ContentPackage

A content package

Table of contents

1. Overview

ContentPackage: *A content package*

A ContentPackage contains a set of application content files. Typically built by ContentBuilder and consumed by ContentDeployment, ContentPackages contain incremental updates to a directory of content.

Being based on the tgz package type, a ContentPackage takes a directory of files and creates a gzip compressed tar file archive of those files. ContentPackage enhances the `tgz` type by including a package manifest file that declares the contents of the package. Additiionally, ContentPackage makes provisions to make backups of files that would be overridden when the archive is extracted to its installation root directory during deployment. These backups are stored in a new ContentPackage archive, tagged for rollback, which is uploaded and registered to the repository before the archive is extracted to its assigned installation root directory.

1.1. manifest.dtd

A valid ContentPackage is one that contains a manifest.xml in the root directory of the archive. Archives that do not contain a proper `manifest.xml` will be rejected by the ContentPackage extract command. The manifest.xml file is described by the following DTD:

```
<!ELEMENT manifest (package,builder,files*)>

<!ATTLIST package
installroot CDATA #REQUIRED
version CDATA #IMPLIED
type CDATA #IMPLIED
buildstamp CDATA #IMPLIED
>

<!ELEMENT builder (logentry*)>

<!ATTLIST builder
depot CDATA #IMPLIED
type CDATA #IMPLIED
object CDATA #IMPLIED
basedir CDATA #IMPLIED
scmConnection CDATA #IMPLIED
scmModule CDATA #IMPLIED
>

<!ATTLIST logentry
date CDATA #IMPLIED
author CDATA #IMPLIED
revision CDATA #IMPLIED
```

```
        >

        <!ELEMENT files (file*)>

        <!ATTLIST file
        pathname CDATA #REQUIRED
        ftype (file|directory) #IMPLIED
        installpath CDATA #IMPLIED
        checksum CDATA #IMPLIED
        mtime CDATA #IMPLIED
        mode CDATA #IMPLIED
        owner CDATA #IMPLIED
        group CDATA #IMPLIED
        >
```

Example: manifest.xml

```
<manifest>
  <package installroot='/tmp/about.war'
           buildstamp='123'
           type='ContentPackage'
           version='${opts.buildstamp}' />
  <builder
      depot='test1'
      type='ContentBuilder'
      object='about'
      basedir='/app02/daily_content_builds/trunk/about/j2ee-apps'
      scmConnection='http://svn/repos/trunk/about/example'
      scmModule='about.war'>
    <logentry date='2007-10-13T20:36:52.339629Z' author='alexh'
revision='833' />
  </builder>
  <files>
    <file
        pathname='jsps/AssemblyView.jsp'
        installpath='/tmp/about.war/jsps/AssemblyView.jsp'
        ftype='file'
        checksum='b21453a03c9da9fd2865d9a49ce154e2' />
    <file
        pathname='jsps/DeploymentView.jsp'
        installpath='/tmp/about.war/jsps/DeploymentView.jsp'
        ftype='file'
        checksum='b21453a03c9da9fd2865d9a49ce154e2' />
  </files>
</manifest>
```

1.2. Rollbacks

During the extraction stage of package installation ContentPackage will read the `manifest.xml` and determine if any of the files contained in the archive would overwrite an existing local file. If so, a new ContentPackage is created and marked as having rollback

files. These rollback packages can be used to restore the previous state, if the results of the new package are no longer desired.

Naming

ContentPackage rollback package objects are initialized using data from the incoming ContentPackage object. Their `name, filename` and `version` values however, are different and follow a convention. The table below describes the convention applied to rollback package objects.

Property	Naming convention
Name:	rollback-${package.base}-${package.version}.tgz
Filename:	rollback-${package.base}-${package.version}.tgz
Version:	rollback-${package.version}.tgz
Release tag:	rollback

If after a deployment occurs a rollback is desired use the ContentUpdater `runChangeDependencies` command:

```
         ad -p project -t ContentUpdater -o object -c
runChangeDependencies --\
            -buildstamp rollback-${buildstamp}
```

This will assign the ContentDeployment package dependencies to use the rollback packages for that version.

2. Design

Super Type

tgz

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

3. Constraints

3.1. Allowed Child Dependencies

- ContentPackage

3.2. Allowed Parent Dependencies

- ContentDeployment

3.3. Allowed Property Values

Property	Allowed Values	Default	Enforced
package-arch	• `noarch`	• `noarch`	
package-filetype	• `tgz`	• `tgz`	true
package-repo-url	• `${framework.pkg`	• `${framework.pkg`	false
package-vendor	•	•	false

4. 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.

4.1. extract

extracts the package

Extracts the content of the package archive file into the installroot directory.

Creates a rollback file containing any local files that would be overwritten by the content of the package.

The extract command utilizes the `create` command when making rollback archives. Additonally, it makes use of the `-upload -register` flags to load and register the new rollback packge.

Usage

```
extract [-filename <>] [-installroot <>]
```

4.1.1. Options

Option	Description
`filename`	*file to extract*
`installroot`	*destination directory*

4.2. create

creates the package archive

Reads data from context to register the package as a new object. See the manifest DTD section above.

Usage

```
create [-filename <>] [-installroot <>] [-register]
[-upload] [-version <>]
```

4.2.1. Options

Option	Description
`filename`	*file to extract*
`installroot`	*destination directory*
`register`	*optional flag specifying package file should be registered*
`upload`	*optional flag specifying package should be uploaded to the repository*
`version`	*package version*