AnySource Asset Adapter User Guide

Contents

Overview

The AnySource^{TM} Asset Adapter toolkit is a framework for automatically assembling and loading assets into Repository Manager. This toolkit provides a rich set of functions for loading common assets, as well as an extensible framework for creating customized solutions that fit your particular needs. This document will show you how to get started using the Asset Adapter from installation, to running a sample, to creating your own rules file for auto-loading assets. For information on extending the capabilities of the toolkit, see “AnySource Asset Adapter Developer Guide”.

Using the Asset Adapter to create assets is most useful when capturing *large quantities* of assets and automating asset updates based on changes to asset information and artifacts held by other third party tools. These external tools are referred to as **Asset Sources**, and may include source code control systems, document control systems, UDDI registries, etc.

The Asset Adapter provides an XML-driven framework that is used to define and automate the asset capture process. In addition, Asset Adapter capabilities can be directly accessed through programmatic (Java) and command line means if desired. Underlying these capabilities is a series of SOAP-based Web services that connect the underlying Automation Extensions client to a library.

The AnySource Asset Adapter framework uses AnySource parsers to extract and interpret information from the various files that make up the asset. Parsers are typically used to analyze and gather asset metadata for eventual inclusion into the asset being assembled, and are invoked by XML-driven tasks that define the asset capture process. The Asset Adapter then directly publishes the asset information to the Asset Library where it can be automatically made available to asset users or, optionally, it can be reviewed, corrected and augmented by ACEs and Asset Publishers as appropriate. SOA Software provides out-of-the-box parsers for open-ended text-based and XML-based processing, as well as specialized Java sources and WSDL parsers. In addition, custom AnySource parsers can be written in Java, optionally taking advantage of Java’s ability to call out to other languages of choice, including scripting languages such as Perl, Python, etc.

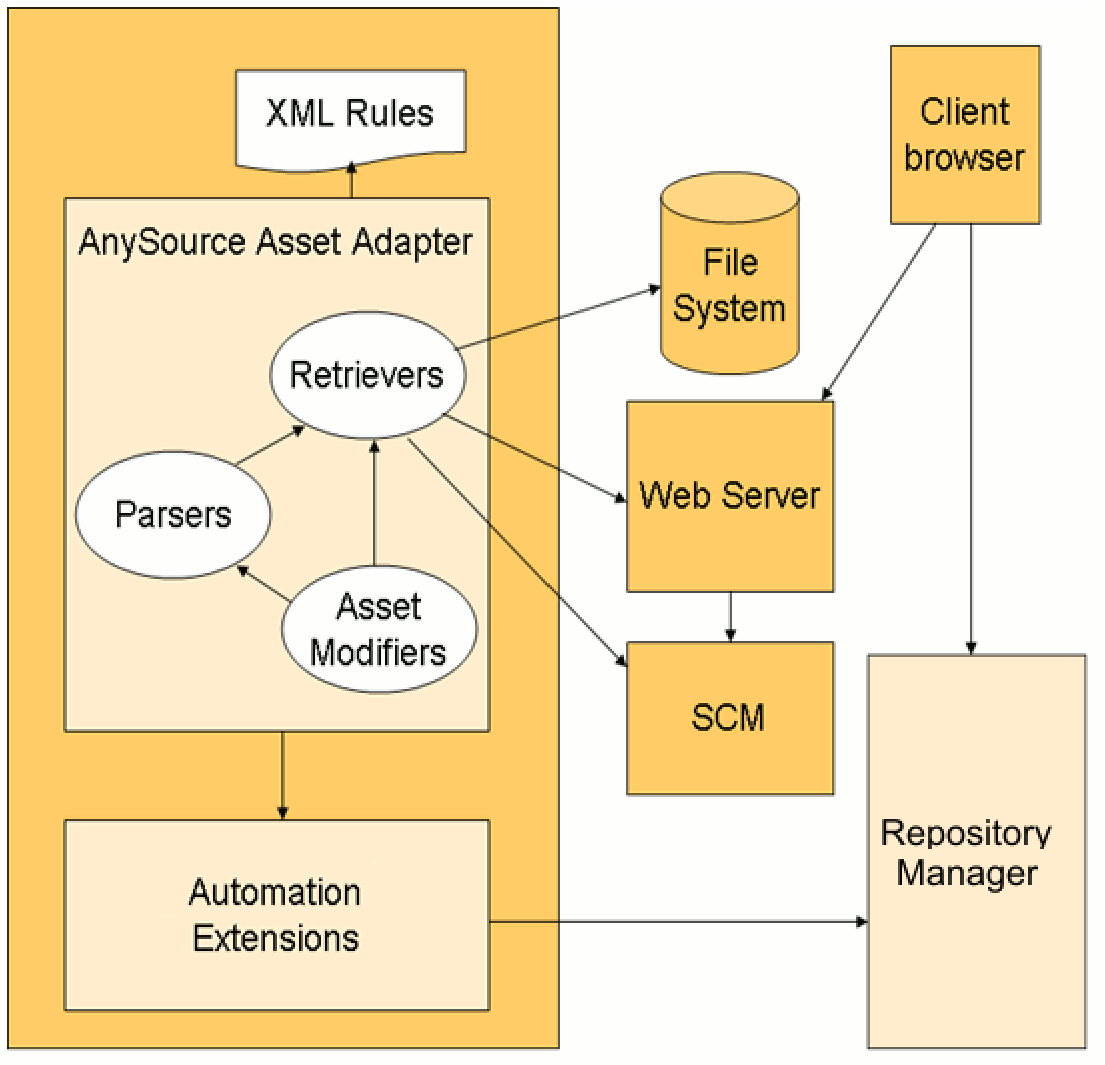
The AnySource Asset Adapter has been integrated with leading SCM systems such as Serena Dimensions and Version Manager, IBM ClearCase, CVS, WebDAV, and Metallect IQ Server, allowing asset artifacts to be loaded into and retrieved from the Library in conjunction with the associated SCM system.

Each integration is a separately licensed feature. Tasks related to each integration are documented here and are indicated as optional. Contact support for more information.

Documentation provided with the AnySource Asset Adapter includes:

- This User Guide which describes how to install and use the AnySource Asset Adapter toolkit
- A Developer Guide which describes how to write AnySource Asset Adapter extensions
- Best Practices documents that provide SCM-specific guidance for usage and deployment of the AnySource Asset Adapter

Functional Overview



The AnySource Asset Adapter runs on a Windows workstation. It reads an XML rules file and executes tasks defined in the file as various Retrievers, Parsers, and Asset Modifiers. The Retrievers read from a file system, Web server, or Source Control Manager as appropriate. Parsers take the data from Retrievers and extract information. Asset Modifiers use Parsers and Retrievers to build up metadata values for an Asset. The Asset Adapter then publishes the assets via Automation Extensions. The Repository Manager client may retrieve asset artifact files directly from Repository Manager or from a Web server. The Web server may work in concert with a particular SCM, allowing files to reside in their natural system of record instead of Repository Manager.

Sample XML Rules File

To get an initial understanding of how the AnySource Asset Adapter works, an example may be helpful. It consists of a sample XML rules file which shows the basic tasks required for assembling and publishing an asset. Following the sample is a short description of the different elements. This is intended to be an overview only. The details at this point are not important, and will be explained in more depth later.

```
<?xml version="1.0"?>
<project name="Asset Adapter 2.0" default="load">
    <property name="default-overview-file"
     value="default-overview.txt"/>
    <property environment="env"/>
    <property name="connections-file" value=
     "${env.ASSET_ADAPTER_AE_HOME}/data/connections.xml"/>
    <property name="connection-name" value="default"/>
    <property name="source-dir" value="${basedir}"/>

    <taskdef file=
     "${env.ASSET_ADAPTER_HOME}/bin/assetadapter.properties"/>

    <target name="load">
        <assetadapter action="publish" offline="false">
            <connection file="${connections-file}"
             name="${connection-name}"/>
            <!-- Define primary asset files -->
            <assetfiles id="assetfiles" assemblyid="app1">
                <!-- fileset used to locate primary asset files -->
                <fsfileset id="textfiles" dir="${source-dir}">
                    <include name="*.txt"/>
                </fsfileset>
            </assetfiles>
            <assembly id="app1" template="generic">
                <!-- Set variables to use in metadata values -->
                <variable name="asset-dir" value="@asset-uri@">
                    <xmap from="^(.*)\\.*$$" to="\1"/>
                </variable>
                <variable name="asset-file" value="@asset-uri@">
                    <xmap from="^.*\\(.*)$" to="\1"/>
                </variable>
                <variable name="asset-name-no-ext"
                 value="@asset-file@">
                    <xmap from="^(.*)\." to="\1"/>
                </variable>

                <!-- Parse description file, set description to the
                     first 240 characters -->
                <textparser id="description-parser" file="@asset-uri@">
                    <key name="description" multivalue="false"
                     expression="^(.{0,240})"/>
                </textparser>

                <assetattribute name="name"
                 value="@asset-name-no-ext@"/>
                <assetattribute name="version" value="1.0"/>
                <assetattribute name="description">
                    <parservalue parserid="description-parser"
                     parserkey="description"/>
                </assetattribute>

                <classifier name="asset-type" value="Application"/>
                <classifier name="vendor" value="LLI"/>
                <classifier name="intended-domain-general"
                 value="Education"/>
                <classifier name="autoload" value="current"/>

                <artifact category="overview" type="by-value"
                 failonerror="true" maxoccurs="1"
                 variable="overview-file-name" file="@asset-uri@"/>
            </assembly>
        </assetadapter>
    </target>
</project>
```

When executed, the default task, named “load”, is run. This invokes the “assetadapter” task which defines a “connection” to a Library and one “assembly” for creating assets of type “Application” using the “generic” asset capture template. The “assetfiles” task defines the set of primary asset files ending in a “.txt” extension in the directory `${source.dir}`, which is in the “samples/sanity” sub-directory of the Asset Adapter home (install) directory. The list of primary asset files is passed to the “assembly” task associated with the “assemblyid” attribute. For each file from the list, a set of tasks are invoked in sequence. First, the predefined variable “asset-uri” is implicitly set to the current primary asset filename. (Also, the predefined variable “primary-retriever-id” is set to the ID of the primary retriever, “assetfiles”, which is not used in this example). The first few tasks set up variables named “asset-dir”, “asset-file”, and “asset-name-no-ext”. They are derived from the “asset-uri” variable. The values for the variables are set using nested Mapper tasks, in this case “xmap”, which define regular expressions that remove portions of the filename. Next is a Parser for extracting the first 240 characters from the primary asset file. Then a set of Asset Modifiers which set the asset name, version and description, a few classifiers, and finally, an overview artifact which consists of the asset file itself.

System Requirements

- Windows 2000/XP/2000 Server
- Automation Extensions (from Support Center/Download Center)
- Disk space required: 12 MB (Asset Adapter) + 15 MB (Automation Extensions)

Install Instructions

- Install Automation Extensions
- Create an install directory, for example, `c:\Program Files\AnySource Asset Adapter`
- Unzip the contents of assetadapter.zip into this directory

For J2EE

From a command window:

- cd the install directory

- cd to the bin directory
- `copy setup.bat.tpl setup.bat`
- Edit setup.bat with a text editor.
- Change `@ASSET_ADAPTER_HOME@` to the install directory
- Save setup.bat
- Run setup.bat

Running the Sanity Test Sample

Create a test library from the Admin Console of your installation.

Create a connection in your Automation Extensions `connections.xml` file:

- `cd %LLI_AUTOMATION%\data`
- Edit `connections.xml` (if it does not exist, create it)
- Here is an sample `connections.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<connections active="default">
  <connection name="default"
      library="mylibrary"
      host="host"
      user="support"
      authenticator="LDAP_AUTHENTICATOR"
      password="" />
</connections>
```

The file needs to contain an entry for a connection to the library you wish to publish to. Fill in the library name, the host of the installation, and the userid with ACE authority to the library. Note: make the password entry blank. You may have multiple connection entries. Connection names are arbitrary, but the sanity test uses a connection named "default".

Cache the password in the connections file for the connection named "default"

- `cd ..\bin`
- `LLICmd start`
- `LLICmd login "..\data\connections.xml" default`
- Type in your password
- `LLICmd stop`

Now check the `connections.xml` file. It should have an encrypted password for the connection entry.

The sanity test sample files are in the samples\sanity directory of the installation directory

- `cd %ASSET_ADAPTER_HOME%\samples\sanity`
- run test.bat

The results should be an asset created named "TestAsset/1.0"in your library.

The XML Rules File

The AnySource Asset Adapter is built on Jakarta Ant and uses an Ant build file to run. This file is referred to as the XML rules file because it will contain the instructions you set for assembling your assets. Since this file is an Ant build file, it may contain Ant-specific tasks as well as Asset Adapter-specific tasks. For more information on Ant, see http://jakarta.apache.org/ant.

The main task for the Asset Adapter is the "assetadapter" task which must be contained within an Ant "target" element. A target can be executed by Ant by specifying the target name or names to run when invoked, or by specifying a default target for the entire file. Here is an example shell of an Asset Adapter rules file:

```
<?xml version="1.0"?>
<project name="Asset Adapter 2.0" default="load">
    <property environment="env"/>
    <taskdef
     file="${env.ASSET_ADAPTER_HOME}/bin/assetadapter.properties"/>
    <target name="load">
        <assetadapter action="publish" ...>
            ...
        </assetadapter>
    </target>
</project>
```

Note the project "name" attribute, which is required, but arbitrary, and the "default" attribute which specifies the default target to execute. The only "target" listed is one named "load" and it contains the "assetadapter" task.

The property element creates property values in this case from all currently defined environment variables from the system and uses the prefix "env" followed by a period to name each variable.

The taskdef element reads in the `assetadapter.properties` file which provides mappings from XML element names to classes which provide the implementation for the element at runtime. Note the use of the property `${env.ASSET_ADAPTER_HOME}` which substitutes the current value of the environment variable `ASSET_ADAPTER_HOME` into the pathname for the properties file.

It is also important to note that XML is case-sensitive, so element and attribute names must be of the proper case. For example, `<assetadapter>` is correct, while `<AssetAdapter>` is not.

Invoking the Asset Adapter

The Asset Adapter is run via a command line invocation which can be executed from a script or batch file. The following is an example invocation from the samples\sanity directory:

```
%ASSET_ADAPTER_ANT_HOME%\bin\startant.bat -buildfile test.xml -verbose
-debug
```

The ASSET_ADAPTER_ANT_HOME environment variable is set in the `setup.bat` file. The "-buildfile" argument points to the XML rules file to use. In this case, the default target is executed. Otherwise a target name would be specified on the command line. The "–verbose" and "-debug" arguments set the logging message level and are optional.

Output messages are printed to standard output. Output may be redirected to a file via the command line, or there are facilities in Ant for setting up output listeners to handle specialized logging functions. This is outside the scope of this document.

Encrypting Passwords

Some Asset Adapter tasks will accept an encrypted password as an attribute. The password can be encrypted using the "encryptpw" command found in the Asset Adapter "bin" directory. It will prompt for a password (which will be hidden from view) and then display the encrypted version of the password which can then be copied and pasted into the appropriate "encryptedpassword" attribute of the given Asset Adapter task.

Ant-specific Tasks

The only Ant-specific tasks that are needed by the Asset Adapter are the "taskdef" element which defines the mappings between XML element names and their corresponding class implementations, and the "property" task which defines values that can be used throughout the file.

As we saw earlier, the "taskdef" element can take a file attribute which contains a list of XML element name to class name mappings in a properties file format. Another form of the "taskdef" element takes a "name" and a "classname" attribute to do a single mapping. See the documentation for Apache Ant at http://jakarta.apache.org/ant for more information.

The "property" element in one form takes the two attributes "name" and "value". This sets a property to the value specified which can be used throughout the rest of the file by using the `${propertyname}` syntax. Note that once properties are set, they are immutable (they cannot be changed throughout the life of this invocation). Properties can also be passed in from the command line, which will override any settings in the XML rules file, or they can be loaded from a file. See the documentation for Apache Ant for more information.

Asset Adapter Tasks

assetadapter

The “assetadapter” element is the primary task responsible for implementing the Asset Adapter. It takes a few attributes and a number of child elements which control the connection, policies and assembly of assets.

Attributes of assetadapter

Attribute	Description	Required
action	Set to “publish” or “dryrun”. Default is “publish”	No.
offline	Set to “true” or “false”. Default is “false”. Work offline to create assets but not publish them. Offline means a connection will not be established with the library. This is helpful during development and debugging. It requires “templatesdir” to be set and the templates to be downloaded to this directory from a previous run or though the use of the “cachetemplates” task.	No.
assetcachedir	The directory to cache the asset files that are generated by the tool. These files are used in subsequent runs for a three-way merge algorithm to preserve updates to assets made outside of the Asset Adapter (for example manual edits using the thin client, IDE extensions or Capture Tool products, or programmatic updates through Automation Extensions.)	No. If not set, three-way merge will not be performed.
templatesdir	The directory to cache templates from the library. This is useful for offline work.	Yes if offline is set to “true”.

Nested Elements of assetadapter

connection

Sets the library connection information. Only one “connection” task is allowed and it is required. A connection must be defined in a `connections.xml` file and the password cached. The `connections.xml` file is created using Automation Extensions. See the **Install Instructions** section for information on creating this file.

Attributes of connection

Attribute	Description	Required
file	The connections.xml file which stores the connection information.	Yes.
name	The name of the connection to use.	Yes.

assetfiles

The assetfiles task creates a list of primary asset files that will drive the assembly of each asset. This task will contain a Retriever like fsfileset for creating the file list. More than one task may be listed in fsfileset, but the last element must be a Retriever and must return the final list. This allows for multiple tasks to collaborate in a special way to create the list of files, but for most cases, only one Retriever is necessary.

Attributes of assetfiles

Attribute	Description	Required
assemblyid	The assembly ID to be executed for each file URI returned by this assetfiles task.	Yes.
id	A string to identify this assetfiles task. If specified, this ID may be used by a child of an assembly by referencing the variable `@primary-retriever-id@`, which is implicitly set before the assembly is executed.	No.

Examples of assetfiles

Create a list of files under the directory from the source.dir directory whose filenames end in ".asset". For each entry in the list, the parent assembly task will assign the variable "asset-uri" to the filename of the current asset being assembled. Subsequent child elements of the assembly task can make use of this variable by using the "`@asset-uri@`" token filtering syntax. See Filtering.

```
<assetfiles variable="asset-uri">
      <fsfileset dir="${source.dir}">
            <include name="*.asset"/>
      </fileset>
</assetfiles>
```

assembly

The assembly task performs the assembly of assets. There may be multiple assembly tasks. The assembly task is defined below.

Custom Nested Elements for assetadapter

Since the AnySource Asset Adapter is an extensible framework, other nested elements may appear as children of the assetadapter task. If a task is implemented by a class that extends the IPublishPolicy or IMergePolicy interface, this class will be instantiated as the policy governing the publishing of assets and merging of asset changes, respectively. A default publishing policy and merge policy are automatically provided. See "AnySource Asset Adapter Developer Guide" for more information.

Publish or Merge Policies

uploadincompletepolicy

The uploadincomplete policy instructs the assetadapter to ignore any validation errors that may be encountered when creating the assets. Assets will be uploaded, but may not be published, regardless of missing or incomplete artifacts, classifiers, and relationships. Therefore, manual intervention in the catalog may be required to actually publish these assets.

Attributes of uploadincompletepolicy

Attribute	Description	Required
publish	If set to true, the asset will be published if it passes validation. If it is set to false or the asset fails validation, it will be stored in the catalog, but it will not be published. The default is "false".	No

Examples of uploadincompletepolicy

```
<assetadapter action="publish" offline="false"
 assetcachedir="${cache.dir}" templatesdir="${templates.dir}">
    <connection file="${connection.file}" name="default"/>
    <uploadincompletepolicy publish="false"/>
    <assetfiles assemblyid="app1">
            ...
    </assetfiles>
    <assembly id="app1" template="default">
             ...
    </assembly>
</assetadapter>
```

Examples of assetadapter

```
<assetadapter action="publish" offline="false"
 assetcachedir="${cache.dir}" templatesdir="${templates.dir}">
    <connection file="${connection.file}" name="default"/>
    <assetfiles assemblyid="app1">
```

```
            ...
    </assetfiles>
    <assembly id="app1" template="default">
             ...
    </assembly>
</assetadapter>
```

assembly

The “assembly” task defines a set of rules for assembling assets. The rules are a set of sub-task elements which identify the primary asset file for each asset and a series of tasks that collaborate to build up the asset metadata. These tasks are called iteratively in sequence. If there are, say, three primary asset files identified, all the tasks will be called in sequence for the first asset, then again for the second asset, and again for the third.

Attributes of assembly

Attribute	Description	Required
template	The asset capture template name to use for validating the assets created by this assembly. This must correspond to the name attribute of the XML template. It is not the template filename.	Yes.
publishtemplate	The filename of a publish template to use for setting publish information (like contacts, owning org group, etc.).	No.
replace	Set to “true” or “false”. Completely replace the contents of an existing asset with new contents, circumventing the merge policy. Default is “false”.	No.
failonerror	Set to “true” or “false”. Stop processing of assets if an error occurs while assembling an asset. Default is “false”.	No.

Nested Elements of assembly

childassembly

The “childassembly” task defines a set of rules for assembling finer grained assets that are based in some way on the asset being assembled by the parent assembly task. This task can be used, for example, to create an asset for every method in a Java class for each Java file processed by the parent assembly task. The childassembly and its nested elements can refer to tasks in the parent assembly by their ID.

Attributes of childassembly (in addition to the assembly task attributes)

Attribute	Description	Required
primaryretrieverid	The ID of a Retriever used to assemble assets for this childassembly. The @asset-uri@ variable is set to each file URI in turn from the Retriever.	Yes.

Nested Elements of childassembly

See assembly.

Examples of childassembly

Use a child assembly to create one asset per method name of a Java source file.

```
<parserretriever id="method-names" parserid="java-parser"
parserkey="methodnames"/>

<!-- Service Operation assets -->
<childassembly id="method-assets" primaryretrieverid="method-
names" template="Operation">
    <variable name="class-name">
        <parservalue parserid="java-parser"
parserkey="classname"/>
    </variable>
    <variable name="asset-version" value="1.0"/>
    <!-- Operation asset name: classname.operation -->
    <variable name="operation-asset-name" value="@class-
name@.@asset-uri@"/>
    <assetattribute name="name" value="@operation-asset-name@"/>
    <assetattribute name="version" value="1.0"/>
    <assetattribute name="description" value="@operation-asset-
name@"/>
    <artifact category="overview" type="by-value"
failonerror="true" tempdir="${temp-artifact-dir}"
name="overview">
        <artifactcontent>
            <parserevaluate parserid="java-parser"
parserexpression="//class/methods/method[@name='@asset-
uri@']/comment/content/html"/>
        </artifactcontent>
    </artifact>
    <!-- Make a relationshp to the "outer" asset where asset-name
and asset-version are defined outside this childassembly -->
    <assetrelationship name="containing-class" assetname="@asset-
name@" assetversion="@asset-version@"/>
</childassembly>
```

Other Nested Elements of assembly

The “assembly” task can take any number of other tasks which extend from the Task or TaskContainer classes and/or implement one of the Asset Collaboration Interfaces. These are described in Asset Collaboration Tasks and Miscellaneous Tasks below.

Asset Collaboration Tasks

In order to build up metadata for an asset, many different types of tasks may need to be employed. Some metadata may come from a text file or an XML file. Artifacts for an asset may have certain file extensions and reside in a directory structure of some known format.

The Asset Adapter defines a set of components that can work together to pull this disparate information together into an asset ready to publish. These components are of three main types: Retrievers, Parsers, and Asset Modifiers. There are also miscellaneous tasks that work with filters which implement a variable passing mechanism that the other components can make use of. The components may specify an "id" attribute that can be referenced by other components to aid in the collaboration.

Retrievers produce a list of file URIs (filenames, URLs, etc.) and retrieve the content for a given file.

Parsers parse the contents of a file or a collection of files and produce a set of key/value(s) which represent information of interest for an asset. Parsers use Retrievers to get the list of files and the file contents they are interested in.

Asset Modifiers update metadata for an asset. They can pull information from Retrievers and Parsers to update asset attributes (name, version, description), classifiers, artifacts and relationships.

An example of a miscellaneous task is the "variable" task which associates a value with a token to be replaced via the "`@variable@` notation in other tasks. This mechanism was designed to overcome the restriction of Ant properties which act like variables, but are immutable, a restriction that did not allow for changing values on each iteration through the assembly loop.

Retriever Tasks

fsfileset

The "fsfileset" task is a Retriever that works with a file system and produces a list of files and/or directories that match patterns specified in an include/exclude list. This mechanism is very similar to Ant's "fileset" task.

Attributes of fsfileset

Attribute	Description	Required
dir	The starting directory.	Yes.

casesensitive	Set to “true” or “false”. Determines if filenames should be matched with respect to case. Default is “false”.	No.
id	A unique string to identify this Retriever. The string is arbitrary but must be unique against all other Retrievers in this assembly.	No.
type	Return only files (“file”), only directories (“dir”), or all (“all”). Default is “file”.	No.

Nested Elements of fsfileset

include

The include element takes one required attribute “name” whose value is a pattern used to match a filename. The pattern may include “*” to match any string or “**/” which matches any number of subdirectories. Directory paths should be specified using forward slashes rather than backslashes. See Apache Ant documentation on fileset for more information.

exclude

The exclude element takes one required attribute “name” whose values is a pattern used to match a filename. If a file name is matched by this pattern, it is excluded from the list. If the pattern matches a directory, all subdirectories are excluded. See Apache Ant documentation on fileset for more information.

Examples

```
<fsfileset dir="/my/source/dir">
    <include name="*.java"/>
    <exclude name="Test*"/>
</fsfileset>
```

Matches files that end in .java except those that start with Test in the directory /my/source/dir.

```
<fsfileset dir="/my/source/dir">
    <include name="**/*.java"/>
</fsfileset>
```

Matches all files that end in .java in /my/source/dir and all subdirectories.

```
<fsfileset dir="/my/source/dir">
    <include name="**/*.java"/>
    <exclude name="**/test/"/>
</fsfileset>
```

Matches all files that end in .java in /my/source/dir and all subdirectories except those that are in a directory or subdirectory named “test”.

parserretriever

The “parserretriever” task turns the values from a Parser key into a Retriever that can be processed by other tasks that work with Retrievers. It is useful for processing child assembly tasks (see childassembly).

Attributes of parserretriever

Attribute	Description	Required
id	A unique string to identify this Retriever. The string is arbitrary but must be unique against all other Retrievers in this assembly.	No.
parserid	ID of the Parser to retrieve a value from.	Yes.
parserkey	The Parser lookup key for the value of interest.	Yes.

Nested Elements of parserretriever

None.

Examples

Return the values from the "methodnames" key of the Parser "java-parser" with a Retriever interface.

```
<parserretriever id="method-names" parserid="java-parser"
parserkey="methodnames"/>
```

Parser Tasks

textparser

The “textparser” task is a Parser for parsing out information from a regular text file. It takes information from a Retriever and applies expressions defined in nested elements to pull out data from the text file and associate them with key names. The expressions are Perl regular expressions for pattern matching and replacement. See www.perl.org for information on Perl.

Attributes of textparser

Attribute	Description	Required
file	The file to parse.	Either file or retrieverid is required, and file overrides retrieverid.
retrieverid	The ID of a Retriever to retrieve the file contents to be parsed.	Yes, unless file is specified.
retrieveruri	A URI of the file to be processed from the Retriever. If not specified, the first file is retrieved.	No.
id	A unique string to identify this Parser. The string is arbitrary but must be unique against all other Parsers in this assembly.	No.

Nested Elements of textparser

key

The "key" element is used to define a key name and an expression or expressions for retrieving one or more values from the text being parsed by this Parser. There may be multiple key elements nested under textparser.

Attributes of key

Attribute	Description	Required
name	The key name.	Yes.
multivalue	Set to "true" or "false". Determines if multiple matches may be performed to produce more than one value for this key. Default is "false".	No.
expression	A Perl regular expression. This expression is used to match a string from the text being parsed by this Parser. If this attribute is used, the replacement expression is assumed to be "\1" and nested expression elements may not be used. If expression is "*" the entire file contents are matched. If expression is an empty string, or not specified and no nested expression elements are specified, the entire file contents are matched.	No.

Nested Elements of key

expression

If the expression attribute is not specified, the key element accepts one or two nested expression elements. The first expression will define the Perl regular expression pattern used to match text. The second expression, if specified, determines the replacement pattern for the value. The replacement pattern may include \1, \2, etc., which indicate the parentheses matched in the first expression. A default replacement expression of "\1" is assumed if a second expression is not given.

xmap (or any other Mapper task)

Multiple Mapper elements may be used to serially transform the expression. See Mapper. The result of the last Mapper is used as the value for the key. Note to developers: any task which implements the IMapper interface may be used in place of this Mapper.

Examples of textparser

A simple property file parser which Parses text from the "property-fileset" Retriever and produces two keys: "version" and "description". The former is set to the text after the string "com.myproject.version=" and the latter is set to the text after the string "com.myproject.description=". The first key uses two expression elements, although the second one is not needed because the default is "\1". The second key uses a single expression attribute with the implied "\1" replacement expression.

```
<textparser id="property-parser" retrieverid="property-files">
    <key name="version" multivalue="false">
        <expression value="^com.myproject.version=(.*)$$"/>
        <expression value="\1"/>
    </key>
    <key name="description"
     expression="^com.myproject.description=(.*)$$"
     multivalue="false"/>
</textparser>
```

Take the contents of a file from the "description-fileset" Retriever, match the entire contents, then take the first 240 characters and assign that to the key "description".

```
<textparser id="description-parser" retrieverid="description-files">
    <key name="description" expression="*" multivalue="false">
        <xmap from="^(.{240})" to="\1"/>
    </key>
</textparser>
```

Retrieve the current file from the primary asset files retriever based on the "@asset-uri@" variable.

```
<textparser id="current-file" retrieverid="asset-files"
retrieveruri="@asset-uri@">
    <key name="whole-file" expression="*"/>
</textparser>
```

xmlparser

The "xmlparser" task is a Parser for parsing out information from an XML file. It takes information from a Retriever and applies expressions defined in nested elements to pull out data from the file and associate them with key names. The expressions are XPath expressions (see http://www.w3.org/TR/xpath).

Attributes of xmlparser

Attribute	Description	Required
file	The file to parse.	Either file or retrieverid is required, and file overrides retrieverid.
retrieverid	The ID of a Retriever to retrieve the file contents to be parsed.	Yes, unless file is specified.
retrieveruri	A URI of the file to be processed from the Retriever. If not specified, the first file is retrieved.	No.
id	A unique string to identify this Parser. The string is arbitrary but must be unique against all other Parsers in this assembly.	No.
namespaceaware	Set to "true" or "false". Determines if parsing should acknowledge XML namespaces in XPath expressions. Default is "false".	No.

Nested Elements of xmlparser

key

The "key" element is used to define a key name and an expression or expressions for retrieving one or more values from the text being parsed by this Parser. There may be multiple key elements nested under xmlparser.

Attributes of key

Attribute	Description	Required
name	The key name.	Yes.
multivalue	Set to "true" or "false". Determines if multiple matches may be performed to produce more than one value for this key. Default is "false".	No.

expression	An XPath expression. This expression is used to match a string from the text being parsed by this Parser.	No.

Nested Elements of key

expression

If the expression attribute is not specified, the key element accepts a nested expression element which defines the XPath pattern used to match the content of the XML file.

xmap (or any other Mapper task)

Multiple Mapper elements may be used to serially transform the expression. See Mapper. The result of the last Mapper is used as the value for the key. Note to developers: any task which implements the IMapper interface may be used in place of this Mapper.

Examples of xmlparser

Parse the contents of an XML file with elements which include "`<version>`" and "`<description>`" elements and store the values in the parser's key/value list.

```
<xmlparser id="property-parser" retrieverid="xml-files">
    <key name="version"
        expression="//version/text()"
        multivalue="false"/>
    <key name="description"
        expression="//description/text()"
        multivalue="false"/>
</xmlparser>
```

Parse the contents of an XML file which includes namespaces and set the "`asset-name`" key to the value of "`name`" attribute of the "`wsdl:service`" element.

```
<xmlparser id="wsdl-parser" namespaceaware="true" retrieverid="xml-
files">
    <key name="asset-name"
        expression="//*[name()='wsdl:service']/@name"
        multivalue="false"/>
</xmlparser>
```

javaxmlparser

The “javaxmlparser” task is a Parser for parsing out information from a Java source file using XPath expressions. It takes information from a Retriever and applies expressions defined in nested elements to pull out data from an XML representation of the file and associate them with key names. The expressions are XPath expressions (see http://www.w3.org/TR/xpath). Javadoc comments and javadoc tags are converted to XML representation and any HTML tags are converted to XHTML format. This XML content can be run through an XSLT stylesheet to format the content as desired. See the “xsltmapper” task.

Attributes of javaxmlparser

Attribute	Description	Required
file	The Java file to parse.	Either file or retrieverid is required, and file overrides retrieverid.
retrieverid	The ID of a Retriever to retrieve the file contents to be parsed.	Yes, unless file is specified.
retrieveruri	A URI of the file to be processed from the Retriever. If not specified, the first file is retrieved.	No.
id	A unique string to identify this Parser. The string is arbitrary but must be unique against all other Parsers in this assembly.	No.
namespaceaware	Set to “true” or “false”. Determines if parsing should acknowledge XML namespaces in XPath expressions. Default is “false”.	No.

Nested Elements of javaxmlparser

See xmlparser.

Example XML file

The following Java source file will produce an internal XML representation that the “javaxmlparser” task will parse.

SampleService.java:

```
/*
 * Copyright (c) 2006 SOA Software, Inc. All rights Reserved.
 */

package com.logiclibrary.service.sample;

import com.logiclibrary.common.*;

/**
 * Sample service to demo AnySource Asset Adapter's
 * Java parsing capability.
 *
 * @author dgross
 */

public interface SampleService
{
    /**
     * Creates an instance of a sample service.
     *
     * @return SampleService object
     * @throws LLException - if error creating service
     */
    public SampleService createSampleService() throws
LLException;

    /**
     * Method to test the service
     */
    public void test() throws RemoteException;

    /**
     * Configure this class with the supplied string.
     * @param str configuration string
     * @throws LLException - if error configuring service
     */
    public void configure(String str) throws LLException;

    /**
     * Execute this class.
     * @param arg1 first String argument
     * @param arg2 second String argument
     * @return int return value, 0=no error
     * @throws LLException - if error configuring service
     */
    public int execute(String arg1, String arg2) throws
LLException;

    /**
     * Private method
     */
    private void internal();
}
```

Resulting XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<java>
  <class qname="com.logiclibrary.service.sample.SampleService"
name="SampleService">
    <comment>
      <content>
        <html>
          <head>
            <title />
          </head>
          <body>Sample service to demo AnySource Asset Adapter's
Java parsing capability.</body>
        </html>
      </content>
      <tags>
        <tag>
          <name>author</name>
          <value>dgross</value>
        </tag>
      </tags>
    </comment>
    <methods>
```

```
      <method name="createSampleService" 
signature="com.logiclibrary.service.sample.SampleService 
createSampleService()" returns="com.logiclibrary.ser
vice.sample.SampleService">
        <comment>
          <content>
            <html>
              <head>
                <title />
              </head>
              <body>Creates an instance of a sample 
service.</body>
            </html>
          </content>
          <tags>
            <tag>
              <name>return</name>
              <value>SampleService object</value>
            </tag>
            <tag>
              <name>throws</name>
              <value>LLException - if error creating 
service</value>
            </tag>
          </tags>
        </comment>
      </method>
      <method name="test" signature="void test()" returns="void">
        <comment>
          <content>
            <html>
              <head>
                <title />
              </head>
              <body>Method to test the service</body>
            </html>
          </content>
          <tags />
        </comment>
      </method>
      <method name="configure" signature="void 
configure(java.lang.String str)" returns="void">
        <comment>
          <content>
            <html>
              <head>
                <title />
              </head>
              <body>Configure this class with the supplied 
string.</body>
            </html>
          </content>
          <tags>
            <tag>
              <name>param</name>
              <value>str configuration string</value>
            </tag>
            <tag>
              <name>throws</name>
              <value>LLException - if error configuring 
service</value>
            </tag>
          </tags>
        </comment>
      </method>
      <method name="execute" signature="int
execute(java.lang.String arg1, java.lang.String arg2)"
returns="int">
        <comment>
          <content>
            <html>
              <head>
                <title />
              </head>
              <body>Execute this class.</body>
            </html>
          </content>
          <tags>
            <tag>
              <name>param</name>
              <value>arg1 first String argument</value>
            </tag>
            <tag>
              <name>param</name>
              <value>arg2 second String argument</value>
            </tag>
            <tag>
              <name>return</name>
              <value>int return value, 0=no error</value>
            </tag>
            <tag>
              <name>throws</name>
              <value>LLException - if error configuring
service</value>
            </tag>
          </tags>
        </comment>
      </method>
      <method name="internal" signature="void internal()"
returns="void">
        <comment>
          <content>
            <html>
              <head>
                <title />
              </head>
              <body>Private method</body>
            </html>
          </content>
          <tags />
        </comment>
      </method>
    </methods>
  </class>
</java>
```

Examples of javaxmlparser

Parse the contents of the current Java file setting the classname key to the name of the Java class and the method names key to the list of method names. The all key will return the contents of the XML file produced from the Java file.

```
<javaxmlparser file="@asset-uri@" id="java-parser" validate="false">
    <key name="all" expression="*"/>
    <key name="classname" expression="//class/@name"/>
    <key name="methodnames" expression="//class/methods/method/@name"
multivalue="true"/>
</javaxmlparser>
```

ejbfunctionsparser

The “ejbfunctionsparser” task is a Parser for parsing out information from an Enterprise Java Bean source file and creating a Functions XML file representation of the methods defined on the interface. This content can be added to the asset as a standard interface-functions artifact or run through an XSLT stylesheet to format the content as desired. See the “xsltmapper” task.

Attributes of ejbfunctionsparser

Attribute	Description	Required
file	The Java file to parse.	Either file or retrieverid is required, and file overrides retrieverid.
retrieverid	The ID of a Retriever to retrieve the file contents to be parsed.	Yes, unless file is specified.
retrieveruri	A URI of the file to be processed from the Retriever. If not specified, the first file is retrieved.	No.
id	A unique string to identify this Parser. The string is arbitrary but must be unique against all other Parsers in this assembly.	No.

Nested Elements of ejbfunctionsparser

See textparser. Note: this parser will return the whole contents of the Functions XML file regardless of what key expression is passed to it.

Examples of ejbfunctionsparser

Parse an EJB file and add an interface-functions artifact with the resulting Functions XML produced.

```
<ejbfunctionsparser file="@ejb-file@" id="ejb-parser">
    <key name="all" expression="*"/>
</ejbfunctionsparser>
<artifact category="interface-functions" type="by-value"
  failonerror="true" tempdir="${temp-artifact-dir}">
    <artifactcontent parserid="ejb-parser" parserkey="all"/>
</artifact>
```

javafunctionsparser

The “javafunctionsparser” task is a Parser for parsing out method information from a Java source file and creating a Functions XML file representation of the methods defined on the class. This content can be added to the asset as a standard interface-functions artifact or run through an XSLT stylesheet to format the content as desired. See the “xsltmapper” task.

Attributes of javafunctionsparser

Attribute	Description	Required
file	The Java file to parse.	Either file or retrieverid is required, and file overrides retrieverid.
retrieverid	The ID of a Retriever to retrieve the file contents to be parsed.	Yes, unless file is specified.
retrieveruri	A URI of the file to be processed from the Retriever. If not specified, the first file is retrieved.	No.
id	A unique string to identify this Parser. The string is arbitrary but must be unique against all other Parsers in this assembly.	No.

Nested Elements of javafunctionsparser

See textparser. Note: this parser will return the whole contents of the Functions XML file regardless of what key expression is passed to it.

Examples of javafunctionsparser

Parse a Java file and add an interface-functions artifact with the resulting Functions XML produced.

```
<javafunctionsparser file="@java-file@" id="java-parser">
    <key name="all" expression="*"/>
</javafunctionsparser>
<artifact category="interface-functions" type="by-value"
  failonerror="true" tempdir="${temp-artifact-dir}">
    <artifactcontent parserid="java-parser" parserkey="all"/>
</artifact>
```

Value Tasks

The following tasks are used to retrieve values from a parser or a retriever or used to construct a value from a variable.

parservalue

The "parservalue" task is used to retrieve a value from a Parser by specifying the parser ID and key. The value retrieved will override the "value" attribute of the parent element. A given parser key may return multiple values, in which case it is up to the parent element on how it handles multiple values.

This element may contain nested tasks such as Mappers to construct/modify the value.

Attributes of parservalue

Attribute	Description	Required
parserid	ID of the Parser to retrieve a value from.	Yes.
parserkey	The Parser lookup key for the value of interest.	Yes.
defaultvalue	Return this value if no values are retrieved from the parser for the given key.	No.
failonerror	Set to "true" or "false". Fail if no value is found for the given key. Ignored if defaultvalue attribute is specified. Default is "false".	No.

Examples of parservalue

Set the asset version to the value from a parser.

```
<assetattribute name="version">
    <parservalue parserid="xml-parser" parserkey="version"/>
</assetattribute>
```

parserkeys

The "parserkeys" task is used to iterate over multiple parser keys that match a given key and/or value expression. The iteration is used by its parent task to iterate over the individual key/values matched. This is useful for creating multiple items (classifiers, artifacts, or relationships) where the parser key name encodes information about the type of item to be added.

Attributes of parserkeys

Attribute	Description	Required
keyexpression	A regular expression used to match keys of interest from the associated Parser. See Perl Regular Expression Syntax.	Yes.
parserid	ID of the Parser to retrieve the keys from.	Yes.
valueexpression	A regular expression used to match values of interest that are associated with the keys matched by the keyexpression. See Perl	No.

	Regular Expression Syntax.	
keyvar	A variable to set the current key to while iterating over the list of keys returned.	No.
valuevar	A variable to set the current value of the current key to while iterating over the list of keys and values returned.	No.

Examples of parserkeys

Using the parserkeys task to create asset relationships based on keys from a Parser of the form "relatedasset.xxx" where "xxx" is the relationship name:

```
<assetrelationship>
    <parserkeys keyexpression="^relatedasset\..*"
        parserid="textparser" keyvar="key" valueexpression=".*"
        valuevar="value"/>
    <relationshipname value="@key@">
        <xmap from="relatedasset\.(.*)" to="\1"/>
    </relationshipname>
    <assetname value="@value@"/>
    <assetversion value="1.0"/>
</assetrelationship>
```

parserevaluate

The "parserevaluate" task uses a Parser to evaluate an expression. It can be used to evaluate an expression that is based on a variable that may not be known to the Parser at the time, such as in a child assembly. See childassembly.

This element may contain nested tasks such as Mappers to construct/modify the value.

Attributes of parserevaluate

Attribute	Description	Required
parserid	ID of the Parser to retrieve a value from.	Yes.
parserexpression	The Parser expression to evaluate. It must match the expression syntax for the given Parser.	Yes.
defaultvalue	Return this value if no values are retrieved from the parser for the given expression.	No.
failonerror	Set to "true" or "false". Fail if no value is found for the given key. Ignored if the defaultvalue attribute is specified. Default is "false".	No.
maxoccurs	A positive integer. This will limit the maximum number of values returned by this task.	No.

Examples of parserevaluate

Evaluate an XPath expression from a javaxmlparser based on the current value of @asset-uri@ (which may be in a childassembly task) and create an overview artifact.

```
<artifact category="overview" type="by-value" failonerror="true"
tempdir="${temp-artifact-dir}" name="overview">
    <artifactcontent>
        <parserevaluate parserid="java-parser" maxoccurs="1"
         parserexpression="//class/methods/method[@name='@asset-
uri@']/comment/content/html"/>
    </artifactcontent>
</artifact>
```

url

The "url" task is used to simplify the creation of a URL (for by-reference purposes). It can be used wherever a parser value is required. It handles encoding and specifying url parameters. The URL constructed will be of the following form:
`protocol://host:port/path?param1name=param1value&param2name=param2value ...`

Attributes of url

Attribute	Description	Required
protocol	The protocol of the URL. For example: "http".	Yes.
host	The hostname of the server to use in the URL.	Yes.
port	The port of the URL. If the port is not specified, then the default port will be used depending on the protocol.	No.
path	The path to the resource.	Yes, if nested urlpath element is not specified.

Nested Elements of url

urlpath

The "urlpath" task represents the path portion of a URL. It can be specified in the url task, but the main purpose of using the urlpath nested task is to allow Mappers to operate on its contents. This element inherits attributes and behavior from the namevalueattribute task.

urlparameter

The "urlparameter" task is used to set a parameter in a URL. The url task will manage encoding the name and value of the parameter correctly. This element inherits attributes and behavior from the namevalueattribute task.

Attributes of urlpath (in addition to namevalueattribute attributes)

Attribute	Description	Required
name	The name of the URL parameter	Yes.

Examples of url

This example can be used, inside a artifactreference task in a by-reference artifact to create a url that performs a search in Google. It depends on a variable called query being set and will replace any "hello" occurrences in that query to "goodbye".

```
<url protocol="http" host="www.google.com" port="80">
    <urlpath value="/search"/>
    <urlparameter name="q" value="@query@">
        <xmap from="hello" to="goodbye"/>
    </urlparameter>
</url>
```

latestversion

The "latestversion" task is used anywhere a parservalue is expected, but most likely in the case of a relatedasset modifier. The latestversion task enumerates the versions of an asset and will return the most recent one alphanumerically. For example if you have an asset in Repository Manager with version 1.0, 2.0, and ABC. The text "ABC" asset will be returned.

Attributes of latestversion

Attribute	Description	Required
assetname	The name of the asset for which the latest version will be returned.	Yes.

Examples of latestversion

This example creates a related predecessor relationship to the latest version of asset "Base"

```
<assetrelationship name="predecessor"
    assetname="Base"/>
    <assetversion>
        <latestversion assetname="Base"/>
    </assetversion>
</assetrelationship>
```

namevalueattribute

The "namevalueattribute" element is not used directly, but is used by other tasks that inherit from its implementation to hold a name/value pair. The value can be from a

Retriever or a Parser, or may be specified directly with the "value" attribute. This element may contain nested tasks such as Mappers to construct/modify the value. Token filtering may be used in the value attribute using the "`@variable@`" token filtering syntax. See Filtering.

Tasks that inherit the behavior and attributes from this task are "artifactcategory", "artifactcontent", "artifactname", "artifactreference", "artifacttype", "artifactversion", "relationshipname", "assetname", "assetversion", "cc_attribute", "classifiername", "classifiervalue", "urlparameter", and "urlpath".

Attributes of namevalueattribute

Attribute	Description	Required
value	The value of the attribute.	Yes, unless nested elements are used to set the value.

Asset Modifier Tasks

assetattribute

The "assetattribute" task is used to set the asset name, asset version or asset description of an asset. It takes a name/value pair where the name is one of "name", "version" or "description". This element may contain nested tasks such as Mappers to construct/modify the value. Token filtering may be used in the value attribute using the "`@variable@`" token filtering syntax. See Filtering.

Attributes of assetattribute

Attribute	Description	Required
name	One of "name", "version" or "description".	Yes.
value	The value of the asset attribute.	Yes, unless nested elements are used to set the value.

Nested Elements of assetattribute

parservalue

The "parservalue" task may be used to set the value for an asset attribute. If parservalue returns multiple values, only the first value will be used. See parservalue above.

Examples of assetattribute

Set the name, version and description of an asset using variable substitution and Parser values.

```
<assetattribute name="name" value="@asset-name-no-ext@"/>
<assetattribute name="version">
    <parservalue parserid="xml-parser" parserkey="version"/>
</assetattribute>
<assetattribute name="description">
    <parservalue parserid="description-parser"
parserkey="description"/>
</assetattribute>
```

classifier

The “classifier” task is a Modifier for adding or replacing an asset classifier. A classifier has a name and a value. Valid classifiers are defined in the Global Definition Template of the library. A classifier value may be set based on the value of a variable, or the results of a Parser.

Attributes of classifier

Attribute	Description	Required
name	The name of the classifier.	Yes, unless nested elements are used to set the name.
value	The value of the classifier.	Yes, unless nested elements or parserid/parserkey are used to set the value.
maxoccurs	A positive integer. This will be the maximum number of classifiers added by this task.	No.
failonerror	Set to “true” or “false”. Should this task fail if there is an error processing this classifier? Default is “false”.	No.
parserid	ID of the Parser to retrieve value(s) from.	No.
parserkey	The Parser lookup key for the value(s) of interest.	No.

Nested Elements of classifier

classifiername

The optional “classifiername” element is used to set the name for this classifier. It overrides the “name” attribute of the parent classifier element. This element inherits attributes and behavior from the namevalueattribute task.

classifiervalue

The optional “classifiervalue” element is used to set the value for this classifier. It overrides the “value” attribute of the parent classifier element. This element inherits attributes and behavior from the namevalueattribute task.

parservalue

The “parservalue” task may be used to set the value for a classifier. If parservalue returns multiple values, multiple classifiers will be created, up to the value of maxoccurs. See parservalue above.

parserkeys

The “parserkeys” task may be used to iterate over a number of parser keys/values in order to create multiple classifiers. See parserkeys above.

Examples of classifier

Set the asset-type classifier:

```
<classifier name="asset-type" value="Application"/>
```

Create author classifiers with values from a parser:

```
<classifier name="author">
    <parservalue parserid="xml-parser" parserkey="author"/>
</classifier>
```

Use the parserkeys task to create classifiers based on keys from a Parser of the form “classifier.xxx” where “xxx” is the classifier name:

```
<classifier>
    <parserkeys keyexpression="^classifier\..*"
        parserid="textparser" keyvar="key"
        valueexpression=".*" valuevar="value"/>
    <classifiername value="@key@">
        <xmap from="classifier\.(.*)" to="\1"/>
    </classifiername>
    <classifiervalue value="@value@"/>
</classifier>
```

artifact

The “artifact” task is a Modifier for adding or replacing an asset artifact. The reference to the actual artifact may be a physical filename or a URI. This reference can be specified with the “file” attribute or by using a Retriever which allows multiple artifacts to be added to the asset. An artifact also has a name, a category and a type (by-value or by-reference), all of which may be set with a corresponding attribute or a nested element.

Normally, an artifact that is referenced by URI will be of type by-reference, but there is a special case of the asset overview artifact which is required to be by-value. In this case, the file must be written to a temporary file and the physical filename added as the reference to the artifact. The "tempdir" attribute is used for this purpose and should point to a directory for the task to write the temporary file to.

Attributes of artifact

Attribute	Description	Required
retrieverid	The ID of a Retriever to retrieve the artifact references and/or file contents.	No.
file	A filename of the asset artifact. Overrides the retrieverid.	No.
variable	The variable name to set to the filename/URI of the current artifact being processed. This variable can be used by child tasks via the "`@variablename@`" token filtering syntax, where variablename is the value of this variable attribute. See Filtering.	No.
maxoccurs	A positive integer. This will be the maximum number of artifacts added by this task.	No.
replace	Set to "true" or "false". If an artifact of this category already exists, it will be replaced. Default is "false".	No.
failonerror	Set to "true" or "false". Should this task fail if there is an error processing this artifact (say the file doesn't exist or the Retriever returned an empty list)? Default is "true".	No.
category	The category of this artifact. The list of artifact categories is defined in the Global Definition Template of the library.	Yes, unless artifactcategory nested element is used.
name	The artifact name. Default will be the artifact category. Allows a different name to be displayed for this particular artifact.	No.
type	Either "by-reference", "by-value", or by-"description".	Yes, unless artifacttype nested element is used.
version	The version info for this artifact. Default is the timestamp of the file. When an asset is republished, a change notification is generated based on the difference between the current and previous version. If the contents of an artifact alone changes, a notification is not generated. By specifying a different artifact version, a change notification will be generated. Normally, the default behavior is	No.

	adequate.	
tempdir	The directory to store a temporary file. Used if the artifact type is "by-value" and the reference is a URI/URL from a Retriever. The file will be retrieved and stored in this directory before being loaded into the repository. The file is not deleted.	No.

Nested Elements of artifact

artifactname

The optional "artifactname" element is used to set the name for this artifact. It overrides the "name" attribute of the parent artifact element. This element inherits attributes and behavior from the namevalueattribute task.

artifactcategory

The optional "artifactcategory" element is used to set the category for this artifact. It overrides the "category" attribute of the parent artifact element. This element inherits attributes and behavior from the namevalueattribute task.

artifactreference

The optional "artifactreference" element is used to set the reference for this artifact. By default, the artifact reference is the filename or URI/URL from a Retriever (or a temporary filename – see the "tempdir" attribute). This value can be overridden, say, to modify a URL or change a directory prefix. This element inherits attributes and behavior from the namevalueattribute task.

artifacttype

The optional "artifacttype" element is used to set the type for this artifact. It overrides the "type" attribute of the parent artifact element. The type must be one of "by-reference" or "by-value". This element inherits attributes and behavior from the namevalueattribute task.

artifactversion

The optional “artifactversion” element is used to set the version information for this artifact. It overrides the “version” attribute of the parent artifact element. This element inherits attributes and behavior from the namevalueattribute task.

artifactcontent

The optional “artifactcontent” element is used to set the contents of a by-value artifact. This element inherits attributes and behavior from the namevalueattribute task.

Examples of artifact

Set the overview artifact to the contents from a Retriever.

```
<artifact category="overview" type="by-value"
    retrieverid="overview-fileset"
    failonerror="true" maxoccurs="1"
    variable="overview-file-name">
    <artifactreference value="@overview-file-name@"/>
</artifact>
```

assetrelationship

The “assetrelationship” task is used to specify an asset that is related to the current asset. This relationship is held by the asset and does not exist on its own (meaning if the asset was removed, the relationship would be removed also).

Attributes of assetrelationship

Attribute	Description	Required
name	The name of the relationship.	Yes.
assetname	The name of the asset this relationship points to.	No. Unless assetid is not specified and assetname child element is not specified.
assetversion	The version of the asset this relationship points to.	No. Unless assetname is used.
failonerror	Set to “true” or “false”. Should this task fail if there is an error processing this assetrelationship? Default is “true”.	No.

Nested Elements of assetrelationship

relationshipname

The optional “relationshipname” element is used to set the name for this relationship. It overrides the “name” attribute of the parent assetrelationship element. This element inherits attributes and behavior from the namevalueattribute task.

assetname

The optional “assetname” element is used to set the name for this relationship. It overrides the “assetname” attribute of the parent relationship element. This element inherits attributes and behavior from the namevalueattribute task.

assetversion

The optional “assetversion” element is used to set the name for this relationship. It overrides the “assetversion” attribute of the parent relationship element. This element inherits attributes and behavior from the namevalueattribute task.

Examples of assetrelationship

```
<assetrelationship name="uses"
    assetname="otherasset" assetversion="1.0"/>
```

assetmerge

The “assetmerge” task is used to merge information from another asset into this one. The nested elements of assetmerge allow you to include or exclude certain artifacts, classifiers, and relationships. By default, all artifacts are included if no artifactinclude or artifactexclude statements are given. If any artifactinclude or artifactexclude statements are given, then the artifact must match at least one include and no excludes. The same applies for classifiers and relationships. Each of the nested elements may take use a ‘*’ wildcard to match multiple values. For example “support-*” will match support-level or support-person.

Attribute	Description	Required
assetname	The name of the asset to merge.	Yes.
assetversion	The version of the asset to merge.	Yes.

Nested Elements of assetmerge

artifactinclude and artifactexclude

The “artifactinclude” task and artifactexclude tasks can be used to limit the artifacts that are merged with the asset.

Attribute	Description	Required

category	The category name of the artifact the existing asset must match to be included or excluded.	Yes.

classifierinclude and classifierexclude

The “classifierinclude” task and classifierexclude tasks can be used to limit the classifiers that are merged with the asset.

Attribute	Description	Required
name	The name of the classifier the existing asset must match to be included or excluded.	Yes.

relatedassetinclude and relatedassetexclude

The “relatedassetinclude” task and relatedassetexclude tasks can be used to limit the related assets that are merged with the asset.

Attribute	Description	Required
Name	The relationship name of the related asset the existing asset must match to be included or excluded.	Yes.

Examples of assetmerge

```
<assetmerge assetname="@asset-name@" assetversion="@latest-version@">
    <artifactinclude category="overview "/>
    <classifierinclude name="*"/>
    <classifierexclude name="internal-*"/>
    <relatedassetexclude name="*"/>
</assetmerge>
```

Mapper Tasks

Mapper tasks are used as child elements of other tasks. They are passed a value from their parent and modify it in some way.

encodeurl

The “encodeurl” task is a Mapper that retrieves the URL form of a file from a Retriever. The Retriever is entrusted with creating a URL that can be used by the browser or IDE to retrieve the file. This is useful for by-reference artifacts in the “artifactreference” task. The “encodeurl” task does not support any child elements. The value passed to this Mapper from its parent is the file name returned from a Retriever.

Attributes of encodeurl

Attribute	Description	Required
retrieverid	The ID of a Retriever.	Yes.

Examples of encodeurl

```
<artifact category="usage-guide" type="by-reference"
    retrieverid="usage-guide-retriever"
    maxoccurs="1"
    variable="usage-guide-file-name">
    <artifactreference value="@usage-guide-file-name@">
        <encodeurl retrieverid="usage-guide-retriever"/>
    </artifactreference>
</artifact>
```

prepend

The “prepend” Mapper adds a given string to the beginning of its input string.

Attributes of prepend

Attribute	Description	Required
value	The string to prepend.	Yes.

Examples of prepend

Prepend the text “http://myserver” to the beginning of the input string:

```
<prepend value="http://myserver"/>
```

substitute

The “substitute” task transforms its input by substituting text.

Attributes of substitute

Attribute	Description	Required
from	The string to match.	Yes.
to	The value to be assigned.	Yes.
replaceall	Set to “true” or “false”. Should all occurrences be replaced? Default is “true”.	No.

Examples of substitute

A transformation from backslash to forward slash would look like this:

```
<substitute from="\" to="/"/>
```

substring

The “substring” task returns a portion of its input.

Attributes of substring

Attribute	Description	Required
startindex	The integer index of the first character to keep. The index value must be an integer from 0 to the length of the input string minus 1. Default is “0”.	No.
length	The number of characters to keep including the starting character. Must be greater than zero. If the number of characters from the start index to the end of the input string are less than the value for length, the rest of the input string is returned.	Yes.

Examples of substring

Return the first 200 characters of the input string:

```
<substring startindex="0" length="200"/>
```

xmap

The “xmap” task is a Mapper which takes a “from” regular expression and a “to” replacement expression to transform its input. Token filtering may be used in the “from” or “to” attribute using the “`@variable@`” token filtering syntax. See Filtering.

Attributes of xmap

Attribute	Description	Required
from	The regular expression to match the input. See Perl Regular Expression Syntax below.	Yes.
to	The replacement expression. This may include replacement tokens of the form \n where n is a positive number and represents an expression matched in the from attribute. See syntax description of () below.	Yes.

Perl Regular Expression Syntax

The following is a brief overview of Perl regular expressions. For more information, consult the Perl documentation.

^	matches beginning of line

`$$`	matches end of line (note a double dollar sign must be used since $ is a special character in Ant)
`.`	matches an arbitrary character
`*`	matches zero or more of the previous character or expression
`+`	matches one or more of the previous character or expression
`()`	groups portions of an expression and are associated with `\1`, `\2`, etc. as numbered from left to right and outermost to innermost grouping.
`{n}`	matches the previous character or expression n times.
`[abc]`	matches one of the characters “a”, “b” or “c”.
`\s`	matches a space character
`\S`	matches a non-space character
`\d`	matches a digit
`\D`	matches a non-digit
`\.`	matches a period
`\\`	matches a single backslash

Examples of xmap

Transform a filename that has and optional “Chess” or “chess” in its name and ends in “.jpr” to a string which will start with “Sample Chess” and end with the rest of the original name:

```
<xmap from="^.*/([Cc]hess)?(.+)\.jpr$$" to="Sample Chess \2"/>
```

Transform an asset URI to a directory pathname by removing everything after the last backslash:

```
<variable name="asset-dir" value="@asset-uri@">
    <xmap from="^(.*)\\.*$$" to="\1"/>
</variable>
```

xsltmapper

The “xsltmapper” task is a Mapper which takes performs an XSLT translation of its XML input using an XSLT stylesheet.

Attributes of xsltmapper

Attribute	Description	Required
style	The filename of an XSLT stylesheet.	Yes.

Examples of xsltmapper

Run the results of a javaxmlparser through an XSLT stylesheet to produce an overview artifact.

```
<artifact category="overview" type="by-value" failonerror="true"
tempdir="${temp-artifact-dir}" name="overview">
```

```
    <artifactcontent parserid="java-parser" parserkey="all">
        <xsltmapper style="overview.xsl"/>
    </artifactcontent>
</artifact>
```

Miscellaneous Tasks

variable

The “variable” task implements a simple name/value pair association. The name and value can be set directly or can be a result of a filtering transformation and/or one or more mapping transformations or a parservalue. Token filtering may be used in the value attribute using the “`@variable@`” token filtering syntax. See Filtering.

Attributes of variable

Attribute	Description	Required
name	The name to be given to the variable.	Yes.
value	The value to be assigned.	Yes, unless child mapping transformation is specified.

Examples of variable

A simple filtering transformation would look like this:

```
<variable name="foo" value="@token@"/>
```

A more complex mapping transformation requires child elements that implement the IMapper interface:

```
<variable name="foo" value="@token@">
    <map from="^(.*)/" to="\1"/>
</variable>
```

Setting a variable to a value returned from a parser:

```
<variable name="asset-name" value="@token@">
    <parservalue parserid="my-parser" parserkey="name"/>
</variable>
```

assetsource

The “assetsource” task can be used to cache capture templates locally. This would normally be used for debugging in conjunction with the “offline” attribute of the “assetadapter” task.

Nested Elements of assetsource

connection

See connection above.

cachetemplates

The optional “cachetemplates” task is used as a child of the “assetsource” task. It is used to copy capture templates (XML files) to a local directory for use with the “offline” attribute of “assetadapter”.

Attributes of parservalue

Attribute	Description	Required
dir	Directory to store the templates in.	Yes.

Example of artifact

```
<target name="offline-setup">
    <mkdir dir="${templates-dir}"/>
    <assetsource>
        <connection file="${connections-file}"
            name="${connection-name}"/>
        <cachetemplates dir="${templates-dir}"/>
    </assetsource>
</target>
```

assetdiff

The “assetdiff” task can be used to determine if the current asset is different from another. The method of comparing this asset with the one specified is quite simple. For artifacts, the artifact in this asset must exist in the specified asset with the same category, name, and version. For classifiers, each must exist in the specified asset with the same name and value. And for related assets, each must exist in the specified asset with the same relationship name, asset name, and asset version. If there is an artifact, classifier, or related asset that is processed in this asset that doesn’t exist in the asset specified, then the resultvariable is set to “true”. Otherwise it is not set. The elements to compare can be limited using the nested elements. By default all artifacts, classifiers, and related assets are compared in the diff.

Attributes of assetdiff

Attribute	Description	Required
assetname	The name of the asset to compare this one to.	Yes.

assetversion	The version of the asset to compare this one to.	Yes.
resultvariable	The variable to set to "true" if the assets differ. Otherwise it is not set	Yes.

Nested Elements of assetdiff

artifactinclude and artifactexclude

The "artifactinclude" task and artifactexclude tasks can be used to limit the artifacts that are diff'ed with the specified asset.

Attribute	Description	Required
category	The category name of the artifact the existing asset must match to be included or excluded in the diff comparison.	Yes.

classifierinclude and classifierexclude

The "classifierinclude" task and classifierexclude tasks can be used to limit the classifiers that are diff'ed with the specified asset.

Attribute	Description	Required
name	The name of the classifier the existing asset must match to be included or excluded in the diff comparison.	Yes.

relatedassetinclude and relatedassetexclude

The "relatedassetinclude" task and relatedassetexclude tasks can be used to limit the related assets that are diff'ed with the specified asset.

Attribute	Description	Required
Name	The relationship name of the related asset the existing asset must match to be included or excluded in the diff comparison.	Yes.

Examples of assetdiff

An example that compares this asset with asset A/1.1. If asset A/1.1's artifacts differ, then the assets-differ variable will be set to "true".

```
<assetdiff assetname="@asset-name@"
           assetversion="@latest-version@"
```

```
          resultvariable="assets-differ">
    <artifactinclude category=”*”/>
</assetdiff>
```

assetskip

The “assetskip” task can be used to skip processing of the current asset. Most likely this would be the result of some conditional.

Examples of assetskip

This example skips processing of the current asset if the variable skip is set to ”true”.

```
<if>
    <istrue value="@skip@"/>
    <then>
        <assetskip/>
    </then>
</if>
```

if

The “if” task is very powerful and allows the Asset Adapter to perform actions depending on the content of certain variables. The “if” task has 3 sections: a condition, a “then” clause, and an “else” clause. If the condition evaluates to true, the “then” clause is executed. Otherwise the “else” clause is executed. The condition clause can only have one element, and must appear as the first subelement in the if task, though it can contain nested elements. The “then” and “else” clauses can have multiple clauses.

Nested Elements of if

Conditional

The condition portion of the if task can include a variety of test conditions. There can only be

and

The “and” task groups other conditions, which are logically and’ed together to arrive at the result.

e.g.
```
<and>
        <equals arg1=”@test1@” arg2=”testval”/>
        <equals arg1=”@test2@” arg2=”testval”/>
    </and>
```

equals

The equal task returns true if both its arguments (arg1, and arg2) are equal.

e.g. `<equals arg1=”@test1@” arg2=”testval”/>`

isfalse

This task return true if its argument is “false”.

e.g. `<isfalse value=”@testval@”/>`

isset

The task returns true if its argument, which is a variable name is set.

e.g. `<isset variable=”variable-name”/>`

istrue

This task return true if its argument is “true”.

e.g. `<istrue value=”@testval@”/>`

not

The not task inverts the meaning of the nested task. It can only contain one direct subtask.

```
e.g. <not>
         <equals arg1=”1” arg2=”2”/>
     </not>
```

or

The “or” task groups other conditions, which are logically or’ed together to arrive at the result.

```
e.g. <or>
         <equals arg1=”@test@” arg2=”testval1”/>
         <equals arg1=”@test@” arg2=”testval2”/>
     </or>
```

Attribute	Description	Required
category	The category name of the artifact the existing asset must match to be included or excluded in the diff comparison.	Yes.

then

The “else” clause doesn’t have a fixed set of tasks that may be specified within it. It should accept any other valid asset adapter task, as long as it makes sense in the context. For example “assembly” would not be a nested task.

else

The “else” clause doesn’t have a fixed set of tasks that may be specified within it. It should accept any other valid asset adapter task, as long as it makes sense in the context. For example “assembly” would not be a nested task.

Examples of if

If an “old-version” variable is set, set a variable called asset-version and append “.0”, otherwise set the asset-version variable to “1.0”.

```
<if>
    <isset variable="old-version"/>
    <then>
        <variable name="asset-version" value="@old-version@.0"/>
    </then>
    <else>
        <variable name="asset-version" value="1.0/>
    </else>
</if>
```

script

The “script” task allows the asset adapter to perform some work that does not fit into the confines of the other Asset Adapter tasks. The only supported scripting language currently is JavaScript. This is not meant to be a complete JavaScript reference, see more definitive sources for that. Instead it is meant to be a guide on how to integrate JavaScript with the Asset Adapter tasks

When the Asset Adapter encounters a script task, there must be some way for JavaScript to be able to process the content of Repository Manager variables. In the future this may be extended to other Asset Adapter concepts, such as Mappers, and Parsers, but for now it is limited to variables.

To retrieve an Asset Adapter variable from the script task, where app1 is the assembly id:

```
      filters = app1.getLocalFilters();
      testvalue = filters.replaceTokens("@testvariablename@");
```

To set an Asset Adapter variable, where app1 is the assemblyid:

```
      prop = project.createTask("variable");
      prop.setFilters(app1.getLocalFilters());
      prop.setName("newvariable");
      prop.setValue(“variablevalue”);
      prop.perform();
```

Attributes of script

Attribute	Description	Required
language	The scripting language to use. Currently only “javascript” is supported.	Yes.

Examples of script

The following example assumes the “version” variable is set to an integer (“5”) for example and increments it.

```
<script language="javascript"><![CDATA[
      // Get the version variable
      filters = app1.getLocalFilters();
```

```
        ver = filters.replaceTokens("@version@");

        // Parse and increment the variable
        jsversion = parseInt(ver);
        jsversion = jsversion + 1;

        // Set the version variable
        task = project.createTask("variable");
        task.setFilters(app1.getLocalFilters());
        task.setName("version");
        task.setValue(jsversion);
        task.perform();
]]></script>
```

retrieverwriter

The “retrieverwriter” task will create a file from the contents of a retriever.

Attributes of retrieverwriter

Attribute	Description	Required
file	The output filename.	Yes.
retrieverid	The ID of a Retriever to retrieve the file contents.	Yes.
retrieveruri	A URI of the file to be retrieved from the Retriever. If not specified, the first file is retrieved.	No.

Nested Elements of retrieverwriter

None.

Examples of retrieverwriter

Save the contents of a file retrieved from the primary asset retriever and saving it in a temporary file.

```
<retrieverwriter file="${temp-file}" retrieverid="asset-files"
retrieveruri="@asset-uri@"/>
```

CVS Tasks (optional)

cvsfileset

The “cvsfileset” task is a Retriever that works with a CVS repository and produces a list of files and/or directories that match patterns specified in an include/exclude list. It requires use of a Web interface to CVS such as CVS Web in order to allow retrieving of files via URL. It returns pathnames in URL form relative to the provided "weburl" attribute. This task makes use of Ant's "cvs" task.

Attributes of cvsfileset

Attribute	Description	Required
weburl	The URL to the CVS web server used to retrieve the contents of a given file from the server.	Yes.
cvsroot	The CVS root specification. E.g., :pserver:*user@host*:*rootdir*.	Yes.
dest	the directory where the checked out files should be placed.	Yes.
package	CVS package/model name.	Yes.
casesensitive	Set to "true" or "false". Determines if filenames should be matched with respect to case. Default is "false".	No.
passwordfile	Path to CVS password file. E.g., .cvspass.	No.
command	CVS command to execute. Default is "checkout".	
id	A unique string to identify this Retriever. The string is arbitrary but must be unique against all other Retrievers in this assembly.	No.
type	Return only files ("file"), only directories ("dir"), or all ("all"). Default is "file".	No.

Nested Elements of cvsfileset

See fsfileset.

Examples

```
<cvsfileset cvsroot=":pserver:bsmith@cvshost:/usr/local/cvsroot"
package="componentx" dest="/cvs/client/dir"
weburl="http://cvshost/cvs/bin/cvsweb.cgi/~checkout~"
  id="xmlfiles">
    <include name="**/*.xml"/>
</cvsfileset >
```

ClearCase Tasks (optional)

cc_attributevalues

The "cc_attributevalues" is a retriever that enumerates ClearCase attribute values on elements or versions. The main purpose of this task is to enumerate a list of values for an assetfiles task to process. For example, if ClearCase elements contain asset_name attributes, then the cc_attributevalues task (nested in an assetfiles task) can be used to enumerate these values and process an asset for each one. Since this task is a retriever, but does not return file references, the results are undefined if you try to use this retriever where files are expected to be returned. For that instance, see the cc_files ClearCase task.

Attributes of cc_attributevalues

Attribute	Description	Required
attribute	The attribute value name in ClearCase to enumerate.	Yes.
path	The path to the ClearCase view that contains the elements/versions to search.	Yes
onelement	If set to true, the ClearCase attributes to search will be on the element. Otherwise, if onelement is not set or set to false the attributes to search will be attached to the ClearCase version.	No
delimiter	A delimiter to use that separates distinct values in the ClearCase attribute. For example, if an ClearCase attribute contains the value “asset1, asset2” and delimiter is set to “,”then this task will return 2 results: “asset1”, “asset2”. If delimiter is not set, then only one result will be returned: “asset1, asset2”	No

Examples of cc_attributevalues

```
<assetfiles id="assettypes" assemblyid="app1">
    <cc_attributevalues attribute="asset_name"
                        path="m:\viewdirectory"
                        onelement="true"
                        delimiter=","/>
</assetfiles>
```

The previous example searches the ClearCase view directory m:\viewdirectory for any elements which have an attribute set to asset_name. The assetfiles task will then execute the assembly named app1 once for every distinct value found in the asset_name attribute. It will interpret comma separated values in the attribute as distinct values.

cc_files

The “cc_files” is a retriever that finds all ClearCase files within a view that match a set of configured attributes.

Attributes of cc_files

Attribute	Description	Required
id	The retriever id. This can be referred to in artifact tasks to create artifacts that reference these ClearCase files.	Yes.

path	The path to the ClearCase view that contains the elements/versions to search attributes on the	Yes
onelement	If set to true, the ClearCase attributes to search will be on the element. Otherwise, if onelement is not set or set to false the attributes to search will be on the ClearCase version.	No

Examples of cc_files

```
<cc_files id="ccfiles" path="m:\viewdirectory" onelement="true">
    <cc_attribute name="asset_name"
                  value="@asset-name@"
                  delimiter=","/>
    <cc_attribute name="artifact_type"/>
</cc_files>
```

The previous example searches for elements within the ClearCase view directory m:\viewdirectory for any elements which have an asset_name set to the value of the @asset-name@ variable and have an artifact_type attribute set. See the nested task

cc_files

The “cc_files” is a retriever that finds all ClearCase files within a view that match a set of configured attributes.

Attributes of cc_files

Attribute	Description	Required
id	The retriever id. This can be referred to in artifact tasks to create artifacts that reference these ClearCase files.	Yes.
path	The path to the ClearCase view that contains the elements/versions to search attributes on the	Yes
onelement	If set to true, the ClearCase attributes to search will be on the element. Otherwise, if onelement is not set or set to false the attributes to search will be on the ClearCase version.	No

Nested Elements of cc_files

cc_attribute

The “cc_attribute” will restrict the types of files that the cc_files task retrieves. Multiple cc_attribute elements can be specified and the cc_files task will only retrieve those elements/version which match all specified cc_attribute nested tasks. The filter criteria can be either just a name, or both a name and value. A delimiter can also be specified if the ClearCase attribute is set to, say, a comma delimited list of attribute values. For example <cc_attribute name="asset_name" value="asset1" delimiter=","/> will match a ClearCase element that contains attribute asset_name="asset1, asset2". If delimiter was not set, then this element would not be returned.

Attributes of cc_attribute

Attribute	Description	Required
name	The name of the attribute in the ClearCase element/version that must be set for the file to be retrieved by cc_files.	Yes
value	The value for the corresponding name attribute in ClearCase that will be matched. If value is not set then the cc_files task will return all files that have the attribute set regardless of what value they contain.	No
delimiter	If delimiter is set then the file will be retrieved by the cc_files task if any of the delimiter separated values are matched.	No

Examples of cc_files

```
<cc_files id="ccfiles" path="m:\viewdirectory" onelement="true">
    <cc_attribute name="asset_name"
                  value="@asset-name@"
                  delimiter=","/>
    <cc_attribute name="artifact_type"/>
</cc_files>
```

The previous example searches for elements within the ClearCase view directory m:\viewdirectory for any elements which have any value in the comma delimited set of ClearCase asset_name values set to the value of the @asset-name@ variable and have an artifact_type ClearCase attribute set.

cc_fileattribute

The "cc_fileattribute" task is a simple task that can be used wherever a parservalue is required, a classifier value for example. The task returns the ClearCase attribute value attached to an element/version.

Attributes of cc_fileattribute

Attribute	Description	Required
file	The ClearCase file (in a view) to that contains the attribute value.	Yes.
attribute	The name of the attribute whose value will be returned	Yes

onelement	If set to true, the ClearCase attribute used will be on the element. Otherwise, if onelement is not set or set to false the attribute used will be on the ClearCase version.	No

Examples of cc_fileattribute

```
<artifact variable="file-uri"
          retrieverid="ccfiles"
          type="by-value"
          failonerror="false"
          maxoccurs="1"
          reference=”m:\viewdirectory\vob\src\Main.java”
          version=”1.0”>
    <artifactcategory>
        <cc_fileattribute file="m:\viewdirectory\vob\src\Main.java"
                          attribute="artifact_type"
                          onelement="true"/>
    </artifactcategory>
</artifact>
```

This example creates a by-value artifact based on the contents of the ClearCase file m:\viewdirectory\vob\src\Main.java. The artifact’s category will be set to the value of the ClearCase artifact_type attribute attached to the element.

cc_fileversion

The “cc_fileversion” task is a simple task that can be used wherever a parservalue is required, a classifier value for example. The task returns the version of a ClearCase file, in a view.

Attributes of cc_fileversion

Attribute	Description	Required
file	The ClearCase file (in a view). The version of the visible file will be returned.	Yes.

Examples of cc_fileattribute

```
<artifact variable="file-uri"
          retrieverid="ccfiles"
          type="by-value"
          category=”Component”
          failonerror="false"
          maxoccurs="1"
          reference=”m:\viewdirectory\vob\src\Main.java”>
    <artifactversion>
        <cc_fileversion file="m:\viewdirectory\vob\src\Main.java"/>
    </artifactversion>
</artifact>
```

This example creates a by-value artifact based on the contents of the ClearCase file m:\viewdirectory\vob\src\Main.java. The artifact’s version will be set to the version of the ClearCase file.

Metallect IQ Server Tasks (optional)

Although IQ Server exposes resources via XML, the standard tasks within the ASAA cannot handle the relationships within the XML. Therefore, several IQ Server specific tasks have been written to handle these relationships between resources. Several of the IQ Server tasks require an XML file to process. This XML file is a resource report from IQ Server and must be generated and saved so the ASAA can use it.

The following section describes usage of the IQ server specific tasks.

iqsresourceretriever

The iqsresourceretriever task can be used both in an assembly task to retrieve IQ Server resources and in the assetfiles task to define which resources are asset candidates. By default no resources are returned by the iqsresourceretriever element, instead they must be included or excluded using the iqsresourceinclude and iqsresourceexclude tasks. If an iqsresourceretriever task is used within an assembly, the child elements are called once for every matching resource (see iqsresourceinclude and iqsresourceexclude).

Attributes of iqsresourceretriever

Attribute	Description	Required
id	ID of the retriever	No.
file	XML file exported from IQ Server to retrieve resources from.	Yes.
resultvar	If used within an assembly, the variable which is assigned the resourceid for every matching resource. This variable is available to the child tasks as they are enumerated over once for every resource.	

iqsresourceinclude and iqsresourceexclude

The iqsresourceinclude and iqsresourceexclude task are used as a subtasks of the iqsresourceretriever task and specifies which resources to include or exclude in the list of returned resources. Every resource IQ Server returns is checked against every iqsresourceinclude and iqsresourceexclude subtask. The last matching include or exclude subtask determines whether the resource is added to the list of resources retrieved by the parent iqsresourceretriever task.

Three types of parameters can be specified in an include or exclude task: type, environment, and component, which correspond to the properties on an IQ Server resource. If one or more are not specified it is assumed to be a wild card and always matches a resource unless a specified parameter does not match.
For example `<iqsresourceinclude type="PARAMETER" environment="JAVA" />` matches all resources that have a type value of "PARAMETER" and environment value of "JAVA" and anything for the component value.

Attributes of iqsresourceinclude and iqsresourceexclude

Attribute	Description	Required
type	The IQ Server type property to match against available resources	No.
environment	The IQ Server environment property to match against available resources	No.
component	The IQ Server component property to match against available resources	No.

iqsresourceparser

The iqsresourceparser may be included directly as a subtask of an assembly task, or a subtask of an iqsresourceretriever task. The purpose of the task is to make keys available for asset modifiers (classifiers, relationships, and artifacts). A resourceid must be specified for subtasks to extract various values from the resource. Since an iqsresourceparser may be called several times (if it is a child of an iqsresourceretriever), any keys with fixed names may be overwritten. Therefore another task, keyaggregator, allows keys to be qualified by the resource retrieved. See the keyaggregator description for an example.

Attributes of iqsresourceparser

Attribute	Description	Required
id	The id of this parser, used by the parserkeys task and others.	Yes.
resourceid	An IQ Server resource ID to parse values from	Yes

keyaggregator

The keyaggregator is used within a parent iqsresourceparser task to qualify keys with resources IDs to avoid them from being overwritten when multiple resources are returned from the parent parser. The name attribute includes a descriptive name (like the normal key task), but can also include a resource variable that qualifies the key. Later on you can match the keys using the name and/or part of the resourceid. The value attribute includes the property name to parse from the resource. In the case of an XML file, this can be either an attribute or element name.

Attributes of keyaggregator

Attribute	Description	Required
name	The name of the key to create. The name can be qualified by a resource id variable. See example	Yes.
value	For an XML file, this is either one or more attributes or element names, enclosed by braces, in the resource that the corresponding value for the key is set to. See example below.	Yes.

multivalue	This operates the same as the multivalue attribute for a key task.	No. default=false

Example:

```
<iqsresourceretriever id="retid" file="${iqsfile}" resultvar="interfaces">
   <iqsresourceinclude component="SOURCE_INTERFACE"/>
   <iqsresourceparser id="apprelparser" resourceid="@interfaces@">
      <keyaggregator name="iname:@interfaces@"
      value="{shortDescription}/{description}"/>
   </iqsresourceparser>
</iqsresourceretriever>
```

iqsreferenceretriever

This is a task that enumerates a set of resources that have a directional relationship to a resource of interest. It must be a child of an assembly and is not allowed in the assetfiles task. Like the resourceretriever, this task will call its child tasks once for every matching resource. The resultvar variable is set to the id of the resource retrieved and the subtasks are then called.

Attributes of iqsreferenceretriever

Attribute	Description	Required
file	The XML file to use that contains the resource specified by the resourceid attribute and the references of interest.	Yes.
resourceid	The id of the resource of interest. The list of matching resources will be determined by the direction, type, and fullyenumerated attributes.	Yes.
direction	One of INCOMING, OUTGOING, or PARENT. If set to INCOMING, resources that have an incoming relationship to the resourceid will be matched. If set to OUTGOING, resources that have an outgoing relationship from the resourceid will be matched. If set to PARENT, the parent of the resourceid will be matched.	Yes.
fullyenumerated	If set to true, all matching resources whether one level or many levels away from resourceid will be matched. In mathematical terms, the set returned from the retriever is closed.	No. default=false
type	The type of relationship to match if the direction attribute is set to INCOMING, or OUTGOING.	Yes.
resultvar	The variable to set for every matching resource. The variable can be used in child iqsresourceretriever or iqsresourceparser tasks to extract data from the matching resources.	Yes.

```
<iqsreferenceretriever file="${iqsfile}" resourceid="@asset-uri@"
direction="PARENT" resultvar="containers" fullyenumerated="true">
   <iqsresourceparser id="parentparser" resourceid="@containers@">
      <keyaggregator name="parent.@containers@" multivalue="false"
      value="${app-component}/${shortDescription}"/>
   </iqsresourceparser>
</iqsreferenceretriever>
```

Rational Team Concert (RTC) Tasks (optional)

Integration with Rational Team Concert 3.0. It can enumerate the RTC source code system to create assets based on files identified within RTC. To use the RTC tasks, the bin/rtc.properties file needs to be added to the list of tasks in the XML rules file, e.g.

```
      <taskdef file="ASSET_ADAPTER_HOME\bin\rtc.properties"/>
```

rtcconnection

The "rtcconnection" task defines a connection to the server used to retrieve RTC content. The connection can be referred to in other RTC related tasks, otherwise the most currently run rtcconnection information is used for other RTC tasks (e.g. rtcfileset).

Attributes of rtcconnection

Attribute	Description	Required
server	The URL to the RTC server. This must be a URL already defined in the Visual Studio client.	Yes.
id	In id for the connection task used in other related tasks to refer to the connection that is use by that task.	Yes.
username	The user to connect to the server	Yes
password	The password of the user to connect to the server	Yes

Nested Elements of rtcconnection

None.

Examples

Defines a connection to an RTC server located at example.com:9443/ccm/.

```
<rtcconnection id="rtcconn" server="https://example.com:9443/ccm/"
username="user" password="secret"/>
```

rtcfileinfo

The “rtcfileinfo” task is a parser that predefines a set of parserkeys that can be used to retrieve version information about a RTC file.

Attributes of rtcfileinfo

Attribute	Description	Required
item	The reference to the file in RTC that is queried for information. The item should be set to a RTC reference in the following format: “/directory/path.txt”	Yes.
id	In id for the parser used in other related tasks.	Yes.
connectionid	The id of the rtcconnection to use when retrieving files. If this attribute is not specified, it defaults to the last executed rtcconnection task.	No.
filesetid	An RTC fileset id that contains the item being referenced. It uses this fileset to identify the version of the file that is being used.	Yes.

Defined parserkeys of rtcfileinfo
The following parser keys are available in parser related tasks.

Parser key	Parser value / description
timestamp	The timestamp of the file.
content-type	The content type of the file
type	The type of the resource (“file” or “directory”).
url	A URL representing the version of the file that was found when this asset adapter task was executed. It is used alongside the RTC Artifact Source (see the Library Process Configuration Guide) to refer to files within a library. Depending on the rtcfileset this file was retrieved from, the URL will contain workspace, component, baseline, or snapshot parameters.

Nested Elements of rtcfileset

None.

Examples

This example pulls information from an item and creates an artifact from it.

```
<rtcfileinfo id="rtcitem" item="@asset-uri@" filesetid="rtcxsds"/>
<variable name="artifactreference">
   <parservalue parserid="rtcitem" parserkey="url"/>
</variable>
```

```
<artifact name="@filename@" category="schema-definition" type="by-reference"
    failonerror="false" reference="${rtc-artifactprefix}/@artifactreference@"
    file="" version="@artifactversion@"/>
```

rtcfileset

The “rtcfileset” task extends the fsfileset task enumerate files and directories stored in RTC as well as retrieve content from those files. See the fsfileset for basic information on how filesets work. This RTC task works with the version of files based on a specified baseline or snapshot. See the rtcfileinfo task for information on how to retrieve other information of the enumerated files.
The retrieved file references (possibly used in an asset-files task) contain RTC server references in the format “/directory/file.txt”

Attributes of rtcfileset

Attribute	Description	Required
dir	The directory in RTC used as a base reference to include or content. It should follow the standard server syntax for naming locations in RTC, e.g. /directory/path	Yes.
id	In id for the fileset used in other related tasks.	Yes.
connectionid	The id of the rtcconnection to use when retrieving files. If this attribute is not specified, it defaults to the last executed rtcconnection task.	No.
workspace	The workspace to use for enumerating and retrieving files.	One of workspace, baseline, or snapshot is required.
component	The component the files reside in.	Yes.
baseline	A baseline to use for enumerating and retrieving files.	One of workspace, baseline, or snapshot is required.
snapshot	A snapshot to use for enumerating and retrieving files.	One of workspace, baseline, or snapshot is required.

Nested Elements of rtcfileset

See fsfileset.

Examples

This example executes an assembly named “xsds” once for each XSD found in the RTC system, setup by a rtcconnection task (not shown).

```
<assetfiles id="xsds" assemblyid="xsds">
```

```
	<rtcfileset component="WebServices" id="rtcxsds" dir="${rtc-folder}"
workspace="TestProject Stream">
		<include name="**/*.xsd"/>
	</rtcfileset>
</assetfiles>
```

Team Foundation Server (TFS) Tasks (optional)

The ASAA tasks require assemblies and connection information provided by the Visual Studio TFS client. Therefore the TFS client will need to be available on the system hosting TFS.

Depending on which version of TFS you are using, you may need to instruct the Asset Adapter to use the version of assemblies for the version of TFS you have installed. Edit the ASAA_HOME/bin/ant153cmd.exe.config file and insert this section after the </startup> tag. Modify the newVersion attributes and set them as follows. If using TFS 2005, use version “8.0.0.0”. If using TFS 2008, use version “9.0.0.0”. If using TFS 2010, use version “10.0.0.0”.

```
<runtime>
	<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
		<dependentAssembly>
			<assemblyIdentity name="Microsoft.TeamFoundation.Client"
			publicKeyToken="b03f5f7f11d50a3a" />
			<bindingRedirect oldVersion="8.0.0.0-10.0.0.0"
			newVersion="10.0.0.0"/>
		</dependentAssembly>
		<dependentAssembly>
			<assemblyIdentity
			name="Microsoft.TeamFoundation.VersionControl.Client"
			publicKeyToken="b03f5f7f11d50a3a" />
			<bindingRedirect oldVersion="8.0.0.0-10.0.0.0"
			newVersion="10.0.0.0"/>
		</dependentAssembly>
	</assemblyBinding>
</runtime>
```

tfsconnection

The “tfsconnection” task defines a connection to the server used to retrieve TFS content. The connection can be referred to in other TFS related tasks, otherwise the most currently run tfsconnection information is used for other TFS tasks (e.g. tfsfileset).

Attributes of tfsconnection

Attribute	Description	Required
Server	The URL to the TFS server. This must be a URL already defined in the Visual Studio client.	Yes.
Id	In id for the connection task used in other related tasks to refer to the connection that is use by that task.	Yes.

Nested Elements of tfsconnection

None.

Examples

Defines a connection to a TFS server located at example.com\FirstCollection.

```
<tfsconnection id="tfsconn" server="example.com\FirstCollection"/>
```

tfsfileinfo

The “tfsfileinfo” task is a parser that predefines a set of parserkeys that can be used to retrieve version information about a TFS file.

Attributes of tfsfileinfo

Attribute	Description	Required
item	The reference to the file in TFS that is queried for information. The item should be set to a TFS item server reference in the following format: “$/directory/path.txt”	Yes.
id	In id for the parser used in other related tasks.	Yes.
connectionid	The id of the tfsconnection to use when retrieving files. If this attribute is not specified, it defaults to the last executed tfsconnection task.	No.

Defined parserkeys of tfsfileinfo
The following parser keys are available in parser related tasks.

Parser key	Parser value / description
changeset-id	The changeset ID of the file.
checkin-date	The date the file was checked in.
size	The size of the file.
deletion-id	The deletion identifier of the file, probably 0.
encoding	An number that represents the encoding (code page/code set) of the file.
id	The item’s id.
type	“file” or “dir” depending on what type of item is referred to.
path	The path of the file
version-url	A URL representing the version of the file that was found when this asset adapter task was executed. It is used alongside the TFS Artifact Source to refer to files within a library. This does not include the “$/” prefix and is escaped for use in a URL.
latest-url	A URL representing the latest version of the file. It is used alongside the TFS Artifact Source to refer to files within a library.

	This does not include the "$/" prefix and is escaped for use in a URL. When used with the TFS artifact source this parser key will always refer to the same version of the file, regardless of whether newer files have been checked in to TFS.

Nested Elements of tfsfileset

None.

Examples

This example pulls information from an item and creates an artifact from it.

```
<tfsfileinfo id="tfsitem" item="@asset-uri@"/>
<variable name="artifactreference">
      <parservalue parserid="tfsitem" parserkey="latest-url"/>
</variable>
<variable name="artifactversion">
      <parservalue parserid="tfsitem" parserkey="changeset-id"/>
</variable>
<artifact name="@filename@" category="message-definition" type="by-reference"
   failonerror="false" reference="${tfs-artifactprefix}/@artifactreference@"
   file="" version="@artifactversion@"/>
```

tfsfileset

The "tfsfileset" task extends the fsfileset task enumerate files and directories stored in TFS as well as retrieve content from those files. See the fsfileset for basic information on how filesets work. This TFS task always works with the latest version of files in TFS. See the tfsfileinfo task for information on how to retrieve version information of the enumerated files.
The retrieved file references (possibly used in an asset-files task) contain TFS server references in the format "$/directory/file.txt"

Attributes of tfsfileset

Attribute	Description	Required
dir	The directory in TFS used as a base reference to include or content. It should follow the standard server syntax for naming locations in TFS, e.g. $/directory/path	Yes.
id	In id for the fileset used in other related tasks.	Yes.
connectionid	The id of the tfsconnection to use when retrieving files. If this attribute is not specified, it defaults to the last executed tfsconnection task.	No.

Nested Elements of tfsfileset

See fsfileset.

Examples

This example executes an assembly named “wsdls” once for each wsdl found in the TFS system, setup by a tfsconnection task (not shown).

```
<assetfiles id="assetfiles" assemblyid="wsdls">
<tfsfileset id="tfscodefiles" dir="$/WebServices " >
             <include name="**/*.wsdl"/>
      </tfsfileset>
</assetfiles>
```

WebDAV Tasks (optional)

webdavfileset

The “webdavfileset” task is a Retriever that works with a repository that supports the WebDAV protocol and produces a list of files and/or directories that match patterns specified in an include/exclude list. This mechanism is very similar to Ant’s “fileset” task.

Attributes of webdavfileset

Attribute	Description	Required
serverurl	The URL to the WebDAV server.	Yes.
dir	The starting directory.	Yes.
casesensitive	Set to “true” or “false”. Determines if filenames should be matched with respect to case. Default is “false”.	No.
id	A unique string to identify this Retriever. The string is arbitrary but must be unique against all other Retrievers in this assembly.	No.
type	Return only files (“file”), only directories (“dir”), or all (“all”). Default is “file”.	No.
user	WebDAV user.	No.
password	Clear text user password. Deprecated. Use encryptedpassword attribute.	No.
encryptedpassword	Encrypted user password created using the encryptpassword script. See Encrypting Passwords.	No.

Nested Elements of webdavfileset

See fsfileset.

Examples

Retrieve all Java files from a WebDAV server using user credentials with an encrypted password.

```
<webdavfileset serverurl="http://mywebdav:8088/dav" user="rsmith"
  encryptedpassword="yPR2NO7Caj4=" dir="/project/source/dir"
  id="sourcefiles">
    <include name="**/*.java"/>
</webdavfileset>
```