# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>vii</td>
</tr>
<tr>
<td>1. Overview</td>
<td>8</td>
</tr>
<tr>
<td>Requirements</td>
<td>8</td>
</tr>
<tr>
<td>Installation</td>
<td>8</td>
</tr>
<tr>
<td>2. Task Reference</td>
<td>10</td>
</tr>
<tr>
<td>assetquery</td>
<td>10</td>
</tr>
<tr>
<td>Description</td>
<td>10</td>
</tr>
<tr>
<td>Parameters</td>
<td>10</td>
</tr>
<tr>
<td>Nested Elements</td>
<td>10</td>
</tr>
<tr>
<td>The assetcriteria Element</td>
<td>10</td>
</tr>
<tr>
<td>The searchterm Element</td>
<td>11</td>
</tr>
<tr>
<td>Examples</td>
<td>14</td>
</tr>
<tr>
<td>assetcreate</td>
<td>14</td>
</tr>
<tr>
<td>Description</td>
<td>14</td>
</tr>
<tr>
<td>Parameters</td>
<td>15</td>
</tr>
<tr>
<td>Nested elements</td>
<td>16</td>
</tr>
<tr>
<td>assetupdate</td>
<td>26</td>
</tr>
<tr>
<td>Description</td>
<td>26</td>
</tr>
<tr>
<td>Parameters</td>
<td>26</td>
</tr>
<tr>
<td>Nested Elements</td>
<td>27</td>
</tr>
</tbody>
</table>
List of Tables

2.1. Parameters for assetquery ................................................................. 10
2.2. Nested Elements for assetquery ...................................................... 10
2.3. Nested Elements for assetcriteria .................................................... 10
2.4. Parameters for searchterm ............................................................... 11
2.5. Valid searchterms ............................................................................. 12
2.6. Valid searchterms (continued) .......................................................... 13
2.7. Valid searchterm operations .............................................................. 14
2.8. Parameters for assetcreate ............................................................... 15
2.9. Nested Elements for assetcreate ...................................................... 16
2.10. Contact Attributes .......................................................................... 18
2.11. Contact Attributes (cont.) .............................................................. 19
2.12. Attributes of files element .............................................................. 21
2.13. assertionresult attributes ............................................................... 23
2.14. Attributes ....................................................................................... 25
2.15. Parameters for assetupdate ............................................................. 27
2.16. Nested Elements for assetupdate .................................................... 27
2.17. Attributes ....................................................................................... 30
2.18. Nested elements for delete ............................................................. 31
2.19. Attributes of files element .............................................................. 32
2.20. assertionresult attributes ............................................................... 34
2.21. Attributes ....................................................................................... 36
List of Examples

1.1. Task Definition ................................................................. 8
1.2. registry-tasks.properties (contents) ........................................ 8
1.3. Opening 'project' tag referencing harvesting antlib ...................... 9
2.1. Simple Query ...................................................................... 14
2.2. Query with 'op' attribute .................................................... 14
2.3. Usage of appliedComplianceTemplateProjects ............................ 17
2.4. Usage of appliedPolicies ..................................................... 17
2.5. Usage of assettype ............................................................... 17
2.6. Usage of categorizations ...................................................... 18
2.7. Usage of contacts ............................................................... 19
2.8. Usage of customAccessSettings ............................................ 20
2.9. Usage of customData .......................................................... 20
2.10. Usage of description ......................................................... 21
2.11. Usage of files .................................................................... 22
2.12. Usage of keywords ........................................................... 22
2.13. Usage of notificationEmail ............................................... 23
2.14. Usage of policyAssertionResults ......................................... 23
2.15. Usage of producingProjects ................................................ 24
2.16. Usage of registrationstatus ................................................. 24
2.17. Usage of relationships ........................................................ 24
2.18. Usage of sfids ................................................................. 25
2.19. Usage of status ............................................................... 25
2.20. Usage of uniqueelement ..................................................... 26
2.21. Usage of vendor ............................................................... 26
2.22. Usage of appliedComplianceTemplateProjects .......................... 28
2.23. Usage of appliedPolicies .................................................... 28
2.24. Usage of assettype ............................................................ 29
2.25. Usage of categorizations .................................................... 29
2.26. Usage of contacts ............................................................ 30
2.27. Usage of customAccessSettings .......................................... 31
2.28. Usage of customData ........................................................ 31
2.29. Usage of delete ............................................................. 31
2.30. Usage of description ......................................................... 32
2.31. Usage of files ............................................................... 33
2.32. Usage of keywords .......................................................... 33
2.33. Usage of notificationEmail ................................................. 33
2.34. Usage of policyAssertionResults ......................................... 34
2.35. Usage of producingProjects ............................................... 34
2.36. Usage of registrationstatus ................................................ 35
2.37. Usage of relationships ...................................................... 35
2.38. Usage of sfids ............................................................... 35
2.39. Usage of status ............................................................. 36
2.40. Usage of uniqueelement .................................................... 36
2.41. Usage of uniqueelement (deleting) ........................................ 37
2.42. Usage of vendor ............................................................. 37
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**

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

Where the registry-tasks.properties contains the following:

**Example 1.2. registry-tasks.properties (contents)**

```properties
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

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

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

...</registry:assetcreate>...</target>
```
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>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryUsername</td>
<td>A valid ALER username.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryPassword</td>
<td>ALER password.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryURL</td>
<td>ALER URL</td>
<td>Yes</td>
</tr>
<tr>
<td>configuration</td>
<td>Controls whether the failure to</td>
<td>No (can be either 'warn' or 'fail', and defaults to 'warn')</td>
</tr>
<tr>
<td></td>
<td>match any asset causes an error, or generates a warning.</td>
<td></td>
</tr>
<tr>
<td>ret_assetids</td>
<td>Property which will receive query</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>results.</td>
<td></td>
</tr>
</tbody>
</table>

Nested Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>AssetCriteria</td>
<td>Query criteria.</td>
<td>Yes</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>searchterm</td>
<td>One a of set of nested query search terms.</td>
<td>Yes - at least one</td>
</tr>
</tbody>
</table>
The `searchterm` Element

Table 2.4. Parameters for `searchterm`

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Search term name</td>
<td>Yes</td>
</tr>
<tr>
<td>value</td>
<td>Search term value</td>
<td>Yes</td>
</tr>
<tr>
<td>op</td>
<td>Search term match operation</td>
<td>No. Defaults to 'eq'. See the Valid search term operations table for possible values.</td>
</tr>
</tbody>
</table>
### Table 2.5. Valid searchterms

<table>
<thead>
<tr>
<th>Name (value of 'name' attribute in 'searchterm' element)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Search for the specified ID.</td>
</tr>
<tr>
<td>name</td>
<td>Search for the specified name.</td>
</tr>
<tr>
<td>/asset/mandatory-data/name</td>
<td>Search for the specified name.</td>
</tr>
<tr>
<td>version</td>
<td>Search for the specified version string.</td>
</tr>
<tr>
<td>/asset/mandatory-data/version</td>
<td>Search for the specified version string.</td>
</tr>
<tr>
<td>assettypeid</td>
<td>Search for Assets of an AssetType with the specified ID.</td>
</tr>
<tr>
<td>archetypeid</td>
<td>Search for Assets of an AssetType with Archetype with specified ID.</td>
</tr>
<tr>
<td>description</td>
<td>Search for the specified description.</td>
</tr>
<tr>
<td>/asset/mandatory-data/description</td>
<td>Search for the specified description.</td>
</tr>
<tr>
<td>keyword</td>
<td>Search for the specified keyword.</td>
</tr>
<tr>
<td>/asset/mandatory-data/keywords/keyword</td>
<td>Search for the specified keyword.</td>
</tr>
<tr>
<td>hashinfo</td>
<td>Search for the specified SFID.</td>
</tr>
<tr>
<td>/asset/custom-data/hashInfos/hashInfo</td>
<td>Search for the specified SFID.</td>
</tr>
<tr>
<td>project</td>
<td>Search for assets that have a producing project with the specified ID.</td>
</tr>
<tr>
<td>/asset/mandatory-data/producing-projects/project</td>
<td>Search for assets that have a producing project with the specified ID.</td>
</tr>
<tr>
<td>general</td>
<td>Search for a string that must be present in name, version, description, keywords, categorizations or indexed fields.</td>
</tr>
<tr>
<td>exactMatch</td>
<td>Set whether to return only exact matches.</td>
</tr>
<tr>
<td>browseableOnly</td>
<td>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.</td>
</tr>
<tr>
<td>offset</td>
<td>the starting index (zero-based) from the matching results</td>
</tr>
<tr>
<td>limit</td>
<td>the number of results ot return, starting at the starting index specified by &quot;offset&quot;</td>
</tr>
</tbody>
</table>
Table 2.6. Valid searchterms (continued)

<table>
<thead>
<tr>
<th>Name (value of 'name' attribute in 'searchterm' element)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>categorization.#</td>
<td>Search for Assets with CategorizationType ID=#, and an assigned Categorization equal to the value of the SearchTerm.</td>
</tr>
<tr>
<td>custom-data/*</td>
<td>Search for asset with an arbitrary element in customdata equal to the value specified in the SearchTerm.</td>
</tr>
<tr>
<td>/asset/custom-data/*</td>
<td>Search for asset with an arbitrary element in customdata equal to the value specified in the SearchTerm.</td>
</tr>
<tr>
<td>status</td>
<td>Search on the active status of the Asset. Value should be one of 0=Active, 10=Inactive, 20=Retired, 30=Deleted, 40=Incomplete.</td>
</tr>
<tr>
<td>uniqueelement</td>
<td>Search for an asset whose unique element is equal to the SearchTerm's value</td>
</tr>
<tr>
<td>underconstruction</td>
<td>Search for assets that have a status of UNDERCONSTRUCTION</td>
</tr>
<tr>
<td>submission</td>
<td>Search for assets that have a status of SUBMITTED.</td>
</tr>
<tr>
<td>rejected</td>
<td>Search for assets that have a status of REJECTED.</td>
</tr>
<tr>
<td>queued</td>
<td>Search for assets that have a status of QUEUED.</td>
</tr>
<tr>
<td>registered</td>
<td>Search for assets that have a status of REGISTERED.</td>
</tr>
<tr>
<td>createdbyid</td>
<td>Search for assets that were created by the specified user ID.</td>
</tr>
<tr>
<td>submittedbyid</td>
<td>Search for assets that were submitted by the specified ID.</td>
</tr>
<tr>
<td>assetstatus</td>
<td>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.</td>
</tr>
</tbody>
</table>
Table 2.7. Valid searchterm operations

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

Examples

Example 2.1. Simple Query

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryUsername</td>
<td>A valid ALER username.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryPassword</td>
<td>A valid ALER password.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryURL</td>
<td>Registry URL</td>
<td>Yes</td>
</tr>
<tr>
<td>configuration</td>
<td>warn/fail configuration: whether to fail, or just print a warning, if an error is encountered</td>
<td>No (defaults to 'fail')</td>
</tr>
<tr>
<td>name</td>
<td>Asset name</td>
<td>Yes</td>
</tr>
<tr>
<td>version</td>
<td>Asset version</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## Nested elements

### Table 2.9. Nested Elements for assetcreate

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>appliedComplianceTemplateProjects</td>
<td>Applied Compliance Template Projects</td>
<td>No</td>
</tr>
<tr>
<td>appliedPolicies</td>
<td>Applied Policies</td>
<td>No</td>
</tr>
<tr>
<td>assettype</td>
<td>The asset type to assign to the newly created asset.</td>
<td>Yes</td>
</tr>
<tr>
<td>categorizations</td>
<td>Asset categorizations.</td>
<td>No</td>
</tr>
<tr>
<td>contacts</td>
<td>Contacts</td>
<td>No</td>
</tr>
<tr>
<td>customAccessSettings</td>
<td>CAS</td>
<td>No</td>
</tr>
<tr>
<td>customData</td>
<td>Custom data</td>
<td>No</td>
</tr>
<tr>
<td>description</td>
<td>Description of asset.</td>
<td>No</td>
</tr>
<tr>
<td>files</td>
<td>Files</td>
<td>No</td>
</tr>
<tr>
<td>keywords</td>
<td>Keywords</td>
<td>No</td>
</tr>
<tr>
<td>notificationEmail</td>
<td>Notification email address</td>
<td>No</td>
</tr>
<tr>
<td>policyAssertionResults</td>
<td>Policy Assertion Results</td>
<td>No</td>
</tr>
<tr>
<td>producingProjects</td>
<td>Producing projects</td>
<td>No</td>
</tr>
<tr>
<td>relationships</td>
<td>Relationships</td>
<td>No</td>
</tr>
<tr>
<td>registrationStatus</td>
<td>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.</td>
<td>No</td>
</tr>
<tr>
<td>sfids</td>
<td>HashInfos (SFIDs)</td>
<td>No</td>
</tr>
<tr>
<td>status</td>
<td>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.</td>
<td>No</td>
</tr>
<tr>
<td>uniqueElement</td>
<td>UniqueElement</td>
<td>No</td>
</tr>
<tr>
<td>vendor</td>
<td>Vendor</td>
<td>No</td>
</tr>
</tbody>
</table>

### 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

```xml
<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

```xml
<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

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name string for the contact</td>
<td>no</td>
</tr>
<tr>
<td>id</td>
<td>Id of the contact. Contacts may be specified for deletion using this attribute.</td>
<td>no</td>
</tr>
<tr>
<td>type</td>
<td>type of contact</td>
<td>no</td>
</tr>
</tbody>
</table>


Table 2.11. Contact Attributes (cont.)

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>email</td>
<td>contact email address</td>
<td>no</td>
</tr>
<tr>
<td>phonenumber</td>
<td>contact phone number</td>
<td>no</td>
</tr>
<tr>
<td>company</td>
<td>company name for the contact</td>
<td>no</td>
</tr>
<tr>
<td>street1</td>
<td>first line of the street address for the contact</td>
<td>no</td>
</tr>
<tr>
<td>street2</td>
<td>second line of the street address for the contact</td>
<td>no</td>
</tr>
<tr>
<td>city</td>
<td>city for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>state</td>
<td>state for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>zip</td>
<td>zip-code for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>country</td>
<td>country for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>faxnumber</td>
<td>the contact fax number</td>
<td>no</td>
</tr>
</tbody>
</table>

Example 2.7. Usage of contacts

```xml
<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

```xml
<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

```xml
<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

```xml
<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

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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>assertionid</td>
<td>The ID of the policy assertion.</td>
<td>Yes</td>
</tr>
<tr>
<td>date</td>
<td>The date of the result. The date must be of the format 'yyyy-mm-dd'.</td>
<td>Yes</td>
</tr>
<tr>
<td>value</td>
<td>If supplied, must be one of 'pass', 'fail' or &quot; (the empty string).</td>
<td>No</td>
</tr>
</tbody>
</table>

Example 2.14. Usage of policyAssertionResults

```xml
<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

```xml
<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

```xml
<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

```xml
<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

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Long representing an ID</td>
<td>no: one of id or value must be specified though</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>no: one of id or value must be specified though</td>
</tr>
</tbody>
</table>
Example 2.20. Usage of uniquenelement

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryUsername</td>
<td>A valid ALER username.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryPassword</td>
<td>ALER password.</td>
<td>Yes</td>
</tr>
<tr>
<td>RegistryURL</td>
<td>ALER URL</td>
<td>Yes</td>
</tr>
<tr>
<td>configuration</td>
<td>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.</td>
<td>No (default is 'fail')</td>
</tr>
<tr>
<td>update_assetids</td>
<td>A comma-delimited string containing the IDs of the assets to update.</td>
<td>No, but required if a nested assetcriteria element is not provided.</td>
</tr>
</tbody>
</table>

Nested Elements

Table 2.16. Nested Elements for assetupdate

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>assetcriteria</td>
<td>An assetcriteria element: assets which match these criteria will be updated.</td>
<td>Yes, unless the update_assetids parameter is supplied.</td>
</tr>
<tr>
<td>assettype</td>
<td>The asset type to assign to the updated asset.</td>
<td>No</td>
</tr>
<tr>
<td>appliedPolicies</td>
<td>Policies applied to this asset.</td>
<td>No</td>
</tr>
<tr>
<td>categorizations</td>
<td>Asset categorizations</td>
<td>No</td>
</tr>
<tr>
<td>customData</td>
<td>Custom Data</td>
<td>No</td>
</tr>
<tr>
<td>description</td>
<td>Description</td>
<td>No</td>
</tr>
<tr>
<td>contacts</td>
<td>contacts</td>
<td>No</td>
</tr>
<tr>
<td>files</td>
<td>Files</td>
<td>No</td>
</tr>
<tr>
<td>keywords</td>
<td>Keywords</td>
<td>No</td>
</tr>
<tr>
<td>notificationEmail</td>
<td>Notification Email</td>
<td>No</td>
</tr>
<tr>
<td>producingProjects</td>
<td>Producing Projects</td>
<td>No</td>
</tr>
<tr>
<td>relationships</td>
<td>Relationships</td>
<td>No</td>
</tr>
<tr>
<td>status</td>
<td>Status</td>
<td>No</td>
</tr>
<tr>
<td>delete</td>
<td>Tag which allows deletion of single-valued elements (description, notificationEmail, uniqueelement).</td>
<td>No</td>
</tr>
</tbody>
</table>
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. Assertion results are matched for deletion by 'id' attribute.

**Example 2.22. Usage of `appliedComplianceTemplateProjects`**

```xml
<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`**

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name string for the contact</td>
<td>no</td>
</tr>
<tr>
<td>id</td>
<td>ID for the contact</td>
<td>no</td>
</tr>
<tr>
<td>type</td>
<td>type of contact</td>
<td>no</td>
</tr>
<tr>
<td>email</td>
<td>contact email address</td>
<td>no</td>
</tr>
<tr>
<td>phonenumber</td>
<td>contact phone number</td>
<td>no</td>
</tr>
<tr>
<td>company</td>
<td>company name for the contact</td>
<td>no</td>
</tr>
<tr>
<td>street1</td>
<td>first line of the street address</td>
<td>no</td>
</tr>
<tr>
<td>street2</td>
<td>second line of the street address</td>
<td>no</td>
</tr>
<tr>
<td>city</td>
<td>city for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>state</td>
<td>state for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>zip</td>
<td>zip-code for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>country</td>
<td>country for the address of the contact</td>
<td>no</td>
</tr>
<tr>
<td>faxnumber</td>
<td>the contact fax number</td>
<td>no</td>
</tr>
</tbody>
</table>

### Example 2.26. Usage of contacts

```xml
<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

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Tag name</th>
</tr>
</thead>
<tbody>
<tr>
<td>notificationEmail</td>
</tr>
<tr>
<td>uniqueElement</td>
</tr>
</tbody>
</table>

Example 2.29. Usage of delete

```xml
<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

```xml
<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

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>assertionid</td>
<td>The ID of the policy assertion.</td>
<td>Yes</td>
</tr>
<tr>
<td>date</td>
<td>The date of the result. The date must be of the format 'yyyy-mm-dd'.</td>
<td>Yes</td>
</tr>
<tr>
<td>value</td>
<td>If supplied, must be one of 'pass', 'fail' or &quot; (the empty string).</td>
<td>No</td>
</tr>
</tbody>
</table>

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**

```xml
<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**

```xml
<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**

```xml
<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

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>Long representing an ID</td>
<td>no: one of id or value must be specified though</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>no: one of id or value must be specified though</td>
</tr>
</tbody>
</table>

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

```xml
<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)

```xml
<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>
```

type
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

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