

ContentProjectBuilder

Builds and manages projects that use the content library

Table of contents

1 Overview.....	2
1.1 Obtaining the content library.....	2
1.2 Using the content library.....	2
2 Design.....	4
3 Constraints.....	4
3.1 Allowed Child Dependencies.....	4
3.2 Allowed Parent Dependencies.....	5
4 Attributes.....	5
4.1 Exported Attributes.....	5
4.2 Defaults for Imported Attributes.....	5
5 Commands.....	6
6 Related Types.....	6
6.1 ContentProjectBuilderSetting.....	6
6.2 ContentProjectBuilderDefaults.....	7
6.3 ContentProjectBuilderTemplateDir.....	7

1. Overview

ContentProjectBuilder: *Builds and manages projects that use the content library*

ContentProjectBuilder is derived from [ProjectBuilder](#) providing the ability to manage a ControlTier project. If you are new to ProjectBuilder see the tutorial starting with: [Getting started using ProjectBuilder](#)

The following sections describe how to obtain, install and use the Content Module Library.

1.1. Obtaining the content library

Follow the steps below to obtain and install the content library.

1) Download the desired library jar file from Sourceforge:

[Content Module Library file release](#)

2) Run the Register command specifying the following arguments:

```
ad -p project -t ProjectBuilder -o content -c Register -- \
    -basedir /path/to/basedir -installroot /path/to/targetdir
```

3) Run the load-library command specifying the following arguments:

```
ad -p project -t ProjectBuilder -o content -c load-library -- \
    -jar /path/to/downloaded/content-version.jar
```

The destination project should now have the content library types loaded.

1.2. Using the content library

If you do not have the content types loaded, see the preceding section.

After the content library is installed, begin using the types in the library beginning with ContentProjectBuilder:

1) Run the Register command specifying the following arguments:

```
ad -p project -t ContentProjectBuilder -o name -c Register -- \
    -basedir /path/to/module/srcdir -installroot
/path/to/build/targetdir -install
```

2) Run the generate-objects command specifying the following arguments:

```
ad -p project -t ContentProjectBuilder -o name -c generate-objects -- \
    -name aName -defaults /path/to/your/defaults.properties -load
```

The `-name` argument is used as the name for each generated object. The `-load` flag specifies to have loaded the object definitions loaded on the server.

3) Run the depot-setup command to deploy the objects in AntDepo:

```
depot-setup -p project -a deploy
```

The objects are now ready for use either via the AntDepo CLI command, `ad`, or via JobCenter webapp GUI.

The objects are now ready for use either via the AntDepo CLI command, `ad`, or via JobCenter webapp GUI.

4) Run the BuildAndUpdate command that will use the newly generated set of objects:

```
ad -p project -t ContentUpdater -o aName -c BuildAndUpdate -- \
  -buildstamp buildstamp
```

Note:

Optionally, you can upload the job definition to Job Center. The generate-objects command will have also generated a `name-job.xml`. Login to JobCenter and push the "Create a new Job" button and upload the `name-job.xml` file.

The generate-objects command reads two template files and a defaults.properties file to generate two working files that can be used to load in the server. Without specific arguments, generate-objects will use templates from its templates directory as defined by the attribute `templateDir`.

Example:

```
--\
  ad -p project -t ContentProjectBuilder -o object -c generate-objects
  -name aName -defaults /path/to/defaults.xml -load
```

Yields an object model like so in the *project*:

```
+aName [ContentUpdater]
|
+ aName [ContentBuilder]
|
|   -aName [BuilderPackageInstallroot]
|   -aName [BuilderScmConnection]
|   -aName [BuilderScmModule]
|
+ aName [ContentSite]
|
|   -aName [ContentDeployment]
```

The `-defaults` option specifies the `defaults.properties` file to use in conjunction with a `object.template.xml` file and provides a set of properties used during the generation process.

Example: defaults.properties

```
#
# Common name given to generated objects
common.name=${opts.name}
#
# ContentBuilder settings
#
scmConnection=http://svn/repos/example-content
scmModule=${opts.name}.war
packageInstallroot=/apps/${opts.name}
#
# deployment locations
#
common.node=${framework.node}
ContentBuilder.node=${framework.node}
ContentDeployment.node=${framework.node}
```

2. Design

Super Type

ProjectBuilder

Extends [ProjectBuilder](#)

Role	Concrete. (Objects can be created.)
Instance Names	Unique
Notification	false
Template Directory	
Data View	Children, proximity: 1
Logger Name	ContentProjectBuilder

3. Constraints

3.1. Allowed Child Dependencies

- BuilderBuildFile1
- BuilderBuildTarget1
- BuilderScmBinding1
- BuilderScmConnection1
- BuilderScmLabel1
- BuilderScmModule1

- BuilderStageExtension1
- BuilderStageFilebase1
- ProjectBuilderDefaults1
- ProjectBuilderDocBase1
- ProjectBuilderForrestHome1
- ProjectBuilderOrganizationDescription1
- ProjectBuilderOrganizationName1
- ProjectBuilderOrganizationURL1
- ProjectBuilderProjectDescription1
- ProjectBuilderProjectName1
- ProjectBuilderProjectURL1
- ProjectBuilderTemplateDir1

1: These types have a *Singleton* constraint. Only one instance may be added as a resource.

3.2. Allowed Parent Dependencies

- Node

4. Attributes

4.1. Exported Attributes

Name	Property
basedir	deployment-basedir
targetdir	deployment-install-root

4.2. Defaults for Imported Attributes

Name	Default
buildfile	\${modules.dir}/ContentProjectBuilder/lib/build.xml
buildtarget	all
defaults	\${modules.dir}/ContentProjectBuilder/templates/defaults.properties
organizationDescription	Maker of the ControlTier software
organizationName	ControlTier
organizationURL	http://www.controltier.com

projectDescription	The content deployment types
projectName	content
projectURL	http://open.controltier.com
scmConnection	https://moduleforge.svn.sourceforge.net/svnroot/moduleforge/trunk
scmModule	content
stageextension	jar
stagefilebase	.*
templateDir	\${modules.dir}/ContentProjectBuilder/templates

5. Commands

Note:

Commandline options displayed in square brackets "[]" are optional. If an option expects arguments, then angle brackets are shown after the option "<>". Any default value is shown within the brackets.

6. Related Types

The following types are defined for use with ContentProjectBuilder.

6.1. ContentProjectBuilderSetting

6.1.1. Overview

ContentProjectBuilderSetting: *A ContentProjectBuilder setting.*

6.1.2. Design

Super Type Setting

Role	Abstract. (Objects cannot be created.)
Instance Names	Unique

6.1.3. Constraints

6.1.3.1. Allowed Parent Dependencies

- Builder

6.2. ContentProjectBuilderDefaults

6.2.1. Overview

ContentProjectBuilderDefaults: *file containing project defaults properties*

6.2.2. Design

Super Type

[ContentProjectBuilderSetting](#)

Role	Concrete. (Objects can be created.)
Instance Names	Unique

6.2.3. Attributes

6.2.3.1. Exported Attributes

Name	Property
defaults	settingValue

6.3. ContentProjectBuilderTemplateDir

6.3.1. Overview

ContentProjectBuilderTemplateDir: *file containing project template files*

6.3.2. Design

Super Type

[ContentProjectBuilderSetting](#)

Role	Concrete. (Objects can be created.)
Instance Names	Unique

6.3.3. Attributes

6.3.3.1. Exported Attributes

Name	Property
templateDir	settingValue