BEA AquaLogic® Enterprise Repository

Harvesting Apache Ant Tasks

Version 3.0 RP1
Revised: February, 2008

Table of Contents

List of Tables

List of Examples

Preface

This document provides instructions for incorporating the ant harvesting tasks into build scripts.

Chapter 1. Overview

In a typical Apache Ant build configuration, there is a rich set of metadata available for harvesting. This metadata reflects the software assets that are being manipulated by the scripts, and the relationships of those assets to each other.

The ALER Ant Harvesting tasks provide a mechanism by which this information can be harvested from Ant build environments. This mechanism provides for the creation and modification of assets in ALER, through a set of Ant Tasks that make calls through REX. There are three tasks:

- **registry.assetquery**
- **registry.assetcreate**
- **registry.assetupdate**

Requirements

The harvesting tasks are compatible with the following configuration of software:

Ant Harvesting Tasks

- Version 3.0 RP1 of ALER and the REX API.
- Java2 SDK v 1.4.x and higher
- Apache Ant Version 1.6.x and higher

Installation

The harvesting tasks are installable by copying harvesting-VERSION.jar and its dependencies to ant's classpath, typically $ANT_HOME/lib.

The ant tasks are then made available using the standard task extension mechanisms of Apache Ant. There are two means of registering the tasks: using the 'taskdef' task, and using XML namespace declarations.

In the first case, tasks can be registered using syntax similar to the following:

Example 1.1. Task Definition

```
<taskdef file="registry-tasks.properties"/>
```

Where the registry-tasks.properties contains the following:

Example 1.2. registry-tasks.properties (contents)

```
registry.assetquery=com.flashline.registry.ant.harvesting.AssetQueryTask
registry.assetcreate=com.flashline.registry.ant.harvesting.AssetCreateTask
registry.assetupdate=com.flashline.registry.ant.harvesting.AssetUpdateTask
```

In the second case, inclusion of a namespace declaration in the opening project tag can associate the tasks with an antlib definition which is present in the task jar:

Example 1.3. Opening 'project' tag referencing harvesting antlib

```
<project name="Asset Create Examples" default="all"
    xmlns:registry="antlib:com.flashline.registry.ant.harvesting">

    ...

    <target name="create">
     <registry:assetcreate configuration="warn" ...>

        ...
```

Chapter 2. Task Reference

assetquery

Description

The **assetquery** task enables a user to query for the presence of assets in ALER using sets of search criteria.

The task is configurable to **fail** or **warn,** and defaults to **warn** if not specified explicitly by the user. If the task is configured to **fail** and no matching assets are found, a BuildException is thrown.

If one or more asset matches are found, the ant property specified by the 'ret_assetids' parameter is set to a comma-delimited list of the matching asset id values. If the task is configured to 'warn' and no assets are matched, a warning is logged, and the property is set to the empty string.

Parameters

Table 2.1. Parameters for assetquery

Attribute	**Description**	**Required**
RegistryUsername	A valid ALER username.	Yes
RegistryPassword	ALER password.	Yes
RegistryURL	ALER URL	Yes
configuration	Controls whether the failure to match any asset causes an error, or generates a warning.	No (can be either 'warn' or 'fail', and defaults to 'warn')
ret_assetids	Property which will receive query results.	Yes

Nested Elements

Table 2.2. Nested Elements for assetquery

Element	**Description**	**Required**
AssetCriteria	Query criteria.	Yes

The assetcriteria Element

A user must provide an **assetcriteria** element for the assetquery task. Assetcriteria must in turn contain at least one searchterm element.

Table 2.3. Nested Elements for assetcriteria

Element	**Description**	**Required**
searchterm	One a of set of nested query search terms.	Yes - at least one

The searchterm Element

Table 2.4. Parameters for searchterm

Attribute	**Description**	**Required**
name	Search term name	Yes
value	Search term value	Yes
op	Search term match operation	No. Defaults to 'eq'. See the Valid search term operations table for possible values.

Table 2.5. Valid searchterms

Name (value of 'name' attribute in 'searchterm' element)	Description
id	Search for the specified ID.
name	Search for the specified name.
/asset/mandatory-data/name	Search for the specified name.
version	Search for the specified version string.
/asset/mandatory-data/version	Search for the specified version string.
assettypeid	Search for Assets of an AssetType with the specified ID.
archetypeid	Search for Assets of an AssetType with Archetype with specified ID.
description	Search for the specified descrition.
/asset/mandatory-data/description	Search for the specified descrition.
keyword	Search for the specified keyword.
/asset/mandatory-data/keywords/keyword	Search for the specified keyword.
hashinfo	Search for the specified SFID.
/asset/custom-data/hashInfos/hashInfo	Search for the specified SFID.
project	Search for assets that have a producing project with the specified ID.
/asset/mandatory-data/producing-projects/project	Search for assets that have a producing project with the specified ID.
general	Search for a string that must be present in name, version, description, keywords, categorizations or indexed fields.
exactMatch	Set whether to return only exact matches.
browsableOnly	Set whether to return only browsable assets. This behavior mimics how asset searches work in the ALER user interface. For example, only registered assets will be returned unless the Assets-In-Progress feature is enabled.
offset	the starting index (zero-based) from the matching results
limit	the number of results ot return, starting at the starting index specified by "offset"

Table 2.6. Valid searchterms (continued)

Name (value of 'name' attribute in 'searchterm' element)	Description
categorization.#	Search for Assets with CategorizaionType ID=#, and an assigned Categorization equal to the value of the SearchTerm.
custom-data/*	Search for asset with an arbitrary element in customdata equal to the value specified in the SearchTerm.
/asset/custom-data/*	Search for asset with an arbitrary element in customdata equal to the value specified in the SearchTerm.
status	Search on the active status of the Asset. Value should be one of 0=Active, 10=Inactive, 20=Retired, 30=Deleted, 40=Incomplete.
uniqueelement	Search for an asset whos unique element is equal to the SearchTerm's value
underconstruction	Search for assets that have a status of UNDERCONSTRUCTION
submission	Search for assets that have a status of SUBMITTED.
rejected	Search for assets that have a status of REJECTED.
queued	Search for assets that have a status of QUEUED.
registered	Search for assets that havea a status of REGISTERED.
createdbyid	Search for assets that were created by the specified user ID..
submittedbyid	Search for assetds that were submitted by the specified ID.
assetstatus	Search for an asset with a registration status equal to the specified value. One of 0=undefined, 10=unsubmitted, 50=submitted, 51=pending review, 52=under review, 55=rejected, 100=registered.

Table 2.7. Valid searchterm operations

Operation (value of 'op' attribute in 'searchterm' element)	Description
eq	Equals
eqic	Equals (case insensitive)
like	Like
likeic	Like (case insensitive)
neq	Not equal to
lt	Less than
lte	Less than or equal to
gt	Greater than
gte	Greater than or equal to
in	In
notin	Not in
btw	Between

Examples

Example 2.1. Simple Query

```
<registry:assetquery ret_assetids="myFoundAssetIDs">
    <assetcriteria>
        <searchterm key="name" value="myAsset" />
        <searchterm key="version" value="1.0" />
    </assetcriteria>
</registry:assetquery>
```

Example 2.2. Query with 'op' attribute

```
<registry:assetquery ret_assetids="myFoundAssetIDs">
    <assetcriteria>
        <!-- match name 'function' using case-insensitive 'like',
             version greater than 1.0 -->
        <searchterm key="name" value="function" op="likeic" />
        <searchterm key="version" value="1.0" op="gt" />
    </assetcriteria>
</registry:assetquery>
```

assetcreate

Description

The **assetcreate** task creates a new asset in ALER

The **name** and **version** must be supplied.

Parameters

Table 2.8. Parameters for assetcreate

Attribute	**Description**	**Required**
RegistryUsername	A valid ALER username.	Yes
RegistryPassword	A valid ALER password.	Yes
RegistryURL	Registry URL	Yes
configuration	warn/fail configuration: whether to fail, or just print a warning, if an error is encountered	No (defaults to 'fail')
name	Asset name	Yes
version	Asset version	Yes

Nested elements

Table 2.9. Nested Elements for assetcreate

Element	Description	Required
appliedComplianceTemplateProjects	Applied Compliance Template Projects	No
appliedPolicies	Applied Policies	No
assettype	The asset type to assign to the newly created asset.	Yes
categorizations	Asset categorizations.	No
contacts	Contacts	No
customAccessSettings	CAS	No
customData	Custom data	No
description	Description of asset.	No
files	Files	No
keywords	Keywords	No
notificationEmail	Notification email address	No
policyAssertionResults	Policy Assertion Results	No
producingProjects	Producing projects	No
relationships	Relationships	No
registrationStatus	The asset's registration status. The content of the 'setting' attribute of this element should be one of the following: unsubmitted, submitted, pending_review, under_review, rejected or registered.	No
sfids	HashInfos (SFIDs)	No
status	The asset's active status. The content of the 'setting' attribute of this element should be one of the following: active, inactive, retired, deleted or incomplete.	No
uniqueElement	UniqueElement	No
vendor	Vendor	No

appliedComplianceTemplateProjects

The **appliedComplianceTemplateProjects** element may contain any number of nested **project** tags, each of which must have a single **id** attribute that specifies the relevant project.

Example 2.3. Usage of appliedComplianceTemplateProjects

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <appliedComplianceTemplateProjects>
    <project id="1234" />
    <project id="5678" />
   </appliedComplianceTemplateProjects>
  </registry.assetcreate>
```

appliedPolicies

Example 2.4. Usage of appliedPolicies

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...

            <appliedpolicies>
                <policy id="1234" />
                <policy id="5678" />
            </appliedpolicies>
</registry:assetcreate>
```

assettype

The **assettype** element specifies the assettype to assign to the newly created asset. Either the **id** or **name** attribute must be specified in order to identify the assettype to assign.

Example 2.5. Usage of assettype

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
<assettype id="${assettypeid}" />
</registry:assetcreate>
```

categorizations

The categorizations nested element contains one or more categorization elements. The nested categorization elements may supply either the **id** attribute or the **name** attribute, but not both.

Example 2.6. Usage of categorizations

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <categorizations>
        <categorization id="123"/>
        <categorization name="Category-A"/>
    </categorizations>
</registry:assetcreate>
```

contacts

A **contacts** element may contain one or more **contact** element, which supports the attributes listed below:

Table 2.10. Contact Attributes

Attribute	**Description**	**Required**
name	name string for the contact	no
id	Id of the contact. Contacts may be specified for deletion using this attribute.	no
type	type of contact	no

Table 2.11. Contact Attributes (cont.)

Attribute	Description	Required
email	contact email address	no
phonenumber	contact phone number	no
company	company name for the contact	no
street1	first line of the street address for the contact	no
street2	second line of the street address for the contact	no
city	city for the address of the contact	no
state	state for the address of the contact	no
zip	zip-code for the address of the contact	no
country	country for the address of the contact	no
faxnumber	the contact fax number	no

Example 2.7. Usage of contacts

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <contacts>
    <contact id="1234" />
    <contact city="Cleveland"
     company="BEA"
     name="Mike Miklavcic" />
   </contacts>
  </registry.assetcreate>
```

customAccessSettings

Example 2.8. Usage of customAccessSettings

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <customaccesssettings>
    <setting name="dmc-asset-cas-1" />
    <setting name="customcas2" />
   </customaccesssettings>
  </registry.assetcreate>
```

customData

The customData nested element contains one or more entry elements. The nested entry elements must supply the **xpath** attribute, in order to identify the path of the customData element to update. The body of each **entry** element contains the data to be updated.

Example 2.9. Usage of customData

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <customdata>
        <entry xpath="custom-data/textcustomdata">
 <![CDATA[[Blah, some blah, some more blah.]>
 </entry>
 <entry xpath="foo/bar/baz">
 <![CDATA[[
 <mynode>
     <mychildnode>A value</mychildnode>
 </mynode>]>
        </entry>
    </customdata>
</registry:assetcreate>
```

description

The description element may contain text or a CDATA section.

Example 2.10. Usage of description

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    <description>
        The description of this asset.
    </description>
</registry:assetcreate>
```

files

The **files** nested element can contain a series of **file** elements, each of which must supply at least **name**, **url** and **desc** attributes.

Table 2.12. Attributes of files element

Name	Type	Description	Required?
name	string	Name of file	yes
url	string	the path to the file. If the file is located in ALER the protocol will be rep://	yes
desc	string	Description of the file	yes
filestorage	string	The associated text for the file. Usually describes the storage for the file.	no
filetypeid	string	The file type ID of the file.	no
securitysettings	space-delimited string	The names of the custom access settings that control access to the file, as a space-delimited string.	no

Example 2.11. Usage of files

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <files>
        <file name="Overwrittenfile2"
          url="http://www.newFoo.com/OWfile2.xml"
          desc="Overwritten2" securitysettings="x y z"/>
        <file name="Overwrittenfile1"
          url="http://www.newFoo.com/OWfile1.xml"
          desc="Overwritten1"/>
    </files>
</registry:assetcreate>
```

keywords

The keywords nested element contains one or more keyword elements. The nested keyword elements accept the single attribute **value**, which contains the literal keyword.

Example 2.12. Usage of keywords

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <keywords>
        <keyword value="${keyvalue}"/>
    </keywords>
</registry:assetcreate>
```

notificationEmail

The notificationEmail element must supply the attribute **value**, which contains the notification email to assign to this asset.

The notificationEmail may be deleted by nesting it within the assetupdate task's 'delete' element.

Example 2.13. Usage of notificationEmail

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    <notificationEmail value="email@somefictitioushost.com"/>
</registry:assetcreate>
```

policyAssertionResults

The policyAssertionResults element contains one or more assertionresult elements. Any character data contained by the element is stored as the assertion result's 'evaluation information.' If this value is in the form of a URL, the content of the URL should contain the evaluation information.

Table 2.13. assertionresult attributes

Attribute	Description	Required
assertionid	The ID of the policy assertion.	Yes
date	The date of the result. The date must be of the format 'yyyy-mm-dd'.	Yes
value	If supplied, must be one of 'pass', 'fail' or '' (the empty string).	No

Example 2.14. Usage of policyAssertionResults

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <policyassertionresults>
    <assertionresult assertionid="1234" date="2007-06-01" value="pass">
     some string text
    </assertionresult>
   </policyassertionresults>
  </registry.assetcreate>
```

producingProjects

The producingProjects nested element contains one or more producingProject elements. The nested producingProject elements may supply either the **id** attribute or the **name** attribute, but not both, in order to identify the producing project.

Example 2.15. Usage of producingProjects

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <producingProjects>
        <producingProject id="123"/>
        <producingProject name="Project-Zed"/>
    </producingProjects>
</registry:assetcreate>
```

registrationstatus

The registrationstatus element must supply the attribute **setting**, which can be one of the following: unsubmitted, submitted, pending_review, under_review, rejected or registered.

Example 2.16. Usage of registrationstatus

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    <registrationstatus setting="submitted"/>
</registry:assetcreate>
```

relationships

The relationships nested element contains one or more relationship elements. The nested relationship elements may supply either the **id** attribute or the **name** attribute, but not both, and the **secondaryassetid** attribute, which identifies the related asset.

Example 2.17. Usage of relationships

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    ...
    <relationships>
        <relationship id="123" secondaryassetid="50054"/>
        <relationship name="RelationshipName" secondaryassetid="50054"/>
    </relationships>
</registry:assetcreate>
```

sfids

Example 2.18. Usage of sfids

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <sfids>
    <sfid value="1000200030004000"/>
    <sfid value="1000200030004001"/>
   </sfids>
  </registry.assetcreate>
```

status

The status element must supply the attribute **setting**, which can be 'active', 'delete', 'inactive', 'retired', or 'incomplete'.

Example 2.19. Usage of status

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    <status setting="active"/>
</registry:assetcreate>
```

uniqueElement

Custom additional unique key which can be applied to Assets. Requires a special system setting to enable and another to specify whether the value is to be unique across all Types or per Type.

Table 2.14. Attributes

Attribute	Description	Required
id	Long representing an ID	no: one of id or value must be specified though
value	String	no: one of id or value must be specified though

Example 2.20. Usage of uniqueelement

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <uniqueelement id="1234"/>
  </registry.assetcreate>
```

vendor

The vendor element may supply either a **id** or **name** attribute identifying the vendor to assign to the asset.

Example 2.21. Usage of vendor

```
<registry:assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
registryURL="${registry.url}"
name="Asset"
version="1.0">
    <vendor name="comco"/>
</registry:assetcreate>
```

assetupdate

Description

The **assetupdate** task updates an asset in ALER with information provided by the ant task.

There are two, mutually exclusive means of specifying the assets which should be updated by the assetupdate task:

1. The **update_assetids** parameter containing a comma delimited list of asset IDs to update.

2. A nested **assetcriteria** element which matches a set of assets to update.

Parameters

The RegistryUsername, RegistryPassword, RegistryURL and configuration parameters are common to all harvesting tasks. The update_assetids parameter is required if an assetcriteria element is not supplied.

Table 2.15. Parameters for assetupdate

Attribute	Description	Required
RegistryUsername	A valid ALER username.	Yes
RegistryPassword	ALER password.	Yes
RegistryURL	ALER URL	Yes
configuration	This parameter specifies the task's behavior in the event of an error. Depending on the value of this parameter, the update task will either fail, or emit a warning message to the log. Acceptable values **warn** or **fail**.	No (default is 'fail')
update_assetids	A comma-delimited string containing the IDs of the assets to update.	No, but required if a nested **assetcriteria** element is not provided.

Nested Elements

Table 2.16. Nested Elements for assetupdate

Element	Description	Required
assetcriteria	An assetcriteria element: assets which match these criteria will be updated.	Yes, unless the **update_assetids** parameter is supplied.
assettype	The asset type to assign to the updated asset.	No
appliedPolicies	Policies applied to this asset..	No
categorizations	Asset categorizations	No
customData	Custom Data	No
description	Description	No
contacts	contacts	No
files	Files	No
keywords	Keywords	No
notificationEmail	Notification Email	No
producingProjects	Producing Projects	No
relationships	Relationships	No
status	Status	No
delete	Tag which allows deletion of single-valued elements (description, notificationEmail, uniqueelement).	No

appliedComplianceTemplateProjects

The **appliedComplianceTemplateProjects** element may contain any number of nested **project** tags, each of which must have a single **id** attribute that specifies the relevant project.

Project elements inside of appliedComplianceTemplateProjects maybe nested within the standard add/overwrite/delete elements. Assertionresults are matched for deletion by 'id' attribute.

Example 2.22. Usage of appliedComplianceTemplateProjects

```
<registry.assetupdate ...>
    ....
    <appliedComplianceTemplateProjects>
        <delete>
            <project id="1234" />
        </delete>
        <add>
            <project id="5678" />
        </add>
    </appliedComplianceTemplateProjects>
</registry.assetupdate>
```

appliedPolicies

The appliedPolicies element contains one or more policy elements, each of which identifies the policy to be applied to the updated asset through an 'id' attribute.

Policy elements inside of appliedPolicies maybe nested within the standard add/overwrite/delete elements.

Example 2.23. Usage of appliedPolicies

```
<registry:assetupdate ...>

    ...

    <appliedpolicies>
        <add><policy id="1234" /></add>
        <delete><policy id="5678" /></delete>
    </appliedpolicies>
</registry:assetcreate>
```

assetcriteria

The nested **assetcriteria** element has the same format as the **assetcriteria** element used in the **assetquery** task.

The following nested elements must contain a single value, which will be assigned to the asset upon successful update.

assettype

The **assettype** element specifies the assettype to assign to the updated asset. Either the **id** or **name** attribute must be specified in order to identify the assettype to assign.

Example 2.24. Usage of assettype

```
<registry:assetupdate ...>
    <assettype id="${assettypeid}" />
</registry:assetupdate>
```

categorizations

The categorizations nested element contains one or more categorization elements. The nested categorization elements may supply either the **id** attribute or the **name** attribute, but not both.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.25. Usage of categorizations

```
<registry:assetupdate update_assetids="999">
    ...
    <categorizations>
        <delete>
            <categorization id="123"/>
 </delete>
        <add>
     <categorization name="Category-A"/>
        </add>
    </categorizations>
</registry:assetupdate>
```

contacts

A **contacts** element may contain one or more **contact** element, which supports the attributes in the table below.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Add semantics: Creates a new contact given the information provided. ID attribute is disregarded.

Update semantics: Specifying an ID will update the contact for not only that asset, but for all assets pointing to that contact ID.

Delete semantics: Specifying the ID will delete that contact from the asset updates are being performed on, however the contact will still remain in the system with any remaining asset associations.

Table 2.17. Attributes

Attribute	Description	Required
name	name string for the contact	no
id	ID for the contact	no
type	type of contact	no
email	contact email address	no
phonenumber	contact phone number	no
company	company name for the contact	no
street1	first line of the street address for the contact	no
street2	second line of the street address for the contact	no
city	city for the address of the contact	no
state	state for the address of the contact	no
zip	zip-code for the address of the contact	no
country	country for the address of the contact	no
faxnumber	the contact fax number	no

Example 2.26. Usage of contacts

```
<registry.assetupdate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
    registryURL="${registry.url}"
    update_assetids="999,998">
    <contacts>
        <delete>
     <contact id="1234" />
        </delete>
        <add>
     <contact city="Cleveland"
         company="BEA"
  name="Mike Miklavcic" />
        </add>
    </contacts>
</registry.assetupdate>
```

customAccessSettings

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.27. Usage of customAccessSettings

```
<registry.assetupdate ...>
   <customaccesssettings>
                           <overwrite>
                               <setting name="dmc-asset-cas-1" />
    <setting name="customcas2" />
                           </overwrite>
   </customaccesssettings>
  </registry.assetupdate>
```

customData

The customData nested element contains one or more entry elements. The nested entry elements must supply the **xpath** attribute, in order to identify the path of the customData element to update. The body of each **entry** element contains the data to be updated.

Example 2.28. Usage of customData

```
<registry:assetupdate update_assetids="999">
    ...
    <customdata>
        <entry xpath="custom-data/textcustomdata">
     <![CDATA[[Blah, some blah, some more blah.]>
 </entry>
 <entry xpath="foo/bar/baz">
     <![CDATA[[
         <mynode>
      <mychildnode>A value</mychildnode>
  </mynode>]>
        </entry>
    </customdata>
</registry:assetupdate>
```

delete

The delete element provides a means by which single-valued tags such as notificationEmail may be deleted from the asset.

Contained elements need not specify any attributes.

Table 2.18. Nested elements for delete

Tag name
notificationEmail
uniqueElement

Example 2.29. Usage of delete

```
<registry:assetupdate update_assetids="999">
  <delete><notificationEmail/></delete>
</registry:assetupdate>
```

description

The description element may contain text or a CDATA section.

The description may be deleted by nesting it within the assetupdate task's 'delete' element.

Example 2.30. Usage of description

```
<registry:assetupdate update_assetids="999">
    <description>
        The description of this asset.
    </description>
</registry:assetupdate>
```

files

The **files** nested element can contain a series of **file** elements, each of which must supply **name**, **url** and **desc** attributes.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Table 2.19. Attributes of files element

Name	Type	Description	Required?
name	string	Name of file	yes
url	string	the path to the file. If the file is located in ALER the protocol will be rep://	yes
desc	string	Description of the file	yes
filestorage	string	The associated text for the file. Usually describes the storage for the file.	no
filetypeid	string	The file type ID of the file.	no
id	string	The id of the file.	no
securitysettings	space-delimited string	The names of the custom access settings that control access to the file, as a space-delimited string.	no

Example 2.31. Usage of files

```
<registry:assetupdate update_assetids="999">
    ...
    <files>
        <delete>
            <file name="Overwrittenfile2"
              url="http://www.newFoo.com/OWfile2.xml"
              desc="Overwritten2"/>
 </delete>
        <add>
     <file name="Overwrittenfile1"
              url="http://www.newFoo.com/OWfile1.xml"
              desc="Overwritten1"/>
        </add>
    </files>
</registry:assetupdate>
```

keywords

The keywords nested element contains one or more keyword elements. The nested keyword elements accept the single attribute **value**, which contains the literal keyword.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.32. Usage of keywords

```
<registry:assetupdate update_assetids="999">
    ...
    <keywords>
        <delete>
            <keyword value="oldkey"/>
 </delete>
        <add>
     <keyword value="${keyvalue}"/>
        </add>
    </keywords>
</registry:assetupdate>
```

notificationEmail

The notificationEmail element must supply the attribute **value**, which contains the notification email to assign to this asset.

The notificationEmail may be deleted by nesting it within the assetupdate task's 'delete' element.

Example 2.33. Usage of notificationEmail

```
<registry:assetupdate update_assetids="999">
    <notificationEmail value="email@somefictitioushost.com"/>
</registry:assetupdate>
```

policyAssertionResults

The policyAssertionResults element contains one or more assertionresult elements. Any character data contained by the element is stored as the assertion result's 'evaluation information.' If this value is in the form of a URL, the content of the URL should contain the evaluation information.

Assertionresult elements inside of appliedPolicies maybe nested within the standard add/overwrite/delete elements. Assertionresults are matched for deletion by the values of the evaluationdate and evaluationvalue tags.

Table 2.20. assertionresult attributes

Attribute	Description	Required
assertionid	The ID of the policy assertion.	Yes
date	The date of the result. The date must be of the format 'yyyy-mm-dd'.	Yes
value	If supplied, must be one of 'pass', 'fail' or '' (the empty string).	No

Example 2.34. Usage of policyAssertionResults

```
<registry.assetupdate ...>
    <policyassertionresults>
        <overwrite>
            <assertionresult id="1234" date="2006-09-25">
                some string text
            </assertionresult>
        </overwrite>
    </policyassertionresults>
</registry.assetupdate>
```

producingProjects

The producingProjects nested element contains one or more producingProject elements. The nested producingProject elements may supply either the **id** attribute or the **name** attribute, but not both, in order to identify the producing project.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.35. Usage of producingProjects

```
<registry:assetupdate ...>
    ...
    <producingProjects>
        <delete><producingProject id="123"/></delete>
        <add><producingProject name="Project-Zed"/></add>
    </producingProjects>
</registry:assetupdate>
```

registrationstatus

The registrationstatus element must supply the attribute **setting**, which can be one of 'undefined', 'unsubmitted', 'submitted', 'pending review', 'under review', 'rejected', or 'registered'.

Example 2.36. Usage of registrationstatus

```
<registry:assetupdate update_assetids="999">
    <registrationstatus setting="submitted"/>
</registry:assetupdate>
```

relationships

The relationships nested element contains one or more relationship elements. The nested relationship elements may supply either the **id** attribute or the **name** attribute, but not both, and the **secondaryassetid** attribute, which identifies the related asset.

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.37. Usage of relationships

```
<registry:assetupdate update_assetids="999">
    ...
    <relationships>
        <delete>
            <relationship id="123" secondaryassetid="50054"/>
 </delete>
        <add>
     <relationship name="Category-A" secondaryassetid="50054"/>
        </add>
    </relationships>
</registry:assetupdate>
```

sfids

These elements may be nested in add, overwrite or delete elements, in order to specify the type of update action to take. If an enclosing element is not used, an 'add' update action will be taken by default.

Example 2.38. Usage of sfids

```
<registry.assetupdate ...>
    <sfids>
 <overwrite>
            <sfid value="1000200030004000"/>
     <sfid value="1000200030004001"/>
        </overwrite>
    </sfids>
</registry.assetupdate>
```

status

The status element must supply the attribute **setting**, which can be 'active', 'delete', 'inactive', 'retired', or 'incomplete'.

Example 2.39. Usage of status

```
<registry:assetupdate update_assetids="999">
    <status setting="active"/>
</registry:assetupdate>
```

uniqueElement

Custom additional unique key which can be applied to Assets. Requires a special system setting to enable and another to specify whether the value is to be unique across all Types or per Type.

Table 2.21. Attributes

Attribute	Description	Required
id	Long representing an ID	no: one of id or value must be specified though
value	String	no: one of id or value must be specified though

uniqueElement does not need to be nested in an add or overwrite element, because the semantics of the two operations are identical. In order to delete a uniqueElement, nest it within a **delete** element: in this case, no attributes need to be specified to uniqueElement itself.

Example 2.40. Usage of uniqueelement

```
<registry.assetcreate configuration="warn"
   registryUsername="${registry.user}"
   registryPassword="${registry.password}"
   registryURL="${registry.url}"
   name="${assetname}"
   version="1.0">
   <assettype name="Harvesting Type A"/>
   <uniqueelement id="1234"/>
  </registry.assetcreate>
```

Example 2.41. Usage of uniqueelement (deleting)

```
<registry.assetcreate configuration="warn"
    registryUsername="${registry.user}"
    registryPassword="${registry.password}"
    registryURL="${registry.url}"
    name="${assetname}"
    version="1.0">
    <assettype name="Harvesting Type A"/>
        <delete>
            <uniqueelement/>
        </delete>
</registry.assetcreate>
```

vendor

The vendor element may supply either a **id** or **name** attribute identifying the vendor to assign to the asset.

Example 2.42. Usage of vendor

```
<registry:assetupdate ...>
    <vendor name="comco"/>
</registry:assetupdate>
```