	Workflow process name.	
`modification_time`	NUMERIC(19)	null
	When the workflow process was last modified.	
`segment_name`	VARCHAR(255)	null
	Workflow process segment name.	
`creator_id`	VARCHAR(40)	null
	ID of collective process instance which created this instance.	
`state`	VARCHAR(16)	null
	The instance's current process state machine state.	
`num_retries`	INTEGER	null
	Number of current transition retries.	

epub_ind_workflow

Contains information about individual workflow instances. Each of these instances represents a single subject (project) going through the workflow process.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
modification_time	NUMERIC(19)	null
	When the workflow process was last modified.	
segment_name	VARCHAR(255)	null
	Workflow process segment name.	
creator_id	VARCHAR(40)	null
	ID of collective process instance which created this instance.	
state	VARCHAR(16)	null
	The instance's current process state machine state.	
process_id	VARCHAR(40)	not null
	The ID of the process.	
num_retries	INTEGER	null
	Number of current transition retries.	

epub_workflow_strs

Contains information about string context variables. String context variables are associated with individual workflow process instances.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	

tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_str	VARCHAR(255)	null
	String variable value.	

epub_workflow_bls

Contains information about boolean context variables associated with individual workflow process instances.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	
tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_bool	NUMERIC(1)	null
	Boolean variable value.	

epub_workflow_lngs

Contains information about long context variables associated with individual workflow process instances.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	
tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_long	NUMERIC(19)	null
	Long variable value.	

epub_workflow_dbls

Contains information about double context variables associated with individual workflow process instances.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	
tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_dbl	NUMERIC(15) scale=4	null
	Double variable valued.	

epub_workflow_dats

Contains information about data context variables associated with individual workflow process instances.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	
tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_date	TIMESTAMP	null
	Date variable value.	

epub_workflow_ris

Contains information about the context variables for a Repository item.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for a workflow instance.	

Column	Data Type	Constraint
tag	VARCHAR(25)	not null
(primary key)	The name of the variable.	
context_item	VARCHAR(25)	null
	The reference to a Repository item.	

epub_workflow_vfs

Contains information about the context variables for a virtual file.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the associated individual workflow instance.	
tag	VARCHAR(25)	not null
(primary key)	Variable name.	
context_file	VARCHAR(255)	null
	The reference to a virtual file.	

epub_workflow_info

Contains information about workflow definitions. Each workflow info corresponds to a workflow process definition created using the ACC.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
workflow_status	INTEGER	null
	Workflow status: 1 (disabled), 2 (running).	
modification_time	NUMERIC(19)	null

Column	Data Type	Constraint
	When the workflow process was last modified.	
creation_time	NUMERIC(19)	null
	When the workflow process was created.	
author	VARCHAR(25)	null
	Who created the workflow process.	
last_modified_by	VARCHAR(25)	null
	Who last modified the workflow process.	
psm_version	INTEGER	null
	Process state machine version.	
wdl	LONG VARBINARY	null
	Contains the WDL bytes which define the workflow process.	

epub_wf_mig_info

Contains workflow migration information.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_info_id	VARCHAR(40)	not null
	Workflow info ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
modification_time	NUMERIC(19)	null
	Modification time at the time of migration.	
psm_version	INTEGER	null
	Process state machine version at the time of migration.	
wdl	LONG VARBINARY	null

Column	Data Type	Constraint
	Workflow process definition at the time of migration.	
migration_status	INTEGER	null
	Migration status: 1 (in Progress), 2 (done).	

epub_wf_mg_inf_seg

Contains segment names of workflow migration infos.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the migration info.	
idx	INTEGER	not null
(primary key)	Segment's index in the list.	
segment_name	VARCHAR(255)	null
	Workflow process segment name.	

epub_wf_templ_info

Contains information about workflow template definitions. Each of these infos corresponds to a workflow template process definition created using the ACC.

Column	Data Type	Constraint
Id	VARCHAR(40)	not null
(primary key)	Unique ID.	
template_name	VARCHAR(255)	null
	Workflow template name.	
modification_time	NUMERIC(19)	null
	When the workflow template was last modified.	
creation_time	NUMERIC(19)	null
	When the workflow template was created.	

Column	Data Type	Constraint
author	VARCHAR(25)	null
	Who created the workflow template.	
last_modified_by	VARCHAR(25)	null
	Who last modified the workflow template.	
wdl	LONG VARBINARY	null
	Contains the WDL bytes which define the workflow template.	

epub_wf_coll_trans

Contains information about pending collective transitions associated with workflow processes.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
modification_time	NUMERIC(19)	null
	When the workflow process was last modified.	
server_id	VARCHAR(40)	null
	Workflow process manager server responsible for completing this collective transition.	
event_type	VARCHAR(255)	null
	Type of the JMS message which triggered this collective transition.	
segment_name	VARCHAR(255)	null
	Workflow process segment name.	
state	VARCHAR(16)	null
	Current process state machine state of the workflow instances participating in the transition.	
coll_workflow_id	VARCHAR(40)	null

Column	Data Type	Constraint
	ID of the collective instance taking the transition.	
step	INTEGER	null
	Current transition step.	
current_count	INTEGER	null
	Current transition count.	
last_query_id	VARCHAR(40)	null
	Current transition last query ID.	
message_bean	LONG VARBINARY	message_bean
	Message bean of the JMS message which triggered this collective transition.	

epub_wf_ind_trans

Contains information about pending individual transitions associated with workflow processes.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
modification_time	NUMERIC(19)	null
	When the workflow process was last modified.	
server_id	VARCHAR(40)	null
	Workflow process manager server responsible for completing this individual transition.	
event_type	VARCHAR(255)	null
	Type of the JMS message which triggered this individual transition.	
segment_name	VARCHAR(255)	null
	Workflow process segment name.	

Column	Data Type	Constraint
state	VARCHAR(16)	null
	Current process state machine state of the workflow instances participating in the transition.	
last_query_id	VARCHAR(40)	null
	Current transition last query ID.	
message_bean	LONG VARBINARY	
	Message bean of the JMS message which triggered this individual transition.	

epub_wf_deletion

Contains information about pending deletions associated with workflow processes.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
modification_time	NUMERIC(19)	null
	When the workflow process was last modified.	

epub_wf_del_segs

Contains the segment names of workflow's pending deletions.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the pending deletion.	
idx	INTEGER	not null
(primary key)	Segment's index in the list.	

Column	Data Type	Constraint
segment_name	VARCHAR(255)	null
	Workflow process segment name.	

epub_wf_migration

Contains information about a workflow's pending migration.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	Unique ID.	
workflow_name	VARCHAR(255)	null
	Workflow process name.	
old_mod_time	NUMERIC(19)	null
	Modification time workflow process is being migrated from.	
new_mod_time	NUMERIC(19)	null
	Modification time workflow process is being migrated to.	
migration_info_id	VARCHAR(40)	not null
	Workflow migration info ID.	

epub_wf_mig_segs

Contains segment names of a workflow's pending migrations.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	ID of the pending migration.	
idx	INTEGER	not null
(primary key)	Segment's index in the list.	

Column	Data Type	Constraint
segment_name	VARCHAR(255)	null
	Workflow process segment name.	

epub_wf_server_id

Contains information on the workflow process manager classification. The table keeps track of how the various workflow process manager servers are classified.

Column	Data Type	Constraint
server_id	VARCHAR(40)	not null
(primary key)	Workflow process manager server ID.	
server_type	INTEGER	not null
	Workflow process manager server type: 0 (workflow editor), 1 (global), 2 (individual).	

View Mapping Tables

The following sections describe tables that contain information in support of the view mapping feature.

vmap_im

Contains information about an item.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for an item's mapping.	
name	VARCHAR(64)	null
	The name of the item mapping.	
description	VARCHAR(1024)	null
	A description of the item mapping.	

Column	Data Type	Constraint
item_path	VARCHAR(256)	not null
	The path to the item component.	
item_name	VARCHAR(64)	not null
	The name of the item.	
is_readonly	NUMERIC(1)	null
	Indicates whether the item is ready-only (1) or editable (0).	
form_handler	INT	null
	The Id for the item descriptor associated with the form handler.	
mode_id	VARCHAR(40)	not null
	The view mapping mode. Options include: edit, view, and browse.	

vmap_fh

Contains information about form handlers associated with view mapping.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for a form handler.	
name	VARCHAR(64)	not null
	The name of the form handler.	
description	VARCHAR(2048)	null
	A description of the form handler.	
component_path	VARCHAR(1024)	null
	The nucleus path to the form handler component.	

vmap_mode

Contains information about view mapping modes.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for a view mapping mode.	
name	VARCHAR(64)	not null
	The name of the view mapping mode.	
description	VARCHAR(1024)	null
	A description for the view mapping mode.	
fallback_id	VARCHAR(40)	null
	A default mode to use when this mode is not available.	

vmap_ivm

Contains information about item views.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	A unique ID for an item view.	
name	VARCHAR(64)	null
	The name of the item view.	
display_name	WVARCHAR(64)	not null
	The display name for the item view.	
description	VARCHAR(1024)	null
	A description for the item view.	
view_id	VARCHAR(4o)	null
	A unique ID for an item view.	

vmap_im2ivm_rel

Contains information about item mappings and associated item view mappings.

Column	Data Type	Constraint
item_id	VARCHAR(40)	not null
(primary key)	A unique ID for an item mapping.	
sequence_num	INTEGER	not null
(primary key)	The sequence number that places the item view in an order within the context of all item views for the item mapping.	
view_id	VARCHAR(40)	not null
	A unique ID for an item view mapping.	

vmap_iv

Contains information about an item view.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	A unique ID for an item view.	
name	WVARCHAR(64)	not null
	The name of the item view.	
description	WVARCHAR(1024)	not null
	A description of the item view.	
uri	VARCHAR(255)	null
	The URI to an associated JSP.	
app_name	VARCHAR(255)	null
	The name of the application where the JSP lives.	
mode_id	VARCHAR(40)	not null
	The item view mode. Options include: edit, view, and browse.	

vmap_pv

Contains information about property views.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	A unique ID for a property view.	
name	WVARCHAR(64)	not null
	The name of a property view.	
type	VARCHAR(128)	not null
	The type of property view.	
description	WVARCHAR(1024)	null
	A description of the property view.	
uri	VARCHAR(255)	null
	The URI to an associated JSP.	
app_name	VARCHAR(255)	null
	The name of the application where the JSP lives.	
mode_id	VARCHAR(40)	not null
	The property view mode. Options include: `edit`, `view`, and `browse`.	
is_default	NUMERIC(1)	null
	Indicates whether the property view is the default view for this property type.	
is_readonly	NUMERIC(1)	null
	Indicates whether the property view is read-only (1) or editable (0).	
is_component	NUMERIC(1)	null
	The type of component. This column is populated for property views that are editors for a collection.	

vmap_ivm2pvm_rel

Contains information about item view mappings and associated property view mappings.

Column	Data Type	Constraint
ivm_id	VARCHAR(40)	not null
	The unique ID for an item view mapping.	
pvm_id	VARCHAR(40)	not null
	The unique ID for a property view mapping.	
name	VARCHAR(64)	not null
	The name of the property.	

vmap_pvm

Contains information about property view mappers.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for a property view mapping.	
description	VARCHAR(1024)	null
	A description of the property view mapping.	
pview_id	VARCHAR(40)	null
	The ID for an associated property view.	
cpview_id	VARCHAR(40)	null
	The ID for an associated component property view.	

vmap_attrval

Contains information about an attribute value.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for an attribute value.	
attr_value	VARCHAR(2048)	null

Column	Data Type	Constraint
	The value of an attribute.	

vmap_attrval_rel

Contains information about mapping items and associated attribute value items.

Column	Data Type	Constraint
mapper_id	VARCHAR(40)	not null
(primary key)	The unique ID for an item mapping, item view mapping, or property view mapping.	
attribute_id	VARCHAR(40)	not null
(primary key)	The unique ID for an attribute value.	
name	VARCHAR(64)	not null
	The name of the attribute.	

vmap_cattrval_rel

Contains information about mapping items and associated component attribute value items.

Column	Data Type	Constraint
mapper_id	VARCHAR(40)	not null
(primary key)	The unique ID for an item mapping, item view mapping, or property view mapping.	
attribute_id	VARCHAR(40)	not null
(primary key)	The unique ID for an attribute value.	
name	VARCHAR(64)	not null
	The name of the attribute.	

vmap_iv2ivad_rel

Contains information about item views and associated item view attribute definitions.

Column	Data Type	Constraint
view_id	VARCHAR(40)	not null
(primary key)	The unique ID for an item view.	
attr_id	VARCHAR(40)	not null
	The ID for the item view attribute definition mapped to the item view.	
name	VARCHAR(64)	not null
(primary key)	The name of the item view attribute definition.	

vmap_ivattrdef

Contains information about item view attribute definitions.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for an item view attribute definition.	
description	VARCHAR(1024)	null
	A description of the item view attribute definition.	
default_value	VARCHAR(1024)	null
	The attribute's default value.	

vmap_pv2pvad_rel

Contains information about property views and associated property view attribute definitions.

Column	Data Type	Constraint
view_id	VARCHAR(40)	not null
	The unique ID for a property view.	
attr_id	VARCHAR(40)	not null
	The ID for a property view attribute definition.	

Column	Data Type	Constraint
name	VARCHAR(64)	not null
	The name of the property view attribute definition.	

vmap_pvattrdef

Contains information about property view attribute definitions.

Column	Data Type	Constraint
id	VARCHAR(40)	not null
(primary key)	The unique ID for a property view attribute definition.	
description	VARCHAR(1024)	null
	A description for the property view attribute definition.	
default_value	VARCHAR(1024)	null
	The default value for the attribute.	

Appendix B: Virtual File Systems

A virtual file system (VFS) is a virtual container that organizes a set of file-like entities into a tree structure. As such, virtual file systems play an important role in both the management of file assets in the content development environment and their deployment to a target (staging or production) environment.

Choosing a VFS Implementation

In the content development environment, all VFSs must be instances of ContentRepositoryVFSService, which provides support for managing versioned file assets. However, there are several options for your target environment. The implementation you should use depends primarily on the mode of deployment you require, and secondarily, on the type of assets to deploy to the VFS:

Deployment Mode	VFS Implementation
switch	Use a SwitchableLocalFileSystem. If you do not require journaling, you can disable it. **Note:** The deployment of JSPs in switch mode is not supported.
online	If you are deploying JSPs to the target VFS, use a SelectiveDeleteVFSService. If you are deploying personalization or scenario assets to the VFS, use a JournalingFileSystemService. The virtual file systems that contain these assets require support for journaling because other services must examine the journal to identify specific types of updates (so the assets can be activated by the appropriate subsystems). For all other asset types, use a LocalVFSService.

This appendix provides reference information on the VFS implementations you can use with ATG Content Administration:

- ContentRepositoryVFSService
- SwitchableLocalFileSystem
- SelectiveDeleteVFSService
- JournalingFileSystemService
- LocalVFSService

ContentRepositoryVFSService

The `atg.vfs.repository.ContentRepositoryVFSService` class is a VFS implementation that sits on top of a content repository and exposes the content items as virtual files. It provides the necessary support for managing file assets in a versioned environment.

This VFS implementation is used in the content development environment to expose the file assets in the `/atg/epub/file/PublishingFileRepository` as virtual files for use in various contexts, such as deployment. One `ContentRepositoryVFSService` instance should exist for each asset destination in your deployment targets to which file assets are to be deployed.

You can configure the following properties of a `ContentRepositoryVFSService`:

Property	Description
`contentRepository`	The `ContentRepository` that contains the items to expose as virtual files. This property must be set to: `/atg/epub/file/SecuredPublishingFileRepository` This is the secured repository that sits on top of and controls access to the `/atg/epub/file/PublishingFileRepository`.
`itemDescriptorNames`	A comma-separated list of the names of the item descriptors that are viewable and accessible through the VFS. A null value exposes all item types. **Note:** Each item descriptor can be exposed via exactly one VFS in the content development environment. In other words, an item descriptor specified in this property for a given VFS cannot be specified in this property for any other VFS.
`mutableFolderDescriptorName`	The item descriptor in the repository specified in the `contentRepository` property that represents a folder in the content repository. A null value uses the first item descriptor. Always set this property to `fileFolder`, as this is the item descriptor that represents a folder in the PublishingFileRepository.

SwitchableLocalFileSystem

The `atg.vfs.switchable.SwitchableLocalFileSystem` class is a VFS implementation that switches between two underlying local file systems and keeps a journal of changes made. Instances of this class are

used to configure default target VFSs for switch deployment (ConfigFileSystem and WWWFileSystem) and also to configure custom target VFSs for switch deployment.

Property	Description
`liveDirectory`	The live directory that contains the files used by external facilities. No other VFS on the server can use this directory. See liveDirectory/stagingDirectory Constraints below this table.
`stagingDirectory`	The inactive directory where file updates are made before performing a switch. No other VFS on the server can use this directory. See liveDirectory/stagingDirectory Constraints below this table.
`dataDirectory`	A dynamically-created directory that stores internal data used during deployments. No other VFS on the server can use this directory.
`journaling`	Indicates whether to write VFS modifications to a journal. **Note:** This property is used only by a switchable ConfigFileSystem VFS for the purpose of updating personalization and scenario assets on a target.
`updateListeners`	The list of event listeners that listen for events that indicate the VFS is updated in a deployment. **Note:** This property is used only by a switchable ConfigFileSystem VFS for the purpose of updating personalization and scenario assets on a target.
`name1`	The logical name of the underlying file system to use as the live VFS at initial application startup. This property is analogous to the `initialDataSourceName` property of a SwitchingDataSource used by a repository. All switchable VFSs in the target must specify the same name in this property. The name must match the name in the `initialDataSourceName` property of all SwitchingDataSource instances used by target repositories.
`name2`	The logical name of the underlying file system to be used as the inactive VFS at initial application start-up. All switchable VFSs in the target must specify the same name in this property.

Property	Description
checksumCacheEnabled	Turns on checksum caching for this VFS, which can be used to improve deployment times for systems where large numbers of file assets are being deployed. See Cache Checksums for File Assets for more information on the checksum caching properties in this service.
checksumCacheEncoding	Used if checksum caching is enabled. Specifies the encoding used to save the cache data (UTF-8, by default). See Cache Checksums for File Assets for more information.
deleteInBackground	Enables background deletion if set to true. By default, this property is set to false (see Background Deletion of File System Assets).
deletionThreadPriorityDelta	Sets the deletion thread's priority. In general, you should let the system set thread priority, and omit this property
deletionThreadDelay	Specifies in milliseconds how long the deletion thread waits before it begins to delete files from the temporary directory. If set to 0 (the default), deletion begins immediately.

liveDirectory/stagingDirectory Constraints

The following constraints apply:

- Each `liveDirectory` and `stagingDirectory` directory must be reserved for the exclusive use of a single VFS.
- `liveDirectory` and `stagingDirectory` directories must be on the same UNIX partition.
- ConfigFileSystem only: The live directory must be located on the application's configuration path.
- WWWFileSystem only:
  - The live directory must be located in the Web server's document root directory or one of its subdirectories.
  - The live directory must not store files used by other services; it should contain only files deployed from the asset management server's WWWFileSystem.

Data Sources

Collectively, the `name1` and `name2` properties are analogous to the `dataSources` property of a SwitchingDataSource used by a repository, as they identify the underlying data stores used by the switchable data store—in this case, a switchable VFS. The labels used in these properties must correspond to those used for each SwitchingDataSource.

For example, if you configured each SwitchingDataSource as follows:

```
dataSources=\
  Red=/atg/dynamo/service/jdbc/Pool1,\
  Blue=/atg/dynamo/service/jdbc/Pool2
```

Then the `name1` and `name2` properties in each SwitchableLocalFileSystem must be configured as follows:

```
name1=Red
name2=Blue
```

The `/atg/epub/DeploymentServer` uses the logical names for all underlying data stores (for all repositories and VFSs) to verify that all agents in the target performed each switch deployment successfully, or, more specifically, to verify that all agents are using the same underlying store as their live store and likewise the same underlying store as their staging store.

Read-only properties

A SwitchableLocalFileSystem also has two read-only properties that identify the current live and staging directories: `liveDataStoreName` and `stagingDataStoreName`, respectively. These properties indicate the current state of the VFS.

SelectiveDeleteVFSService

The `atg.service.vfs.SelectiveDeleteVFSService` class is a VFS implementation for a local file system that supports selective deletion of files. This VFS implementation must be used in Web application staging and production environments where you deploy JSPs (see Configure JSP File Asset Support).

Because a Web application can contain other files besides JSPs such as servlets, the VFS must provide a mechanism to indicate explicitly which files in the local file system can be deleted during a deployment. This is essential for a full deployment, which otherwise deletes all files in the asset destination before writing the new files. Also, a deployment that explicitly deletes a file can succeed only if `SelectiveDeleteVFSService` makes the file available for deletion.

You can configure the following properties of a `SelectiveDeleteVFSService`:

Property	Description
`localDirectory`	The root folder of the Web application—for example, `{atg.dynamo.root}/Publishing/WebAppRef/j2ee-apps/webappref`. This is the exploded directory where Web application JSPs should be deployed from the asset management server.
`fileExtensions`	The list of extensions for the file types that can be deleted when the file system is updated by a deployment. Typically, this property is set to `jsp,jspf`. If this property is empty, no files can be deleted.

`filePaths`	If specified, exposes for deletion only those files with the specified file extension that are also on one of the specified paths. `filePaths` can specify one or more comma-delimited file paths as follows: `filePaths=`*`file-path`*`[,`*`file-path`*`]...`

Used in combination, `fileExtensions` and `filePaths` provide the following options for determining which files are exposed for deletion:

Property settings	Files to delete
`fileExtensions=`*`empty`*	None
`fileExtensions=`*`ext`* `filePaths=`*`empty`*	All files with the specified extension
`fileExtensions=`*`ext`* `filePaths=`*`path`*	All files on the specified path with the *`ext`* extension.

JournalingFileSystemService

The `atg.vfs.journal.JournalingFileSystemService` class is a VFS implementation that wraps an underlying local VFS and keeps a journal of changes made to it.

This VFS implementation is used in your staging and production targets if you are using ATG Content Administration to manage your personalization and scenario assets. These assets are deployed to each `/atg/epub/file/ConfigFileSystem` (class `atg.vfs.journal.JournalingFileSystemService`) in a target. See Repositories for more information.

You can configure the following properties of a `JournalingFileSystemService`:

Property	Description
`virtualFileSystem`	The underlying local VFS for which to keep a journal of changes.
`journalDirectory`	The directory in which to create the journal file. If the directory does not exist, it is created.
`updateListeners`	The list of event listeners that should be notified when the VFS is updated. Listeners can iterate through the journal to identify specific types of changes and act accordingly.
`clearJournalOnUpdate`	Indicates whether to clear the data in the journal after all listeners are notified that the VFS is updated.

LocalVFSService

The `atg.service.vfs.LocalVFSService` class is a VFS implementation for a local file system.

This VFS implementation is used in your staging and production environments if you configure your targets for *online* mode deployments performed (see Deployment Modes).

You can configure the following properties of a `LocalVFSService`:

Property	Description
`localDirectory`	The local directory to hold the file system (created on demand)—for example, `{atg.dynamo.home}/doc`. This is the directory to which the files should be deployed from the asset management server. Make sure the directory does not store files used by other services; it should contain only files deployed from the VFS at the exact same Nucleus location on the asset management server.
`checksumCacheEnabled`	Turns on checksum caching for this VFS, which can be used to improve deployment times for systems where large numbers of file assets are being deployed. See Cache Checksums for File Assets for more information on the checksum caching properties in this service.
`checksumCacheEncoding`	Used if checksum caching is enabled. Specifies the encoding used to save the cache data (UTF-8, by default).
`checksumCacheDataFile`	Specifies the directory to be used to store the checksum cache data.

Appendix C: Form Handlers

This appendix describes ATG Content Administration form handler classes for which components are included. Each class section explains the configuration and navigational properties and submit handler methods defined in it. Some form handlers have pre and post methods, which are methods defined to execute directly before or after a given submit handler method. Pre and post methods are described in this appendix as well.

Keep in mind that only properties and methods defined specifically in these classes are mentioned. A class without a Navigational Properties section, for example, does not have any navigational properties specified in the class itself, although it might inherit such properties from a parent class. Inherited properties and methods are not included in this discussion. For a comprehensive listing of the elements in each class, see the *ATG API Reference*.

This section describes the following ATG Content Administration form handlers.

AddNoteFormHandler

The `atg.epub.servlet.AddNoteFormHandler` class lets you save a user's entry as a note attached to a project. ATG Content Administration includes one component of this class called `/atg/epub/servlet/AddNoteFormHandler`.

Configuration Properties

The AddNoteFormHandler has the following configuration properties:

Property	Function
note	Holds a user's entry.
processId	Holds the ID for the process where you want to add a note.
projectId	Holds the ID for the project where you want to add a note.

Submit Handler Method

The AddNoteFormHandler has the following submit handler methods:

Method	Function
handleAddNote	Adds a note to a project or process. When you use this method, you must specify a value to the note property and either the projectId or processId property.

Pre and Post Methods

The AddNoteFormHandler has the following pre and post methods:

Method	Function
preAddNote	Empty method that is executed before the handleAddNote method. Define this method when you subclass AddNoteFormhandler and you require a new method that is executed immediately before handleAddNote.
postAddNote	Empty method that is executed after the handleAddNote method. Define this method when you subclass AddNoteFormhandler and you require a new method that is executed immediately after handleAddNote.

Example

This example creates a form that lets users add a note to an active project or process. Since adding a note requires a project or process ID, this code looks first for an active project to pass to the form handler. If one doesn't exist, the active process ID is used instead. The form created here has a large text box where a user can enter information that, when the Add a Note button is clicked, will display as a note.

```
<dspel:importbean bean="/atg/epub/servlet/AddNoteFormHandler"/>
  <dspel:form formid="addForm"  action="addNote.jsp" method="post">

     <c:choose>
        <c:when test="${param.projectId ne null}">
           <dspel:input type="hidden" bean="AddNoteFormHandler.projectId"
            value="${param.projectId}"/>
        </c:when>
        <c:otherwise>
           <dspel:input type="hidden" bean="AddNoteFormHandler.processId"
            value="${param.processId}"/>
        </c:otherwise>
     </c:choose>

     <p>Note:<dspel:textarea bean="AddNoteFormHandler.note" rows="4"
      cols="60"></dspel:textarea>

     <dspel:input type="submit" bean="AddNoteFormHandler.addNote" value="Add a
      Note"/>
  </dspel:form>
```

AssetDiffFormHandler

The `atg.epub.servlet.AssetDiffFormHandler` class lets you select two versions of an item and resolve the property value differences between them. ATG Content Administration includes `/atg/epub/servlet/AssetDiffFormHandler` for working with assets in the asset portlet and `/atg/epub/servlet/ProjectDiffFormHandler` for working with projects in the project portlet.

Configuration Properties

The `AssetDiffFormHandler` has the following configuration properties:

Property	Function
`assetURI`	Holds the asset's URI and ID, which includes the repository name, item descriptor name and asset ID. This does not provide version-specific information.
`deletedWarning`	Holds the message displayed when one version of an item involved in a comparison is deleted.
`diffChangeList`	Refers to a `PropetyChangeList` that records the differences between the two versions of the item in `PropertyChange` objects.

Property	Function
diffProperties	Holds the properties that have one value in one version and a different value in the other.
indexedVersions	Holds an array of version numbers. If this property has only one value, the working version is used as the second value. Use this property to specify the versions you want to compare.
item1	Holds the RepositoryItem representing one version involved in the comparison. This property is automatically populated, using the assetURI and indexedVersions properties to locate the Repository ID.
item2	Holds the RepositoryItem representing one version involved in the comparison. This property is automatically populated, using the assetURI and indexedVersions properties to locate the Repository ID.
selectedVersionsForMerge	When there are property value conflicts across two versions, this property identifies which version's property value is used. This property is a map in which the key holds the property with conflicting values and the value holds a 1 (representing item1) or 2 (representing item2).
versions	Holds an array of version numbers. Use this property to work with all versions at the same time.
viewAttribute	Holds the name of the portlet you use to access the item. Options include assetView and projectView.
workspaceName	Holds the current project's workspace. The best way to specify a value for this attribute is to set it to a method that accesses the project's workspace.

Navigational Property

The AssetDiffFormHandler has the following navigational property:

Property	Function
successView	Holds an integer that represents the view used to display the item.

Submit Handle Methods

The AssetDiffFormHandler has the following submit handler methods:

Method	Function
handleDiffVersions	This handler method accomplishes four tasks: 1. Locates the asset version RepositoryItems by combining the version (from the indexedVersions property) and the asset URI 2. Saves the two located RepositoryItems to the item1 and item2 properties, respectively 3. Compares the property values in the RepositoryItems 4. Records in the diffProperties property the property names with different values, and the values themselves as objects listed in the diffChangeList property. To use this method, you must specify values for the following properties: assetURI workspaceName viewAttribute succesView indexedVersions Note that workspaceName is only required when your project's working version is involved in the comparison.
handleMergeConflicts	Resolves which of the conflicting property values to use in the latest version by overwriting the current values with the user's selection. To use this method, you need to specify values for the following properties: selectedVersionsForMerge, viewAttribute, and succesView.
handleMergeCurrentVersion	Saves all property values in the user's version to the latest version and discards all values provided in the competing version. To use this method, you need to specify values for the following properties: projectId, viewAttribute, and succesView.
handleMergeLatestVersion	Saves all property values in the competing version to the latest version and discards all values provided in the user's version. To use this method, you need to specify values for the following properties: projectId, viewAttribute, and succesView.

Example

This example creates a form that displays two conflicting versions of an asset. The form uses the `projectView` view and, After the **View Conflicting Values** button is clicked, the next page appears in the view represented by the integer `12`.

```
<dspel:importbean bean="/atg/epub/servlet/ProjectDiffFormHandler"/>
   <dspel:form formid="addForm"  action="diffAssets.jsp" method="post">

      <dspel:input type="submit" bean="ProjectDiffFormHandler.diffVersions"
       value="View Conflicting Values"/>

      <dspel:input type="hidden" bean="ProjectDiffFormHandler.viewAttribute"
       value="projectView"/>
      <dspel:input type="hidden" bean="ProjectDiffFormHandler.successView"
       value="12"/>
      <dspel:input type="hidden" value="Myapplication/project/springPRCampaign"
       bean="ProjectDiffFormHandler.assetURI"/>
      <dspel:input type="hidden" value="${project.workspace}"
       bean="ProjectDiffFormHandler.workspaceName"/>
      <dspel:input type="hidden" bean="ProjectDiffFormHandler.indexedVersions[0]"
       value="4"/>
      <dspel:input type="hidden" bean="ProjectDiffFormHandler.indexedVersions[1]"
       value="null"  />
   </dspel:form>
```

BinaryFileAssetFormHandler

The `atg.epub.servlet.BinaryFileAssetFormHandler` lets you create and update binary file assets. Although you can modify the properties of binary assets in ATG, if you want to modify the file's contents, you need to work with the original file on your file system. ATG includes another class, `TextFileAssetFormHandler`, which lets you create assets from binary files: the contents of those files appear as an editable property value in the resultant asset.

This class uses a request-scoped component that temporarily stores its contents in the session-scoped asset info component defined for a given page. That way, one `BinaryFileAssetFormHandler` component can service all form pages that are able to work with binary (non-text) assets. ATG Content Administration includes one component of this class called `/atg/epub/servlet/BinaryFileAssetFormHandler`.

Instead of creating pages that explicitly use this form handler, you create an item mapping that associates this form handler component with an asset type. The forms in JSPs rely on an asset editor to read the item mapping so that, when a user requests a binary asset, for example, the `BinaryFileAssetFormHandler` is used.

By default, the `BinaryFileAssetFormHandler` is available in the following contexts:

- Editing asset properties in `assetEditPage.jsp`
- Viewing asset properties in the Asset tab of `assetPropertyView.jspf`
- Viewing asset properties in the Asset Browser of `assetBrowserAssetDetail.jsp`

Configuration Properties

The `BinarFileAssetFormHandler` has the following configuration properties:

Property	Function
`assetEditor`	Holds the asset editor object used by the form handler.
`fileName`	Holds the asset name that will represent the binary file.
`parentFolderPath`	Holds the path to the binary file on your file system.
`uploadedFile`	Holds the name of the binary file on your file system that you want to upload to ATG and work with as an asset.

CreateProcessFormHandler

The `atg.epub.servlet.CreateProcessFormHandler` class lets you create projects. ATG Content Administration includes one component of this class called `/atg/epub/servlet/CreateProcessFormHandler`.

Configuration Properties

The `CreateProcessFormHandler` has the following configuration properties:

Property	Function
`description`	Holds a description that is provided to the project you are creating.
`displayName`	Holds the name used for the project in the ATG Business Control Center and the ATG Control Center.
`process`	Holds the project object after it is created. Other components can access this object on a JSP using this property.
`versioningLayerTools`	Holds the `VersioningLayerTools` object that manages the resources used in the active workflow.
`workflowName`	Holds the name of the workflow that is used by the project you are creating.

Navigational Property

The `CreateProcessFormHandler` has the following navigational property:

Property	Function
`successProjectView`	Holds an integer that represents the page that is displayed when a project is created. The integer to page mapping is determined by a component of the `atg.epub.portlet.project.ProjectPortlet` class.

Submit Handler Method

The `CreateProcessFormHandler` has the following submit handler methods:

Method	Function
`handleCreateProcess`	Creates a project, associates a workflow with it, and saves the project to the `process` property. To use this method, you need to specify values for the `workspaceName` and `displayName` properties.

Pre and Post Methods

The `CreateProcessFormHandler` has the following pre and post methods:

Method	Function
`preCreateProcess`	Empty method that is executed before the `handleCreateProcess` method. Define this method when you subclass `CreateProcessFormhandler` and you require a new method that is executed immediately before `handleCreateProcess`.
`postCreateProcess`	Empty method that is executed after the `handleCreateProcess` method. Define this method when you subclass `CreateProcessFormhandler` and you require a new method that is executed immediately after `handleCreateProcess`.

Example

This example defines a form used to create projects. Form fields are provided for the project display name and description. Clicking the form's Create button creates a project and displays the specified success page.

```
<dspel:importbean bean="/atg/epub/servlet/CreateProcessFormHandler"/>
<dspel:form formid="createForm" name="createForm" action="${createActionURL}"
method="post">

   <dspel:input type="hidden" bean="CreateProcessFormHandler.workflowName"
    value="${workflowDef}"/>
   <dspel:input type="hidden"  bean="CreateProcessFormHandler.successProjectView"
    value="ProjectDetailView.jsp"/>
   <dspel:input type="text" bean="CreateProcessFormHandler.displayName" size="40"
    maxlength="40"/>
   <dspel:input bean="CreateProcessFormHandler.description" size="40"
    maxlength="255" type="text"/>

   <dspel:input bean="CreateProcessFormHandler.createProcess" type="submit"
    value="Create"/>
</dspel:form>
```

FireWorkflowOutcomeFormHandler

After you select an outcome for a task, the `/atg/epub/servlet/FireWorkflowOutcomeFormHandler` component of the `atg.epub.servlet.FireWorkflowOutcomeFormHandler` class fires a `WorkflowOutcome` event. The Process Manager listens for `fireWorkflowOutcome` events and, when it receives one, makes the next task in the workflow available.

Configuration Properties

The `FireWorkflowOutcomeFormHandler` has the following configuration properties:

Property	Function
`actionNote`	Holds the text that is saved as a note attached to the task.
`outcomeElementId`	Holds the ID representing the outcome the user selected.
`taskElementId`	Holds the ID representing the task for which an outcome was selected.

Submit Handler Method

The FireWorkflowOutcomeFormHandler has the following submit handler method:

Method	Function
handleFireWorkflowOutcome	Fires a WorkflowOutcome event when a task is completed. This method requires values for the following properties: projectId (or, if you are working with a process, processId), taskElementId, and outcomeElementId.

Pre and Post Methods

The FireWorkflowOutcomeFormHandler has the following pre and post methods:

Method	Function
preFireWorkflowOutcome	Empty method that is executed before the handleFireWorkflowOutcome method. Define this method when you subclass FireWorkflowOutcomeFormhandler and you require a new method that is executed immediately before handleFireWorkflowOutcome.
postFireWorkflowOutcome	Empty method that is executed after the handleFireWorkflowOutcome method. Define this method when you subclass FireWorkflowOutcomeFormhandler and you require a new method that is executed immediately after handleFireWorkflowOutcome.

Example

In this example, an approval form permits a user to add a note to a task and complete the task by clicking the Approval button. In order for the approval to be processed, the form provides the active project ID (or process ID when a project ID doesn't exist), task ID, and the outcome ID.

```
<dspel:importbean bean="/atg/epub/servlet/FireWorkflowOutcomeFormHandler"/>

<dspel:form name="actionFormName" formid="actionFormName"  action="actionNote.jsp"
method="post">
   <dspel:input type="hidden" bean="FireWorkflowOutcomeFormHandler.processId"
    value="${param.processId}"/>

      <c:if test="${param.projectId != null}">
```

```
        <dspel:input type="hidden"
         bean="FireWorkflowOutcomeFormHandler.projectId"
         value="${param.projectId}"/>
      </c:if>

   <dspel:input type="hidden" bean="FireWorkflowOutcomeFormHandler.taskElementId"
    value="${param.taskId}"/>
   <dspel:input type="hidden" value="${param.outcomeId}"
    bean="FireWorkflowOutcomeFormHandler.outcomeElementId"/>

   Enter your note here:<dspel:textarea rows="8" cols="40"
   bean="FireWorkflowOutcomeFormHandler.actionNote"/>

   <dspel:input type="submit" name="Approval"
    bean="FireWorkflowOutcomeFormHandler.fireWorkflowOutcome"/>
</dspel:form>
```

ProcessSearchFormHandler

The `atg.epub.servlet.ProcessSearchFormHandler` class lets you search for processes based on the criteria you specify. ATG Content Administration includes one component of this class called `/atg/epub/servlet/ProcessSearchFormHandler`.

`ProcessSearchFormHandler` defines properties that let you specify the workflow type and process status as search criteria. Other properties indicate whether:

- Returned items must be created by the active or any user
- A user's entry in a textbox is a partial or complete value

`ProcessSearchFormHandler` inherits additional properties you might want to use in a search form from parent classes. If no criteria are specified, all processes are returned by a search.

Note: This class specifies a property called `textSearchPropertyName`. Ignore this property and instead use `textSearchPropertyNames` property, which is inherited from a parent class.

Configuration Properties

The `ProcessSearchFormHandler` has the following configuration properties:

Property	Function
`mineOnly`	Indicates whether processes created by the current user (`true`) or all users (false) are returned by the search. The default value is `false`.

performDefaultSearch	Indicates whether a comprehensive list of processes is generated when the form displays. Such a list can be displayed in the form page and make a search unnecessary. The default value is true.
profile	Holds the Profile component. The default value is /atg/userprofiling/Profile.
startingWith	Indicates whether the user's entry is intended to be a complete criteria value (false) or the beginning portion of one (true). The default value is true.
status	Holds a process status. Only processes with this status are returned by the search. Options include: Edit: Process is not yet deployed. Completed: workflow process is complete and cannot be edited. Deployed:* Process is deployed to a target site. Edit_Running:* The Outreach campaign is deployed and running on the target site, and is open for modification. Running:* The Outreach campaign is deployed and running on the target site. * Valid only for ATG Outreach
workflowType	Holds the type of workflow associated to the process. Only processes with this type of workflow are returned by the search. The default workflow type is standard.

Submit Handler Method

The ProcessSearchFormHandler has the following submit handler method:

Method	Function
handleSearch	Returns all processes that match the specified criteria. By default, all processes are returned. If you want to limit the search by specifying criteria, you need to include form input elements for status, workflowType or other relevant ProcessSearchFormHandler properties. To search on other process property values, set the textSearchPropertyNames to process.<propertyName>. The entry a user provides to the textInput property is compared to the property you indicate here.

Example

This example creates a form that permits users to define the criteria used for locating processes. The form provides a text field where a user can enter a partial process name. Drop-down lists allows users to select the status and workflow type of the process they want to locate. There is also a checkbox that allows users to specify that they want only processes they own to be returned.

```
<dspel:importbean bean="/atg/epub/servlet/ProcessSearchFormHandler"/>
<dspel:form name="searchForm2" formid="searchForm2" action="${actionURL}"
 method="post">

   <dspel:input type="hidden" bean="ProcessSearchFormHandler.startingWith"
    value="true"/>
   Process Name: <dspel:input bean="ProcessSearchFormHandler.textInput" size="30"
   type="text"/>

   <dspel:select bean="ProcessSearchFormHandler.status">
      <dspel:option value="Edit"> Editable Processes </dspel:option>
      <dspel:option value="EditRunning"> Editable Campaigns </dspel:option>
      <dspel:option value="Running"> Campaigns Deployed to a Target
      </dspel:option>
      <dspel:option value="Deployed"> Processes Deployed to a Target
      </dspel:option>
      <dspel:option value="Completed"> Completed Processes and Campaigns
      </dspel:option>
   </dspel:select>

   Type of Workflow: <dspel:select bean="ProcessSearchFormHandler.workflowType">
      <dspel:option value=""/>
      <dspel:option value="${workflow.processName}"/>
   </dspel:select>

   Only My Processes:<dspel:input type="checkbox" value="true"
   bean="ProcessSearchFormHandler.mineOnly"/>

   <dspel:input type="submit" bean="ProcessSearchFormHandler.search" value="Find
    Processes"/>
</dspel:form>
```

ProjectFormHandler

The `atg.epub.servlet.ProjectFormHandler` class lets you modify the assets associated with a project. Use this form handler to add an asset to a project, remove an asset from a project, and revert changes made to an asset. ATG Content Administration includes one component of this class called `atg/epub/servlet/ProjectFormHandler`.

Configuration Properties

The ProjectFormHandler has the following configuration properties:

Property	Function
asset	Holds the ID for the asset you are working with.
assetAction	Holds the integer representing the type of action this form handler executes. Set this property to the value of ADD_ASSET_ACTION, DEL_ASSET_ACTION, DISCARD_ASSET_ACTION, or REVERT_ASSET_ACTION properties as follows: <dspel:setvalue bean="ProjectFormHandler.assetAction" beanvalue="ProjectFormHandler.DEL_ASSET_ACTION"/>
assets	Holds an array of IDs for the assets you are working with.
version	Holds the ID for the asset version you are working with.

Non-Configurable Properties

The ProjectFormHandler provides the following non-configurable properties:

Property	Function
ADD_ASSET_ACTION	Holds an integer representing the add asset action. This property holds a constant value that can't be changed and is not visible in the ATG Control Center.
DEL_ASSET_ACTION	Holds an integer representing the delete asset action. This property holds a constant value that can't be changed and is not visible in the ATG Control Center.
DISCARD_ASSET_ACTION	Holds an integer representing the discard asset action. This property holds a constant value that can't be changed and is not visible in the ATG Control Center.
REVERT_ASSET_ACTION	Holds an integer representing the revert asset action. This property holds a constant value that can't be changed and is not visible in the ATG Control Center.

Submit Handle Methods

The ProjectFormHandler has the following submit handler methods:

Method	Function
handleAddAssets	Adds assets to the project. This method requires values for the following properties: projectId, assetAction and either asset or assets.
handleDiscardAssets	Removes assets from the project. This method requires values for the following properties: projectId, assetAction and either asset or assets.
handlePerformAssetAction	Reads the assetAction property to determine the type of action to perform. This method is invoked by the handleAddAsset, handlerDiscardAssets, and handlerReverAsset methods. This method requires values for the following properties: projectId, assetAction and either asset or assets.
handleRevertAsset	Discards unsaved changes made to an asset and displays values held by the last saved version. This method requires values for the following properties: projectId, assetAction and asset.

Pre and Post Methods

The ProjectFormHandler has the following pre and post methods:

Method	Function
preDiscardAssets	Empty method that is executed before the handleDiscardAssets method. Define this method when you subclass ProjectFormhandler and you require a new method that is executed immediately before handleDiscardAssets.
postDiscardAssets	Empty method that is executed after the handleDiscardAssets method. Define this method when you subclass ProjectFormhandler and you require a new method that is executed immediately after handleDiscardAssets.
preRevertAsset	Empty method that is executed before the handleRevertAsset method. Define this method when you subclass ProjectFormhandler and you require a new method that is executed immediately before handleRevertAsset.
postRevertAsset	Empty method that is executed after the handleRevertAsset method. Define this method when you subclass ProjectFormhandler and you require a new method that is executed immediately after handleRevertAsset.

Example

This example describes how to create a form that lets users remove an asset from a project. The active project ID is passed to the form handler. The asset name appears next to a checkbox with the label: Discard this asset? When the checkbox is selected, the asset is saved to the form handler `asset` property. By setting `assetAction` to the value of the `DISCARD_ASSET_ACTION` property, you indicate that the specified asset is removed from the specified project. A button at the bottom of the form invokes the `handlerDiscardAssets` method.

```
<dspel:importbean bean="/atg/epub/servlet/ProjectFormHandler"/>

<dspel:form formid="discardForm" action="${actionURL}" method="post">

   <dspel:input type="hidden" bean="ProjectFormHandler.projectId"
    value="${currentProjectId}"/>

   Discard this asset? <c:out value="="${assetURI.displayName}"/>
   <dspel:input type="checkbox" bean="ProjectFormHandler.asset"
    value="${assetURI}"/>

   <dspel:input type="hidden" bean="ProjectFormHandler.assetAction"
    beanvalue="ProjectFormHandler.DISCARD_ASSET_ACTION"/>
   <dspel:input type="hidden" bean="ProjectFormHandler.discardAssets"
    value="Remove this Asset from this Project"/>
</dspel:form>
```

RepositoryAssetFormHandler

The `atg.epub.servlet.RepositoryAssetFormHandler` class lets you create and update your repository assets. This class uses a request-scoped component that temporarily stores its contents in the session-scoped asset info component defined for a given page. That way, one `RepositoryAssetFormHandler` component can service all form pages that are able to work with repository assets. ATG Content Administration includes one component of this class called `/atg/epub/servlet/RepositoryAssetFormHandler`.

Instead of creating pages that explicitly use this form handler, you create an item mapping that associates this form handler component with an asset type. The forms in JSPs rely on an asset editor to read the item mapping so that when a user requests a repository asset, for example, the `RepositoryAssetFormHandler` is used.

By default, the `RepositoryAssetFormHandler` is available in the following contexts:

- Editing asset properties in `assetEditPage.jsp`
- Viewing asset properties in the Asset tab of `assetPropertyView.jspf`
- Viewing asset properties in the Asset Browser of `assetBrowserAssetDetail.jsp`

Configuration Properties

The RepositoryAssetFormHandler has the following configuration properties:

Property	Function
actionType	Holds the type of action being initiated. Options include: update (for updating the values of asset properties) and setView (for changing the view used to display the asset).
assetInfoPath	Holds the asset info component's path and name.
assetURI	Holds the asset's URI and ID, which includes the repository name, item descriptor name and asset ID.
attributes	Holds view mapping attributes used by the form handler. This property is a map that saves an attribute name as the key and the attribute value as the value. To indicate that the form handler value property should be a value Dictionary, all asset editors define atgFormValueDict as a key and value as the value by default. If you want to customize the attributes your form handler uses, code your asset editor to update this property accordingly.
componentPath	Holds the repository's path and name of which the asset is a part.
contextOp	Holds information used to managing the state of the asset info data stack.
displayName	Holds the asset's display name.
editMode	Indicates whether the asset is in edit (true) or view-only (false) mode.
transientItem	Holds the transient asset.
useRequestLocale	Indicates whether the error messages are displayed in the user's locale (true) or the server default locale (false).
view	Holds the integer representing the view being used.

Submit Handler Methods

The RepositoryAssetFormHandler has the following submit handler methods:

Method	Function
handleChangeView	Saves the view selection indicated in the asset editor to the view property and displays the asset in that view.

handleCreateTransient	Creates an empty asset RepositoryItem, which is saved to the transientItem property. A different form handler provides values to the asset's properties. **Caution:** Subclasses of this form handler should not overwrite this method.
handleSubmitAction	Executes the action indicated in the actionType property.

SegmentAssetFormHandler

The atg.epub.servlet.SegmentAssetFormHandler lets you create and update segment assets. Each segment is made up of properties and property values called characteristics. You can add and remove characteristics from your segments using this form handler.

This class uses a request-scoped component that temporarily stores its contents in the session-scoped asset info component defined for a given page. That way, one SegmentAssetFormHandler component can service all form pages that are able to work with segment assets. ATG Content Administration includes one component of this class called /atg/epub/servlet/SegementAssetFormHandler.

Instead of creating pages that explicitly use this form handler, you create an item mapping that associates this form handler component with an asset type. The forms in JSPs rely on an asset editor to read the item mapping so that, when a user requests a segment asset, for example, the SegmentAssetFormHandler is used.

By default, the SegmentAssetFormHandler is available in the following contexts:

- Editing asset properties in assetEditPage.jsp
- Viewing asset properties in the Asset tab of assetPropertyView.jspf
- Viewing asset properties in the Asset Browser of assetBrowserAssetDetail.jsp

Configuration Properties

The SegmentAssetFormHandler has the following configuration properties:

Property	Function
assetEditor	Holds the asset editor object used by the form handler.
removeIndex	Holds the integer representing the characteristic you want to remove from the segment.
targetingTools	Holds the component that determines the list of properties you can use in your segments. The default value for this property is /atg/targeting/html/TargetingTools.

Submit Handler Methods

The SegmentAssetFormHandler has the following submit handler methods:

Method	Function
handleAddCharacteristic	Creates a new characteristic in the segment.
handleDeleteCharacteristic	Deletes an existing characteristic from the segment.

TaskActionFormHandler

The atg.epub.servlet.TaskActionFormHandler manages task ownership by letting users assign tasks to themselves or other users. A user can also return his or her tasks to the unclaimed list. ATG Content Administration includes one component of this class, /atg/epub/servlet/TaskActionFormHandler. The component is globally scoped.

Configuration Properties

The TaskActionFormHandler has the following configuration properties:

Property	Function
assignee	Holds the name of the user to whom the task is being assigned. This value must use this format: user.primaryKey:user.userDirectory.userdirectoryName where user is the Profile component (including Nucleus address), primaryKey is the user ID, userDirectory is the user directory component (including Nucleus address), and userDirectoryName is the user directory display name.
taskElementId	Holds the ID for the task.

Submit Handler Methods

The TaskActionFormHandler has the following submit handler methods:

Method	Function
handleAssignTask	Assigns a task to a user. When you use this method, you must specify a value to the projectId (or, if you are working with a process, processId), assignee, and taskElementId properties.
handleClaimTask	Assigns a task to the active user. When you use this method, you must specify a value to the projectId (or, if you are working with a process, processId) and taskElementId property.
handleReleaseTask	Returns a task that is currently owned by the active user to the unclaimed list. When you use this method, you must specify a value to the projectId (or, if you are working with a process, processId) and taskElementId property.

Pre and Post Methods

The TaskActionFormHandler has the following pre and post methods:

Method	Function
preAssignTask	Empty method that is executed before the handleAssignTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately before handleAssignTask.
postAssignTask	Empty method that is executed after the handleAssignTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately after handleAssignTask.
preClaimTask	Empty method that is executed before the handleclaimTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately before handleClaimTask.
postClaimTask	Empty method that is executed after the handleClaimTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately after handleClaimTask.
preReleaseTask	Empty method that is executed before the handleReleaseTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately before handleReleaseTask.

postReleaseTask	Empty method that is executed after the handleReleaseTask method. Define this method when you subclass TaskActionFormhandler and you require a new method that is executed immediately after handleReleaseTask.

Example

This example shows how to create a form that lets you change the owner for a task. The example begins with the creation of the form and the Assign Task button that, when clicked, assigns a task to a user. Since this form handler works for process and project tasks, the code determines which is being used and passes the appropriate ID to the form handler. The task ID is also passed to the form handler. If the current task is complete, the task owner displays. For all other circumstances, the form contains a drop-down list with one option for each user eligible to own the task.

```
<dspel:importbean bean="/atg/epub/servlet/TaskActionFormHandler"/>

<dspel:form formid="assignForm" name="assignForm" method="post"
 action="${actionURL}">
   <dspel:input type="submit" bean="TaskActionFormHandler.assignTask"
    value="Assign Task"/>
   <c:choose>
      <c:when test="${taskInfo.taskDescriptor.assignable eq false}"/>
      <c:otherwise>
         <dspel:input type="hidden" bean="TaskActionFormHandler.processId"
          value="${projectContext.process.id}"/>
         <c:if test="${isProjectView eq true}">
            <dspel:input type="hidden" bean="TaskActionFormHandler.projectId"
             value="${projectContext.project.id}"/>
         </c:if>

         <dspel:input type="hidden" bean="TaskActionFormHandler.taskElementId"
          value="${taskInfo.taskDescriptor.taskElementId}"/>

         <pws:getAssignableUsers var="assignUsers"
          taskDescriptor="${taskInfo.taskDescriptor}"/>

         <c:set var="unowned" value="${taskInfo.owner eq null}"/>
         <c:choose>
            <c:when test="${taskStatus.completed}">
               <c:out value="${taskInfo.owner.firstName}
                ${taskInfo.owner.lastName}"/>
            </c:when>
            <c:otherwise>
               <dspel:select bean="TaskActionFormHandler.assignee">

                  <c:forEach var="user" items="${assignUsers}">
                     <c:if test="${currentUserId ne user.primaryKey}">
```

```
                        <dspel:option value="${user.primaryKey}
                         :${user.userDirectory.userDirectoryName}"
                         selected="${user.primaryKey eq
                         taskInfo.owner.primaryKey}"><c:out
                         value="${user.firstName}
                         ${user.lastName}"/></dspel:option>
                    </c:if>
                  </c:forEach>

               </dspel:select>
            </c:otherwise>
         </c:choose>
      </c:otherwise>
   </c:choose>
</dspel:form>
```

TextFileAssetFormHandler

The `atg.epub.servlet.TextFileAssetFormHandler` lets you create and update text assets. Text assets have a property called `textContent` that displays the file contents and makes them available for modification.

This class uses a request-scoped component that temporarily stores its contents in the session-scoped asset info component defined for a given page. That way, one `TextFileAssetFormHandler` component can service all form pages that are able to work with text assets. ATG Content Administration includes one component of this class called `/atg/epub/servlet/TextFileAssetFormHandler`.

Instead of creating pages that explicitly use this form handler, you create an item mapping that associates this form handler component with an asset type. The forms in JSPs rely on an asset editor to read the item mapping so that, when a user requests a text asset, for example, the `TextFileAssetFormHandler` is used.

By default, the `TextFileAssetFormHandler` is available in the following contexts:

- Editing asset properties in `assetEditPage.jsp`
- Viewing asset properties in the Asset tab of `assetPropertyView.jspf`
- Viewing asset properties in the Asset Browser of `assetBrowserAssetDetail.jsp`

Configuration Properties

The `TextFileAssetFormHandler` has the following configuration properties:

Property	Function
`fileName`	Holds the asset name that represents the text file.
`parentFolderPath`	Holds the path to the text file on your file system.
`text`	Holds the contents of the text file.
`uploadedFile`	Holds the name of the text file on your file system that you want to upload to ATG and work with as an asset.

Note: If a value is provided to `uploadedFile` and `text`, the value in `uploadedFile` is used.

Appendix D: PWS 2.0 Tag Library

The PWS 2.0 tag library contains a core set of tags that let you access ATG Content Administration-related Nucleus components in JSPs. By default these tags use the pws prefix, such as `pws:getAsset`, although you can use any prefix you like. All tags in this library support JSP 2.0 technology and are enabled to use the Expression Language (EL).

The tag library source and definition files are included in your ATG installation at the following location:

```
<ATG10dir>/Publishing/taglib/pwsTaglib-2_0.tld
```

For code samples that use these tags, see:

```
<ATG10dir>/PubPortlet/PubPortlets.ear/portlets.war/html
```

The PWS Tag Library includes the following tags:

Tag Name	Description
`pws:canFireTaskOutcome`	Determines whether a user can perform a workflow task.
`pws:categorize`	Organizes objects based on categories and properties.
`pws:createVersionManagerURI`	Accesses the URI for the VersionManager used by a particular asset.
`pws:display`	Modifies a text string by removing HTML formatting, adding text, and/or shortening the string length.
`pws:getAsset`	Locates an asset using its URI.
`pws:getAssignableUsers`	Assembles a list of the users who are permitted to execute a workflow task.
`pws:getCurrentProject`	Accesses the current project ID.
`pws:getDependentProjects`	Finds projects that have dependence on each other.
`pws:getDeployedProjects`	Locates all projects deployed to a particular target site.
`pws:getDeployment`	Accesses a deployment item using a deployment ID
`pws:getDeployments`	Creates a list of deployment IDs for a given target site.

Tag Name	Description
pws:getItemSubTypes	Locates all subtypes for a particular asset type.
pws:getProcess	Accesses a process using a process ID.
pws:getProcesses	Creates a list of process IDs that have a certain status.
pws:getProject	Accesses a project using a project ID.
pws:getProjectAssets	Locates assets in a given project.
pws:getProjects	Creates a list of project IDs that have a certain status.
pws:getProjectsPendingDeployment	Creates a list of projects that have are not deployed despite the appearance of being deployment-ready.
pws:getTarget	Accesses a target site via its ID.
pws:getTargets	Creates a list of target IDs.
pws:getTasks	Creates a list of tasks based on the criteria you specify.
pws:getVersionedAssetTypes	Creates a list of asset types defined to work with the VersionManager.
pws:getWorkflowDefinitions	Accesses all workflow definition files.
pws:getWorkflowDescriptor	Finds the workflow descriptor used by a particular process.

pws:canFireTaskOutcome

This tag determines whether a user has the necessary permissions to perform a task in a given workflow. After the tag is executed, the result is stored in a result object that's named by the `var` attribute. Other tags can access the result object's properties.

Attributes

The following attributes are defined for the `pws:canFireTaskOutcome` tag:

Attribute	Description	Required?
var	Names the Boolean result object that identifies whether the user can perform the specified task.	yes
taskInfo	The `atg.workflow.TaskInfo` object that represents the task to be evaluated.	yes

Result Object Property

The following property is defined for the result object produced by the `pws:canFireTaskOutcome` tag:

Property	Description
`hasAccess`	A Boolean property that indicates whether the user has permission to perform the task.

Example

```
<pws:canFireTaskOutcome var="taskPermission" taskInfo="deleteProject"/>
<c:choose>
     <c:when test="${taskPermission=true}">
       Your next task is to delete the project.<br>
     </c:when>
     <c:otherwise>
         There are no tasks for you right now.
     </c:otherwise>
  </c:choose>
```

pws:categorize

This tag uses the `category` attributes in the repository XML definition file to sort the property descriptors for each property defined in the given repository. If you have access to the `RepositoryItem` itself, use the `repositoryItem` and `itemDescriptor` attributes. Otherwise, you need to specify the `repositoryPath` and `itemDescriptorName` attributes.

After the tag is executed, the property descriptors are stored in an array result object named by the `var` attribute. The result object has properties that other tags can access.

Attributes

The following attributes are defined for the `pws:categorize` tag:

Attribute	Description	Required?
`id`	Returns the instance of the tag. Call `getItems()` to return a collection of `atg.beans.DynamicPropertyDescriptors` for properties to display.	no

Attribute	Description	Required?
var	Names the Array result object to contain the sorted property descriptors.	no
repositoryItem	The RepositoryItem with properties that you want to sort.	no
itemDescriptor	The itemDescriptor for the RepositoryItem.	no
repositoryPath	The Nucleus path to the itemDescriptor.	no
itemDescriptorName	The name of the itemDescriptor.	no
hidden	Indicates whether properties that are designated as hidden properties are included in (true) or excluded from (false) the sorted collection. Hidden properties hold information that is not directly applicable to your user interface. The default value is false.	no
expert	Indicates whether properties that are designated as expert properties are included in (true) or excluded from (false) the sorted collection. Expert properties hold information that is useful for only some technical ACC users. The default value is false.	no
specificProperties	A comma-separated list of properties to which the result object should be restricted. Can also be used to override the categories attribute for given properties (see below).	no
excluded	A comma-separated list of properties that should be excluded from the result object. Can also be used to override the excludedCategories attribute for given properties (see below).	no
categories	A comma-separated list of categories. Only properties that have these category values are included in the result object.	no
excludedCategories	A comma-separated list of categories. Properties that have these category values are excluded from the result object.	no
hideSingleCategoryDescriptor	If only one category exists, indicates that the category descriptor should not be returned (true).	no

For an example of this tag in use, see oneColumnEdit.jsp, located in:

<ATG10dir>/PubPortlet/PubPortlets.ear/portlets.war/html/views/item/gsa/

pws:createVersionManagerURI

This tag assembles the URI used to locate a particular asset that is governed by the VersionManager. The attributes that are required depend on the asset type:

- RepositoryItem: If you have access to the item itself, set the repositoryItem attribute. Otherwise, use the componentName, itemDescriptorName, and itemId attributes.
- Versioned file: Use virtualFile if you have access to the file itself or specify the componentName and virtualFilePath attributes to locate it.
- Any object whose data type is unknown: set the object attribute.

After the tag is executed, the resultant URI is stored in a result object named by the var attribute. If var is not specified, the URI string is sent to the page's output stream.

Attributes

The following attributes are defined for the pws:createVersionManagerURI tag:

Attribute	Description
var	(optional) Names the result object to contain the VersionManager URI string.
versionManagerName	(optional) The name of the VersionManager. The default value is the PublishingVersionManager.
repositoryItem	The RepositoryItem object, required for RepositoryItem asset types if you have access to the item itself. Otherwise, set the attributes componentName, itemDescriptorName, and itemId.
componentName	The component that represents the RepositoryItem or virtual file. If you specify this attribute, you must also specify the following attributes: either virtualFilePath or both of the following: itemDescriptorName and itemId.
itemDescriptorName itemID	A file-based asset's item descriptor name and item ID. These attributes must be used with componentName .
virtualFilePath	The path to the file-based asset stored in ATG. This attribute also requires the attribute componentName.

Attribute	Description
virtualFile	The virtual file object, used for a versioned file if you have access to the file itself; otherwise, use attributes componentName and virtualFilePath.
object	A java.lang.Object that is either a RepositoryItem or a virtual file; required for any object whose data type is unknown.

pws:display

This tag takes the text you specify and modifies it accordingly. You can configure this tag to make the following modifications, which are performed in this order:

- Remove HTML tags included in the text
- Shorten the text to include all characters up to the first space or, if that space is after the character number indicated in the maxLength attribute, shorten the text to the indicated maximum length.
- Add specific text to the end of the text string.

Keep in mind that the resultant text string is longer than the value indicated by the maxLength attribute if you also use the appendedText attribute. The resultant text is saved to a String result object named by the var attribute.

Note: This tag is an instance of the EscapeHTMLTextTag class.

Attributes

The following attributes are defined for the pws:display tag:

Attributes	Description	Required?
var	Names the result object to contain the text.	yes
text	The text that you want to modify.	yes
appendedText	The text that should be appended to the resultant text.	no
maxLength	The maximum number of characters that remain in the text after HTML characters are removed. When you use this attribute, the text ends just before the first space or, if that exceeds the maxLength value, at the maxLength value itself.	no

pws:getAsset

This tag locates an asset using the URI you specify. After the tag is executed, the resultant asset is stored in a result object named by the `var` attribute. The result object has properties that other tags can access.

Attributes

The following attributes are defined for the `pws:getAsset` tag:

Attribute	Description	Required?
`var`	Names the result object to contain the asset.	yes
`uri`	The URI that identifies the asset's location.	yes
`workspaceName`	The name of the workspace associated with the working version of the asset.	no

Result Object Properties

The following properties are defined for the result object produced by the `pws:getAsset` tag:

Property	Description
`asset`	Contains the asset.
`workingVersion`	Identifies the working version of the asset, if you specified the `workspaceName` attribute.

pws:getAssignableUsers

This tag determines all users who have the permissions necessary to execute a given workflow task. After the tag is executed, the results are stored in a Set object named by the `var` attribute. The set object consists of `atg.userdirectory.User` objects.

Attributes

The following attributes are defined for the `pws:getAssignableUsers` tag:

Attribute	Description	Required?
`var`	Names the result object to contain the Set of `User` objects.	yes

taskDescriptor	The name of the task descriptor that defines the task.	yes
includeAccUsers	Indicates whether ACC users are included (true) or excluded (false). The default value is false.	no

pws:getCurrentProject

This tag uses the Dynamo request object to access the project that is active during the current session and stores that project and its related process information in a result object named by the var attribute. Other tags can access information about the project using properties on the result object.

Attribute

The following attribute is defined for the pws:getCurrentProject tag:

Attribute	Description	Required?
var	Names the result object to contain the active project.	yes

Result Object Properties

The following properties are defined for the result object produced by the pws:getCurrentProject tag:

Property	Description
process	Contains the active process object.
project	Contains the active project object.

pws:getDependentProjects

This tag uses the candidate project ID to locate all projects on which it is dependent for a given target site. You must specify one candidate project (in the candidateProject attribute) to receive a list of dependent projects or specify multiple candidate projects (in the candidateProjects attribute) to receive a list of dependent projects for them.

In order for candidate Project A to be dependent on the Project B for target site C, all of the following must be true:

- Project A has assets with property values that refer to resources used by Project B.

- Project A is not checked in and deployed to target site C.

A collection of projects are saved to the result object named by the `var` attribute.

Attributes

The following attributes are defined for the `pws:getDependentProjects` tag:

Attribute	Description	Required?
`var`	Names the result object to contain a Collection of project IDs representing projects that are dependent on the candidate project.	yes
`target`	The name of the target site associated with the candidate and dependent projects.	no
`candidateProjects`	The names of projects that depend on other projects.	no
`candidateProject`	The name of a project that depends on another project.	no

pws:getDeployedProjects

This tag obtains information about deployed projects and stores that information in a result object named by the `var` attribute. That result object has properties so other tags can access project information.

Attributes

The following attributes are defined for the `pws:getDeployedProjects` tag:

Attribute	Description	Required?
`var`	Names the result object to contain project information.	yes
`target`	The target site associated with the version of the project about which you want information. If no target is specified, deployed projects for all targets are located.	no
`index`	The number that designates where, in the context of the accessed projects, the list of returned projects should begin. The default value is `0`.	no
`count`	The number of projects to save to the result object.	no

totalCountVar	Names a secondary result object that holds the total number of deployed projects in the database located by this tag. A subset of the total can be saved to the result object.	no

Result Object Properties

The following properties are defined for the var-named result object produced by the pws:getDeployedProjects tag:

Property	Description
count	Contains the number of deployed projects saved to the result object.
projects	Contains the IDs for the deployed projects.
totalCount	Contains the total number of deployed projects located by the pws:getDeployedProjects, a subset of which can be saved to result object.

pws:getDeployment

This tag accesses an atg.deployment.server.Deployment object when you provide that item's deployment ID. The Deployment object is stored in a result object named by the var attribute.

Attributes

The following attributes are defined for the pws:getDeployment tag:

Attribute	Description	Required?
var	Names the result object to contain atg.deployment.server.Deployment object.	yes
deploymentId	The ID for the Deployment object.	yes

pws:getDeployments

This tag locates all IDs for the atg.deployment.server.Deployment objects that meet the criteria you specify and stores those IDs in a result object named by the var attribute. Other tags can access the result object and the properties defined on it.

Attributes

The following attributes are defined for the `pws:getDeployments` tag:

Attribute	Description	Required?
`var`	Names the result object to contain an array of `Deployment` IDs.	yes
`target`	The name of the target site for which you want to locate associated deployment objects. If no target is specified, all `Deployment` objects that meet the criteria are returned.	no
`projects`	The projects that are part of the `Deployment` objects you want to locate.	no
`queuedOnly`	Indicates whether to include pending `Deployment` objects that are deployed as soon as the target is available (`true`) or to exclude such `Deployment` objects (`false`). The default value is `true`.	no
`scheduledOnly`	Indicates whether to include pending `Deployment` objects that are scheduled to be executed on a future date `true`) or to exclude such deployment objects (`false`). The default value is `true`.	no
`index`	The number that designates where, in the context of the located `Deployment` objects, the list of returned `Deployment` IDs should begin. The default value is `0`.	no
`count`	The total number of `Deployment` IDs to save to the result object.	no
`totalCountVar`	Names a secondary result object that holds the total number of `Deployment` IDs located by this tag. A subset of the total can be saved to the result object.	no

Result Object Properties

The following properties are defined for the `var`-named result object produced by the `pws:getDeployments` tag:

Property	Description
`count`	Contains the number of deployed projects saved to the result object.
`deployments`	Contains the IDs for the deployed projects.
`totalCount`	Contains the total number of deployed projects located by the `pws:getDeployments`, a subset of which can be saved to result object.

pws:getItemSubTypes

This tag uses the item descriptor and path you specify to locate its descendant item descriptors. The list of `RepositoryItemDescriptor` objects returned by this tag is saved to a result object named by the `var` attribute. The super-type item descriptor is included in the result set if it represents an item type that isn't defined as hidden or expert level.

Attributes

The following attributes are defined for the `pws:getItemSubTypes` tag:

Attribute	Description	Required?
`var`	Names the result object to contain a list of `RepositoryItemDescriptor` objects.	yes
`repositoryPath`	The Nucleus path to the item descriptor	yes
`itemType`	The item descriptor for which you want to locate descendant item descriptors.	yes

pws:getProcess

This tag uses a process ID to obtain a process and stores it as a result object named by the `var` attribute.

Attributes

The following attributes are defined for the `pws:getProcess` tag:

Attribute	Description	Required?
`var`	Names the result object to contain the specified process.	yes
`processId`	The ID for the process that you want to access.	yes

pws:getProcesses

This tag accesses a list of processes that have a specific status and saves that list to a result object named by the `var` attribute so other tags can interact with its properties.

Attributes

The following attributes are defined for the `pws:getProcesses` tag:

Attributes	Description	Required?
`var`	Names the result object to contain a Collection of `process` objects.	yes
`status`	The status that is currently assigned to all processes that are returned. The default value is `Edit` (process is not deployed). Other options include: - `Edit_Running` (a campaign is deployed to a target site and eligible for modification) - `Deployed` (a process is deployed to a target site and all of its project tasks are complete) - `Running` (a campaign is deployed to a target site and all of its tasks are complete) - `Completed` (workflow process is complete)	no
`sortProperties`	The order in which the processes are organized is based on the values for properties included in this comma separated list. Each process is sorted by the chronological, numerical, or alphabetical value held by the specified properties. You can provide as many properties as you like. For example, by supplying two properties, you specify a primary and secondary sort order.	no
`index`	The number that designates where, in the context of the accessed processes, the list of returned processes should begin. The default value is `0`.	no
`count`	The number of processes to save to the result object.	no

Result Object Property

The following property is defined for the result object produced by the `pws:getProcesses` tag:

Property	Description
`processes`	Contains a list of process IDs.

pws:getProject

This tag uses a project ID to obtain a project and stores it as a result object named by the `var` attribute.

Attributes

The following attributes are defined for the `pws:getProject` tag:

Attribute	Description	Required?
`var`	Names the result object to contain the located project.	yes
`projectId`	The ID for the project that you want to access.	yes

pws:getProjectAssets

This tag locates all assets for a given project that meet the criteria you specify. After the tag is executed, the collection of assets is stored in a result object that's named by the `var` attribute. Other tags can access the result object's properties.

Attributes

The following attributes are defined for the `pws:getProjectAssets` tag:

Attribute	Description	Required?
`var`	Names the result object to contain the project's assets and each asset's size.	yes
`projectId`	The ID for the project that you want to access.	yes
`showExpert`	Indicates whether assets designated as expert-level should be included in the result set. The default value is `false`.	no
`checkedIn`	Indicates whether only checked-in assets should be included in the result set. The default value is `false`.	no

Result Object Properties

The following properties are defined for the result object produced by the `pws:getProjectAssets` tag:

Property	Description
`assets`	A collection of assets associated to the specified project.
`size`	An integer that indicates the assets size in bytes.

pws:getProjects

This tag accesses a Collection of projects that have a specific status and saves that list to a result object named by the `var` attribute so other tags can interact with its properties.

Attributes

The following attributes are defined for the `pws:getProjects` tag:

Attributes	Description	Required?
`var`	Names the result object to contain a Collection of `project` objects.	yes
`status`	The status that is currently assigned to all projects that are returned. The default value is `Active`. Other options include: `Completed`, `Suspended`, and `Error`. For a definition of each status, see the *Creating and Managing Projects* chapter of the *ATG Content Administration Guide for Business Users*.	no
`sortProperties`	The order in which the projects are organized is based on the values for properties included in this comma separated list. Each project is sorted by the chronological, numerical, or alphabetical value held by the specified properties. You can provide as many properties as you like. For example, by supplying two properties, you specify a primary and secondary sort order.	no
`index`	The number that designates where, in the context of the accessed projects, the list of returned projects should begin. The default value is `0`.	no
`count`	The number of processes to save to the result object.	no

Result Object Property

The following property is defined for the result object produced by the `pws:getProjects` tag:

Property	Description
`projects`	Contains a Collection of projects with the specified status.

`size`	Contains the number of projects in the `projects` property.

pws:getProjectsPendingDeployment

This tag locates a list of projects that are ready for deployment, but are not yet deployed. There are two reasons why projects might be in this situation:

- Projects are approved and either waiting in a queue to be deployed or are scheduled to be deployed on a future date.
- Projects are orphaned, meaning that they satisfy the workflow requirements for deployment, but encounter a workflow configuration issue that prevents them from being approved.

After this tag executes, it creates a result object named by the `var` attribute. That result object is a collection of `ToDoProject` objects, one representing each located pending project.

Attributes

The following attributes are defined for the `pws:getProjectsPendingDeployment` tag:

Attribute	Description	Required?
`var`	Names the result object to contain `ToDoProject` objects.	yes
`target`	The target site for which you want to find pending projects. If no target is specified, pending projects for all targets are located.	no
`index`	The number that designates where, in the context of the accessed projects, the list of returned projects should begin. The default value is `0`.	no
`count`	The total number of projects to save to the result object.	no
`totalCountVar`	Names a secondary result object that holds the total number of pending projects located by this tag. A subset of the total can be saved to the result object.	no

ToDoProject Object Properties

The following properties are defined for the `ToDoProject` objects produced by the `pws:getProjectsPendingDeployment` tag for each pending project:

Property	Description
orphaned	Indicates whether the project is orphaned. Orphaned projects are checked in and appear to be ready for deployment, but encounter a workflow configuration issue that prevents them from being approved. For example, if the target site is configured after the last workflow task is complete, the target will not recognize the pending project.
project	Contains an atg.epub.project.Project object.
target	Contains the name for the target site.
time	Contains the timestamp for the date on which the projects were approved or orphaned.
days	Contains the numbers of days for which the project is pending.
hours	Contains the number of hours added to the days to total the amount of time the project is pending.

pws:getTarget

This tag locates a target and stores it in a result object named by the var attribute.

Attributes

The following attributes are defined for the pws:getTarget tag:

Attribute	Description	Required?
var	Names the result object to contain information about target site(s).	yes
targetId	The ID of a specified target site.	yes

pws:getTargets

This tag accesses a list of atg.deployment.server.Target objects for your ATG instance and saves that list to a result attribute named by the var attribute.

Attribute	Description	Required?
var	Names the result object to contain a list of atg.deployment.server.Target objects.	yes

pws:getTasks

This tag locates a list of tasks that match the specified criteria and saves those tasks to a result object named by the var attribute. None of this tag's attributes, other than var, are required. However, if you indicate that you want inactive as well as active tasks (by setting active="false") or unclaimed as well as claimed tasks (unowned="true") to be returned, you must also specify a process ID. When you know the specific task ID for the task you want returned, it's best to enter that ID directly in the taskElementId attribute.

Attributes

The following attributes are defined for the pws:getTasks tag:

Attributes	Description	Required?
var	Names the result object to contain a list of tasks.	yes
userOnly	Indicates whether only tasks assigned to the current user (true) or all tasks for the current process (false) are located. The default value is false.	no
unowned	Indicates whether unclaimed (true) or all (false) tasks are located. The default value is false.	no
active	Indicates whether active (true) or all (false) tasks are located. The default value is true.	no
processId	The ID for the process that has tasks you want to locate.	yes, if unowned is set to true or active is set to false.
index	The number that designates where, in the context of the accessed tasks, the list of returned tasks should begin. The default value is 0.	no
taskElementId	The ID for a particular task that you want to locate.	no
count	The total number of tasks that are saved to the result object.	no

Result Object Properties

The following properties are defined for the result object produced by the `pws:getTasks` tag:

Property	Description
`processSize`	Contains the number of tasks in the project workflow's parent process.
`processTasks`	Contains the list of located process tasks.
`projectSize`	Contains the number of tasks in the project workflow.
`projectTasks`	Contains the list of located project workflow tasks.
`size`	Contains the number of located tasks.
`tasks`	Contains each located task.

pws:getVersionedAssetTypes

This tag accesses, from the default VersionManager, a list of asset types that are defined to work in an iterative, publishing environment. The list can be made up of asset types that are `RepositoryItems`, virtual files or a combination of the two. The asset types located by this tag are stored as a list in a result object that is named by the `var` attribute and accessible to other tags.

Attribute

The following attribute is defined for the `pws:getVersionedAssetTypes` tag:

Attribute	Description	Required?
`var`	Names the result object to contain an array of asset types, one for each repository or virtual file system.	yes

Result Object Properties

The following properties are defined for the result object produced by the `pws:getVersionedAssetTypes` tag:

Property	Description
componentPath	Contains the Nucleus path to the asset type component.
repositoryName	Contains the name of the Repository to which the asset type is a part.
types	Contains an array of asset type names.
repository	Contains the Boolean value that indicates whether the asset type is a repository that has types that are RepositoryItems (true).
fileSystem	Contains the Boolean value that indicates whether the asset type is a virtual file system that has types that are virtual files (true).

pws:getWorkflowDefinitions

This tag accesses all available workflows and saves them to a result object named by the var attribute so other tags can access them.

Attributes

The following attributes are defined for the pws:getWorkflowDefinitions tag:

Attribute	Description	Required?
var	Names the result object to contain workflow information.	yes
userOnly	Indicates whether only workflows that the current user can start (true) or all workflows (false) are located. The default value is false.	no
index	The number that designates where, in the context of the accessed workflows, the list of returned workflows should begin. The default value is 0.	no
count	The number of workflows to save to the result object.	no

Result Object Properties

The following properties are defined for the result object produced by the pws:getWorkflowDefinitions tag:

Property	Description
size	Contains the number of workflow definitions in the workflowsDefinition property.

workflowDefinitions	Contains a list of atg.workflow.WorkflowDescriptor objects.

pws:getWorkflowDescriptor

This tag accesses the workflow descriptor used by the workflow managing a given process. Descriptor information is stored in a result object named by the var attribute so other tags can access it.

Attributes

The following attributes are defined for the pws:getWorkflowDescriptor tag:

Attribute	Description	Required?
var	Names the result object to contain information about the workflow descriptor.	yes
processId	The name of the process with the workflow descriptor you are looking for.	yes

Result Object Properties

The following properties are defined for the result object produced by the pws:getWorkflowDescriptor tag:

Property	Description
workflowProcess	Contains the workflow descriptor for a process workflow.
workflowProject	Contains the workflow descriptor for the project workflow.

Index

D

E

F

R

S

T

U

V

W