Opsware® SAS DCML Exchange Tool (DET) 2.5 Reference Guide

Corporate Headquarters

599 North Mathilda Avenue Sunnyvale, California 94085 U.S.A.
T + 1 408.744.7300 F +1 408.744.7383 www.opsware.com

Table of Contents

Preface

Welcome to the Opsware DCML Exchange Tool (DET). This tool enables you to export almost all server management content from any Opsware mesh – standalone or multimaster mesh – and import it into any other Opsware mesh. Opsware can also provide pre-packaged server management content appropriate for new installations that can be imported into a mesh after initial setup. See your Opsware Representative for information on obtaining this content.

Content that can be exported includes:

- Applications
- Application Configurations
- Application Configuration Templates
- Custom Extensions
- Custom Fields Schema
- Customers
- Distributed Scripts
- OSs
- Packages
- Patches
- Patch Policies
- Server Compliance Criteria
- Server Groups
- Service Levels
- Templates
- User Groups

In addition, information associated with the content is also exported, including:

- MRLs
- Install Hooks
- Units
- Install Order Relationships
- Custom Attributes
- Software Lists
- Configuration Tracking Policies
- Unit Scripts
- Patch Exceptions

About This Guide

This guide describes how to use the DCML Exchange Tool and provides information about how to export and import content, how to create export filters, how to specify handling of duplicates encountered during the import process, and complete information about command line options and arguments.

This guide is intended for system administrators who are responsible for specifying the content used on Opsware-managed servers.

Contents of this Guide

This guide contains the following chapters and appendices:

Chapter 1: Installing and Configuring the DCML Exchange Tool – provides information about the installation and configuration of the DCML Exchange Tool, including what files to download and where to find them, how to set the environment variable, how to obtain needed certificates, how to create a configuration file and what properties to use, and example configuration files.

Chapter 2: Using the DCML Exchange Tool – provides information about how to export content, what kind of content can be exported, commands and arguments for exports, how to create filters for exporting content and example filters, how to import content, policies for importing each content type, information about how each content type handles importing duplicates, and a description of the content directories.

Appendix A: DCML Exchange Tool Command Line – provides information about command line commands and their associated arguments.

Appendix B: Import Delete Conditions – describes all conditions that must be met for a content item to be marked as deleted, and those cases in which the content item will be renamed upon import.

Conventions in this Guide

This guide uses the following typographical and formatting conventions.

NOTATION	DESCRIPTION
`Courier`	Identifies text of displayed messages and other output from Opsware programs or tools.
`Courier Bold`	Identifies user-entered text (commands or information).
`Courier Italics`	Identifies variable user-entered text on the command line or within example files.

Icons in this Guide

This guide uses the following iconographic conventions.

ICON	DESCRIPTION
📄	This icon is a note. It identifies especially important concepts that warrant added emphasis.
☑	This icon is a requirement. It identifies a task that must be performed before an action under discussion can be performed.
💡	This icon is a tip. It identifies information that can help simplify or clarify tasks.
⚠	This icon is a warning. It is used to identify significant information that must be read before proceeding.

Contacting Opsware, Inc.

The main web site and phone number for Opsware, Inc. are as follows:

- http://www.opsware.com/index.htm
- +1 (408) 744-7300

For links to the latest product documentation and software downloads, see the Opsware Customer Support site:

- https://download.opsware.com/opsw/main.htm

For troubleshooting information, you can search the Opsware Knowledge Base at:

- https://download.opsware.com/kb/kbindex.jspa

The Opsware Customer Support email address and phone number follow:

- support@opsware.com/

+1 (877) 677-9273

Chapter 1: Installing and Configuring DET

IN THIS CHAPTER

This chapter contains the following topic:

Installing and Configuring DCML Exchange Tool (DET)

The following instructions detail how to install and configure the DCML Exchange Tool (DET).

The DCML Exchange Tool (DET) can be run on any UNIX computer, though not necessarily a managed server. (Although DET is not supported on the Windows platform, it does support import and export of Windows content.)

1. Log on to an Opsware-managed server as the root user.
2. If you do not already have them, download JRE 1.4.x or JDK 1.4.x from www.sun.com, and install the program on the server where you have logged in.
3. Download the cbt-<version>.zip file from download.opsware.com and unzip the distribution on the server where you have logged in. You will need a login ID and password for the download site; ask your Opsware administrator if you do not have a login ID and password.
4. Set your JAVA_HOME environment variable to point to a Java 1.4.x installation. For example, in csh you would issue the following command:
   ```
   % setenv JAVA_HOME <j2re 1.4.x installation>
   ```
5. Optionally, you can set the PATH environment variable to include the DET install directory: `<cbt-install_dir>/bin`.

6 Perform the following steps for each mesh that DET will be importing into or exporting from.

1. Obtain a copy of the opsware-ca.crt trust certificate from
   `/var/lc/crypto/twist/opsware-ca.crt`
   and save it in a location DET can access. This step is optional if you are running DET from the server where the Opsware Command Center is installed.
2. Obtain a copy of the spog.pkcs8 client certificate from
   `/var/lc/crypto/twist/spog.pkcs8`
   and save it in a location DET can access. This step is optional if you are running DET from the server where the Opsware Command Center is installed.
3. Obtain the twist username and password - this is set during the twist install and the Opsware administrator should have this information.
4. Create a target mesh configuration file that contains the location and identity information required to access the Opsware mesh components.

Creating a Target Mesh Configuration File

Create a target mesh configuration file to simplify the use of DET. A sample default configuration file is installed with DET at the following location:

```
cbt/cfg/default.properties
```

The mesh configuration file is a key=value pair text file that contains Opsware component access information that would otherwise need to be given on the DET command-line. To define the parameters of the DET mesh configuration file, make a copy of this file and save it to a known location.

Table Table 1-1 contains all possible DET configuration-related properties. These properties can be either given on the DET command-line or specified in a configuration file.

The default configuration property values listed in Table 1-1 assume that you are running DET on an Opsware mesh running the Opsware Command Center. (It is for this reason that the .host properties shows a `localhost` value.) Also, `twist.certpaths`, `ssl.trustcerts`, and `ssl.keypairs` assume paths on an Opsware Command Center server.

If a configuration-related property is not specifically mentioned in the mesh Configuration file, the default value shown in the Configuration Properties table below will be used.

Table 1-1: Configuration Properties

PROPERTY NAME	DEFAULT VALUE	DESCRIPTION
`cbt.numthreads`	`1`	Number of concurrent threads used for export. For exporting content, you can specify as many threads as you wish. However, for importing content, DET 5.3 supports only one thread.
`spike.enabled`	`true`	Use Spike for authentication and authorization on all XML-RPC-based servers.
`spike.host`	`way`	Spike's host name or IP.
`spike.path`	`wayrpc.py`	Spike's base URL path.
`spike.password`	`<no default>`	User password for Spike authentication. This is an OCC user's password and is set during the installation of the mesh. Contact your Opsware Administrator (or the person who installed the mesh) for this information.
`spike.port`	`1018`	Spike's listener port.
`spike.protocol`	`https`	Spike's listener protocol. This is typically HTTPS.

Table 1-1: Configuration Properties

PROPERTY NAME	DEFAULT VALUE	DESCRIPTION
`spike.username`	`admin`	User name for Spike authentication. This is the user who was granted permissions by the cbtperm tool. This username needs to be needs to be an admin account that has permissions to create or modify objects; however, it cannot be the `detuser` account. The `det` default configuration sets `spike.username` to account: admin
`spin.host`	`spin`	Data Access Engine's host name or IP.
`spin.path`	`spinrpc.py`	Data Access Engine's base URL path.
`spin.port`	`1004`	Data Access Engine's listener port.
`spin.protocol`	`http`	Data Access Engine's listener protocol. HTTP if the DET is on the same server as the Opsware Command Center and is running a cleartext spin in a multi-server mesh or HTTPS for any other configuration.
`ssl.keyPairs`	`/var/lc/crypto/twist/spog.pkcs8`	Comma-separated list of client certificates used to communicate with XML-RPC-based servers.

Table 1-1: Configuration Properties

PROPERTY NAME	DEFAULT VALUE	DESCRIPTION
`ssl.trustCerts`	`/var/lc/crypto/twist/opsware-ca.crt`	Comma-separated list of trust certificate files used to communicate with XML-RPC-based servers.
`ssl.useHttpClient`	`true`	Use the HTTPClient library instead of JDK's built-in HTTP client.
`twist.certPaths`	`/var/lc/crypto/twist/opsware-ca.crt`	Comma-separated list of trust certificates used to communicate with the Web Services Data Access Engine.
`twist.host`	`localhost`	Web Services Data Access Engine's host name or IP.
`twist.password`	`<no default>`	Web Services Data Access Engine's password. This password is set during the installation of the mesh. Contact your Opsware Administrator (or the person who installed the mesh) for this information.
`twist.port`	`1032`	Web Services Data Access Engine's listening port.
`twist.protocol`	`t3s`	Web Services Data Access Engine's protocol. This should be t3 or t3s.

Table 1-1: Configuration Properties

PROPERTY NAME	DEFAULT VALUE	DESCRIPTION
`twist.username`	`detuser`	Web Services Data Access Engine's username. This needs to be "`detuser`". This account is a system account, and the password is set during install of the mesh.
`way.host`	`way`	Command Engine's host name or IP.
`way.path`	`wayrpc.py`	Command Engine's base URL path.
`way.port`	`1018`	Command Engine's listener port.
`way.protocol`	`https`	Command Engine's listener protocol. This is typically HTTPS.
`word.host`	`word`	Software Repository's host name or IP.
`word.path`	`wordrpc.py`	Software Repository's base URL path.
`word.port`	`1003`	Software Repository's listener port.
`word.protocol`	`https`	Software Repository's listener protocol. This is HTTPS.
`mail.transport.protocol`	`smtp`	Mail transport protocol used for your mail server.
`mail.smtp.host`	`smtp`	Mail server hostname.
`mail.smtp.port`	`25`	Port number used by your mail server.
`mail.from`	`<currentuser>@<currenthost>`	Email address to use for the From field in the notification email.

The following is an example of a target mesh configuration file that contains only essential mesh configuration information.

```
twist.host=twist.c07.dev.opsware.com
twist.port=1032
twist.protocol=t3s
twist.username=<detuser>
twist.password=<twist_password>
twist.certPaths=<absolute path to opsware-ca.crt>

spike.username=<OCC_user>
spike.password=<OCC_user_password>
spike.host=way.c07.dev.opsware.com
way.host=way.c07.dev.opsware.com
spin.host=spin.c07.dev.opsware.com
word.host=theword.c07.dev.opsware.com

ssl.keyPairs=<absolute path to spog.pkcs8>
ssl.trustCerts=<absolute path to opsware-ca.crt>

mail.transport.protocol=smtp
mail.smtp.host=mail
mail.smtp.port=44
mail.from=joe_user@yourcompany.com
```

Distribution Directory

The following list shows what an expanded `cbt-<version>.zip` file contains.

```
% ls -R cbt

cbt:
   bin/
   cfg/
   filters/
   lib/

cbt/bin:
   cbt*
   cbtperm*
   rdql*

cbt/cfg:
   core.owl
   default.properties
   filter.owl
   java.policy
```

```
    license.bea
    logging.bootstrap
    logging.template
    mail.properties
    opsware.owl
    version.txt

cbt/filters:

    all.rdf
    appconfigfile.rdf
    appconfig.rdf
    app.rdf
    compliancecriteria.rdf
    custext.rdf
    customer.rdf
    customfield.rdf
    distscript.rdf
    os.rdf
    package.rdf
    patchpolicy.rdf
    patch.rdf
    servergroup.rdf
    servicelevel.rdf
    template.rdf
    usergroup.rdf

cbt/lib:

    activation.jar
    antlr.jar
    bea-license.jar
    cbt.jar
    certicom-jdk14-wl700-patch.jar
    common-1.2.0.jar
    commons-lang-2.0.jar
    commons-logging.jar
    concurrent.jar
    copyright.txt
    CR186100_700sp5.jar
    ejb-2.0.jar
    HTTPClient-hacked.jar
    icu4j.jar
    jakarta-oro-2.0.5.jar
    jena_0604.jar
    junit.jar
    LICENSE-jaf.txt
    LICENSE-javamail.txt
    mail.jar
```

```
NOTES-javamail.txt
opsware_common-1.0.5.jar
rdf-api-2001-01-19.jar
spinclient-14b.0.0.108.jar
twistclient-latest.jar
weblogic.jar
xercesImpl.jar
xml-apis.jar
```

Chapter 2: Using DET

I N T H I S C H A P T E R

This chapter includes the following topics:

Introduction to the DCML Exchange Tool (DET)

The DCML Exchange Utility (DET) allows you to import and export Opsware content. The primary function of this tool is to provide a way to inject a newly-installed Opsware mesh with content from an existing mesh. This tool can also be used to export partial content from one mesh and import it into other mesh instances.

In the context of DET, "content" means user-created server management information in Opsware. This includes the following content types: Applications, Application Configurations, Application Configuration Templates, Custom Extensions, Custom Fields Schema, Customers, Distributed Scripts, SOs, Packages, Patches, Patch Policies, Server Compliance Criteria, Server Groups, Service Levels, Templates, and User Groups.

Associated content information includes MRLs, Install Hooks, Configuration Tracking Policies, Custom Attributes, Custom Field Schema.

Content does not include managed environment type information. For example, facility information and server properties are not included. Also, CD&R is not included in this release of DET.

DET is a command-line utility that can be run on any Unix host with network connectivity to a target Opsware mesh. DET is written in Java and uses OWL and RDF for its schema definition and persistent store. DET imports and exports Opsware content by using Opsware component API's to extract both configuration and large binary content, such as packages and scripts.

DET Relationship to DCML

The content exported by the DET is in compliance with DCML Framework Specification v0.11, the first publicly-available specification of DCML. The DCML Exchange Tool uses a proprietary extension schema to describe contents exported from Opsware. The exported data.rdf is a valid DCML instance document that is parsable by a compliant DCML processor.

Exporting Content

DET exports the content you specify from a target Opsware mesh to an RDF/XML file that can be imported by DET into another Opsware mesh.

The export command is:

```
cbt -e <content_dir> -f <filter_file> -cf <target_core_config>
```

The command and its arguments indicate:

- `content_dir` - the path to a directory where the exported content will be stored. This directory will be created by the export function if it does not already exist.
- `filter_file` - a set of rules that tells DET what content it should export from the target Opsware mesh. See the "Export Filters" on page 25 for information on creating this file.
- `target_core_config` - a configuration file that tells DET where the various Opsware components are located, and what identity it should use to access them. Instructions for creating this file are found at "Creating a Target Mesh Configuration File" on page 14.

The export command can be run multiple times using the same arguments, with the following caveats:

- If a filter has been specified, DET will ignore any previous exports in the content directory and will restart the export process.

- If the export command specifies a content directory that contains a valid export (one which previously succeeded), DET will prompt the user if it is OK to overwrite. If the user says it is not OK to overwrite, then DET will exit.

Before beginning an export or import process in a standalone mesh, shut down the Opsware Command Center to prevent users from changing any Opsware content until the process has completed.

In a multimaster mesh, first use the multimaster tools to ensure that the mesh is caught up and there are no conflicts, then shut down all Opsware Command Centers in the mesh to prevent users from changing any Opsware content until the process has completed.

See the Opsware SAS Administration Guide for information about stopping and restarting the Opsware Command Center.

Export Filters

An export filter is a user-specified rule that tells DET what content to export – content that will subsequently be imported. Export filters are used with the following content types:

- Application Export Filter
- Application Configuration Export Filter
- Application Configuration Template Export Filter
- Custom Extension Export Filter
- Custom Fields Schema Export Filter
- Customer Export Filter
- Distributed Script Export Filter
- OS Export Filter
- Package Export Filter
- Patch Export Filter
- Patch Policy Export Filter

- Server Compliance Criteria Export Filter
- Server Group Export Filter
- Service Level Export Filter
- Template Export Filter
- User Group Export Filter

Example Export Filter File

DET reads export filters in a specified filter file. The filter file is encoded in RDF/XML. The following is an example of a simple filter file that contains a single export filter rule.

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE rdf:RDF [
3.  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4.  ]>
5.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#”
6.           xlmns=”http://www.opsware.com/ns/cbt/0.1/filter#">
7.  <ApplicationFilter rdf:ID="exportAppServers">
8.    <path>/Application Servers/Package Test</path>
9.    <directive rdf:resource="&filter;Descendants" />
10. </ApplicationFilter>
11. </rdf:RDF>
```

This example shows the standard filter headers in lines 1 through 6. These lines are the same in every filter, as is Line 11, which is the standard filter footer.

Lines 7 through 10 are the lines that are unique in each filter and indicate the specific function of the filter.

In the example above, there is just one export filter rule. However, filters can contain any number of unique filters between the standard header and footer lines. For example, this filter contains three export filter rules:

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <!DOCTYPE rdf:RDF [
3.  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
4.  ]>
5.  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#”
6.           xlmns=”http://www.opsware.com/ns/cbt/0.1/filter#">
7.  <ApplicationFilter rdf:ID="exportAppServers">
8.    <path>/Application Servers/Package Test</path>
9.    <directive rdf:resource="&filter;Descendants" />
10. </ApplicationFilter>
11. <ApplicationFilter rdf:ID="exportAppServfoo">
12.   <path>/Application Servers/Foo</path>
```

```
  <directive rdf:resource="&filter;Node"/>
</ApplicationFilter>
<CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
  <scriptName>Bulk_Password_Changes</scriptName>
</CustomExtensionFilter>
</rdf:RDF>
```

Example filters can be found in the DET install directory under:

```
<install_dir>/filters
```

This directory includes examples for each filter type and also an `all.rdf` filter, that exports all known Opsware data types from an Opsware mesh.

The following sections describe each filter type and their allowed parameters. In general, filter types map to an object type that can be manipulated by a user of the Opsware Command Center. The Patch Filter, for example, maps to the Opsware Command Center Patch object. Naming of the filters and their attributes also maps to the naming structure of the Opsware Command Center so filter authors can quickly acquaint themselves with filters and their relevance to Opsware content.

Application Export Filter

An application export filter tells DET what application nodes and associated content to export. The following application nodes are shown in the Opsware Command Center by clicking the Software link in the navigation panel followed by the Application link on the Software menu.

- Application Servers
- Database Servers
- OS Extras
- Other Applications
- System Utilities
- Web Servers

The following tables describe the syntax of the Application Filter element:

Table 2-1: Application Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=”unique name”.`

Table 2-2: Application Export Filter Nested Elements

ELEMENT	DESCRIPTION
`path (required)`	An absolute path from the top level of the software tree to the node to be exported. The path separator is “/”.

Table 2-2: Application Export Filter Nested Elements (continued)

ELEMENT	DESCRIPTION

`directive (required)`	An empty content element with a single `rdf:resource` parameter. The parameter refers to one of three constants:
	• `Descendants` – export all descendants of the given path including the leaf of the path.
	• `Node` – only export the given node.
	• `Path` - export all nodes along the path and no other nodes.
	For example, given the following path:
	`/Custom Applications/A/B/C/D`
	and your path is
	`/Custom Applications/A/B`
	If the `rdf:resource` parameter is Node, node B is exported.
	If the `rdf:resource` parameter is Path, nodes A and B are exported.
	If the `rdf:resource` parameter is Descendants, nodes B, C, and D are exported.

`customerName (optional)`	This optional element restricts the export to nodes owned by this customer at or below the specified path. If the node specified by the path is not owned by the specified customer, nothing is exported and a warning is logged. For examples of how this element works in a filter file, see “customerName Element Examples” on page 55.

Application Export Filter Examples

Export the /Application Servers/Foo node only.

```
<ApplicationFilter rdf:ID="exportAppServfoo">
  <path>/Application Servers/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</ApplicationFilter>
```

Export Bar and Baz nodes along the given path. (Note that the stack root is not exported.)

```
<ApplicationFilter rdf:ID="exportDBServBarBaz">
  <path>/DBServer/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</ApplicationFilter>
```

Export the patchtool node and all its descendants, including the leaf node.

```
<ApplicationFilter rdf:ID="exportSUpatchtool">
  <path>/System Utilities/patchtool</path>
  <directive rdf:resource="&filter;Descendants"/>
</ApplicationFilter>
```

Export all Apache web servers that belong to the Acme customer:

```
<ApplicationFilter rdf:ID="exportAcmeAppServApache">
  <path>/Application Servers/Web Servers/Apache Web</path>
  <directive rdf:resource="&filter;Descendents"/>
  <customerName>Acme</customerName>
</ApplicationFilter>
```

Application Configuration Export Filter

The Application Configuration export filter tells DET what Application Configurations you want to export. An Application Configuration is a container for one or more Application Configuration Template files. Thus, if you export an Application Configuration, you will also be exporting all template files inside it.

Table 2-3: Application Configuration Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name"`

Table 2-4: Application Configuration Export Filter Nested Elements

ELEMENT	DESCRIPTION
`configurationName (optional)`	An optional element that specifies the name of the Application Configuration. Use this if you want to export specific Application Configurations by name.
`customerName (optional)`	An optional element that specifies to export all Application Configurations that have been associated with the specified customer.
`osPlatform rdf:resource (optional)`	An optional element that specifies to export all Application Configurations that have been associated with the specified OS.

Application Configuration Export Filter Example

Export all Application Configurations.

```
  <ApplicationConfigurationFilter rdf:ID="getAllAppConfigs"/>
```

Export only the Application Configuration named “iPlanet” that is customer independent and that has been associated with the SunOS 5.8 operating system.

```
<ApplicationConfigurationFilter rdf:ID="getSpecificAppConfigs">
  <configurationName>iPlanet</configurationName>
  <customerName>Customer Independent</customerName>
  <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</ApplicationConfigurationFilter>
```

Application Configuration Template Export Filter

The Application Configuration Template export filter tells DET what Application Configuration Template files you want to export.

Table 2-5: Application Configuration Template Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name"`

Table 2-6: Application Configuration Template Export Filter Nested Elements

ELEMENT	DESCRIPTION
`configurationFileName` (optional)	An optional element that specifies the name of the Application Configuration Template. Use this if you want to export specific Application Configuration Templates by name.
`osPlatform rdf:resource` (optional)	An optional element that specifies to export all Application Configurations that have been associated with the specified OS.
`customerName` (optional)	An optional element that specifies to export all Application Configuration Templates that have been associated with the specified customer.

Application Configuration Template Export Filter Examples

Export all Application Configuration Templates.

```
<ApplicationConfigurationFileFilter
rdf:ID="getAllAppConfigTemps"/>
```

Export the specific Application Configuration Template named “iplanet6.1_mimetypes.tpl” that is customer independent and is associated with the Red Hat Enterprise Linux AS 3 X86_64 operating system.

```
<ApplicationConfigurationFileFilter
rdf:ID="getSpecificAppConfigTemp">

  <configurationFileName>iplanet6.1_mimetypes.tpl</
configurationFileName>

  <customerName>Customer Independent</customerName>

  <osPlatform rdf:resource="&filter;Red_Hat_Enterprise_Linux_
AS_3_X86_64"/>

  </ApplicationConfigurationFileFilter>
```

Custom Extension Export Filter

The custom extension export filter tells DET to either export a specific custom extension or all custom extensions. If you want to export more than one custom extension, but not all, create a filter for each custom extension you want to export.

Table 2-7: Custom Extension Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”.`

Table 2-8: Custom Extension Export Filter Nested Elements

ELEMENT	DESCRIPTION
`scriptName (optional)`	An optional element that specifies a script to export. The script name does not include the account prefix. If this element is omitted, all custom extension scripts are exported.

Custom Extension Export Filter Examples

Export the Bulk_Password_Changes custom extension script only.

```
<CustomExtensionFilter rdf:ID="exportCustExtBulkPasswd">
  <scriptName>Bulk_Password_Changes</scriptName>
</CustomExtensionFilter>
```

Export all custom extension scripts.

```
<CustomExtensionFilter rdf:ID="exportAllCustExtScripts"/>
```

Custom Fields Schema Export Filter

The custom fields schema export filter tells DET to export all custom fields definitions from a mesh.

Table 2-9: Custom Field Schema Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name".`

Custom Field Schema Export Filter Example

Export all custom field definitions from a mesh:

```
<CustomFieldSchemaFilter rdf:ID="getCustomFieldsSchema"/>
```

Customer Export Filter

The customer export filter tells DET to export all or specific customers from a mesh.

Table 2-10: Customer Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name".`

Table 2-11: Customer Export Filter Nested Elements

ELEMENT	DESCRIPTION
`customerName (optional)`	An optional element that specifies a unique customer to export.

Customer Export Filter Example

Export the all customers from a mesh:

```
<CustomerFilter rdf:ID="exportAllCustomers"/>
```

Export Customer named “Acme Computers” from a mesh:

```
<CustomerFilter rdf:ID="exportAcmeCustomer">
  <customerName>Acme Computers</customerName>
</CustomerFilter>
```

Distributed Script Export Filter

The distributed export script filter tells DET to either export a specific distributed script or all distributed scripts. Only shared distributed scripts are exported and imported.

Table 2-12: Distributed Script Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”`

Table 2-13: Distributed Script Export Filter Nested Elements

ELEMENT	DESCRIPTION
`scriptName (optional)`	An optional element that specifies a script to export. The script name does not include the __global__ prefix. If this element is omitted, all shared distributed scripts are exported.

Distributed Script Export Filter Examples

Export all shared distributed scripts.

```
<DistributedScriptFilter rdf:ID="exportAllSharedScripts"/>
```

Export the shared ls distributed script only.

```
<DistributedScriptFilter rdf:ID="exportScriptLS">
  <scriptName>ls</scriptName>
</DistributedScriptFilter>
```

OS Export Filter

The Operating System export filter tells DET what Operating System node or Operating System type to export.

Table 2-14: OS Export Filter Parameters

PARAMETERS	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”`.

Table 2-15: OS Export Filter Nested Elements

ELEMENT	DESCRIPTION
`osPlatform (required)`	A required nested element. This empty element has an `rdf:resource` parameter. This parameter may refer to one of the following operating systems: • AIX_4.3 • AIX_5.1 • AIX_5.2 • AIX_5.3 • HP-UX_10.20 • HP-UX_11.00 • HP-UX_11.11 • HP-UX_11.23 • Red_Hat_Enterprise_Linux_AS_2.1 • Red_Hat_Enterprise_Linux_AS_3 • Red_Hat_Enterprise_Linux_AS_3_X86_64 • Red_Hat_Enterprise_Linux_AS_4 • Red_Hat_Enterprise_Linux_AS_4_X86_64 • Red_Hat_Enterprise_Linux_ES_2.1 • Red_Hat_Enterprise_Linux_ES_3 • Red_Hat_Enterprise_Linux_ES_3_X86_64 • Red_Hat_Enterprise_Linux_ES_4 • Red_Hat_Enterprise_Linux_ES_4_X86_64 • Red_Hat_Enterprise_Linux_WS_2.1 • Red_Hat_Enterprise_Linux_WS_3 • Red_Hat_Enterprise_Linux_WS_3_X86_64 • Red_Hat_Enterprise_Linux_WS_4 • Red_Hat_Enterprise_Linux_WS_4_X86_64 • SuSE_Linux_8 • SuSE_Linux_Enterprise_Server_8 • SuSE_Linux_Enterprise_Server_9 • SuSE_Linux_Enterprise_Server_9 X86_64 • Red_Hat_Linux_6.2 • Red_Hat_Linux_7.1 • Red_Hat_Linux_7.2 • Red_Hat_Linux_7.3 • Red_Hat_Linux_8.0 • SunOS_5.6 • SunOS_5.7 • SunOS_5.8 • SunOS_5.9 • SunOS_5.10 • SunOS_5.10 X86 • UNKNOWN • Windows_2000 • Windows_2003 • Windows_NT_4.0

OS Export Filter Examples

Export the “7.1 for mwp” Red Hat Linux 7.1 OS.

```
<OSFilter rdf:ID="exportOSRHLinux71">
  <osPlatform rdf:resource="&filter;Red_Hat_Linux_7.1"/>
  <osName>7.1 for mwp</osName>
</OSFilter>
```

Export all Solaris 5.6 operating systems.

```
<OSFilter rdf:ID="exportOSSun56">
  <osPlatform rdf:resource="&filter;SunOS_5.6"/>
</OSFilter>
```

Package Export Filter

The package export filter tells DET to export all or specified packages from a mesh.

For Microsoft Hotfixes and service packs, it is possible that the Microsoft package you want to export has not yet had its binary file uploaded, even though the package shows as existing in the mesh. For example, a user may have uploaded the Microsoft Patch Database to the mesh, but not yet uploaded the actual binary file of the package In this case, a unit record for the package will have been created in the Opsware model, but there is no content to export. In this case, if you try to export the package content using the Package Export Filter, the content of the Microsoft package will not be exported.

Table 2-16: Package Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”.`

Table 2-17: Package Export Filter Nested Elements

ELEMENT	DESCRIPTION
`packageType (required)`	A required element that specifies the package type you want to export. This parameter may refer to one of the following package types: • AIX_Base_Fileset • AIX_LPP • AIX_Update_Fileset • APAR • Build_Customization_Script • HPUX_Depot • HPUX_Fileset • HPUX_Patch_Fileset • HPUX_Patch_Product • HPUX_Product • RPM • Solaris_Package • Solaris_Package_Instance • Solaris_Patch • Solaris_Patch_Cluster • Unknown • Windows_Hotfix • Windows_MSI • Windows_OS_Service_Pack • Windows_ZIP
`packageName (optional)`	An optional element that allows you to specify a named package. The name of the package is the Name field as it appears in the Package Properties page in the Opsware Command Center, not the filename of the package.

Table 2-17: Package Export Filter Nested Elements (continued)

ELEMENT	DESCRIPTION
`osPlatform (optional)`	An optional element that allows you to specify the operating system of a named package. • AIX_4.3 • AIX_5.1 • AIX_5.2 • AIX_5.3 • HP-UX_10.20 • HP-UX_11.00 • HP-UX_11.11 • Red_Hat_Enterprise_Linux_AS_2.1 • Red_Hat_Enterprise_Linux_AS_3 • Red_Hat_Enterprise_Linux_AS_3_X86_64 • Red_Hat_Enterprise_Linux_AS_4 • Red_Hat_Enterprise_Linux_AS_4_X86_64 • Red_Hat_Enterprise_Linux_ES_2.1 • Red_Hat_Enterprise_Linux_ES_3 • Red_Hat_Enterprise_Linux_ES_3_X86_64 • Red_Hat_Enterprise_Linux_ES_4_X86_64 • Red_Hat_Enterprise_Linux_WS_2.1 • Red_Hat_Enterprise_Linux_WS_3 • Red_Hat_Enterprise_Linux_WS_3_X86_64 • Red_Hat_Enterprise_Linux_WS_4_X86_64 • SuSE_Linux_8 • SuSE_Linux_Enterprise_Server_8 • SuSE_Linux_Enterprise_Server_9 • Red_Hat_Linux_6.2 • Red_Hat_Linux_7.1 • Red_Hat_Linux_7.2 • Red_Hat_Linux_7.3 • Red_Hat_Linux_8.0 • SunOS_5.6 • SunOS_5.7 • SunOS_5.8 • SunOS_5.9 • SunOS_5.10 • Windows_2000 • Windows_2003 • Windows_NT_4.0
`customerName (optional)`	An optional element that allows you to specify the customer of a named package.

Package Export Filter Example

Export all RPM packages for all servers that are customer independent and that run on the SunOS_5.8 operating system:

```
<PackageFilter rdf:ID="exportCIPackages">
    <packageType rdf:resource="&filter;RPM"/>
    <customerName>Customer Independent</customerName>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</PackageFilter>
```

Export the RPM package named ”software1.0.0-1.rpm” for all servers that belong to the Acme Computers customer:

```
<PackageFilter rdf:ID="exportAcmePackages">
    <packageType rdf:resource="&filter;RPM"/>
    <packageName>software1.0.0-1.rpm</packageName>
    <customerName>Acme Computers</customerName>
</PackageFilter>
```

Patch Export Filter

The patch export filter tells DET what patch or patch type to export.

For Windows patch content that was defined previous to DET 2.5, make sure that the Windows MBSA patch definitions are the same for both the source mesh and the destination mesh, or undefined Windows patches will not get imported.

Table 2-18: Patch Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”.`

Table 2-19: Patch Filter Nested Elements

ELEMENT	DESCRIPTION
`patchType (required)`	A required nested element. This empty element has an `rdf:resource` parameter. This parameter may refer to one of the following patch types: • APAR • APAR_FILESET • UPDATE_FILESET • AIX_Update_Fileset • HPUX_PATCH_PRODUCT • HPUX_Patch_Product • HPUX_PATCH_FILESET • HPUX_Patch_Fileset • SOL_PATCH • Solaris_Patch • SOL_PATCH_CLUSTER • Solaris_Patch_Cluster • HOTFIX • Windows_Hotfix • SERVICE_PACK • Windows_OS_Service_Pack • PATCH_META_DATA • Microsoft_Patch_Database
`patchName (optional)`	An optional element that specifies the name of a specific patch. The name must be the patch unit_name, which is the name shown in the Opsware Command Center.

Patch Filter Examples

Export the IY13260 APAR.

```
<PatchFilter rdf:ID="exportAPARIY13260">
  <patchName>IY13260</patchName>
  <patchType rdf:resource="&filter;APAR"/>
</PatchFilter>
```

Export all Solaris patches.

```
<PatchFilter rdf:ID="exportSolPatches">
  <patchType rdf:resource="&filter;SOL_PATCH"/>
</PatchFilter>
```

Patch Policy Export Filter

The patch policy export filter tells DET what user-defined patch policy to export. (Vendor recommended policies will not be exported.)

The optional nested elements <`patchPolicyName`> and <`osPlatform`> can be specified to filter for a specific patch policy. If no optional nested elements are specified, all patch policies in the target mesh are exported

Table 2-20: Patch Policy Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name".`

Table 2-21: Patch Policy Export Filter Nested Elements

ELEMENT	DESCRIPTION
`patchPolicyName`	An optional element that specifies the unique name of the patch policy.

Table 2-21: Patch Policy Export Filter Nested Elements (continued)

ELEMENT	DESCRIPTION
`osPlatform`	An optional element that specifies a specific operating system of the patch policy using an `rdf:resource` parameter. This parameter can refer to one of the following operating systems: • Windows_2000 • Windows_2003 • Windows_NT_4.0

Patch Policy Export Filter Examples

Export all patch policies from the target mesh:

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
```

Export only the patch policies named “BestWindowsPoliciesNT” on the Windows NT operating system, and “BestWindowsPolicies2003” on the Windows 2003 operating system:

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
  <patchPolicyName>BestWindowsPoliciesNT</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_NT/>
</PatchPolicyFilter>

<PatchPolicyFilter rdf:ID="PatchPolicies2"/>
  <patchPolicyName>BestWindowsPolicies2003</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

Export all Patch Policies for the Windows 2003 operating system:

```
<PatchPolicyFilter rdf:ID="PatchPolicies"/>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

Server Compliance Criteria Export Filter

The Server Compliance Criteria export filter tells DET what Server Compliance Criteria you want to export.

Table 2-22: Server Compliance Criteria Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name"`

Table 2-23: Server Compliance Criteria Filter Nested Elements

`selectionCriteriaNam (optional)`	An optional element that specifies the name of the Server Compliance Criteria. Use this if you want to export specific Server Compliance Criteria by name.
`osType rdf:resource (optional)`	An optional element that specifies to export all Server Compliance Criteria that have been associated with the specified OS. The two possible values for this element are either Windows or Unix. For example, `<osType rdf:resource="&filter;Windows"/>` or `<osType rdf:resource="&filter;Unix"/>`

Server Compliance Criteria Export Filter Examples

Export all Server Compliance Criteria.

```
<ComplianceSelectionCriteriaFilter
rdf:ID="getAllSelectionCriteria"/>
```

Export the specific Server Compliance Criteria named “My Selection Criteria” that has been associated with the Windows operating system.

```
<ComplianceSelectionCriteriaFilter
rdf:ID="getSpecificSelectionCriteria">
  <selectionCriteriaName>My Selection Criteria</
selectionCriteriaName>
  <osType rdf:resource="&filter;Windows"/>
</ComplianceSelectionCriteriaFilter>
```

Server Group Export Filter

The server groups export filter tells DET to export specified server groups from a mesh.

Table 2-24: Server Group Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”.`

Table 2-25: Server Group Filter Nested Elements

ELEMENT	DESCRIPTION
`path (required)`	A required element that specifies the name of the server group to export.

Table 2-25: Server Group Filter Nested Elements (continued)

ELEMENT	DESCRIPTION

`directive (required)`	A required empty content element with a single `rdf:resource` parameter. Allows you to specify the contents of the groups to export. The parameter refers to one of three constants:
	• Node: Exports only the leaf node of the path, but create empty placeholders (name and description, no rules) down the path if the path doesn't already exist.
	• Path: Exports all groups along the path (name, description, and rules) but not the descendants.
	• Descendants: Exports all descendants of the given path, including the leaf node of the path.
	For example, given the following path:
	`/Group/A/B/C/D`
	and your path is
	`/Group/A/B`
	If the `rdf:resource` parameter is Node, server group node B is exported.
	If the `rdf:resource` parameter is Path, server group nodes A and B are exported.
	If the `rdf:resource` parameter is Descendants, server group nodes B, C and D are exported.

Table 2-25: Server Group Filter Nested Elements (continued)

ELEMENT	DESCRIPTION
`customerName (optional)`	This optional element restricts the export of attached server group nodes so that only those attached nodes owned by this customer get exported. The customerName element does not affect the export of nodes referenced by dynamic server group rules.

Notes

- Core specific information such as group membership and “Date last used”, or History properties, are not exported.
- Static groups can also be exported; however, only the name and description of the group are exported.
- If a dynamic group rule references a custom field, the custom field schema will only export the individual custom field, not the whole schema.
- The path defines whether a group is public or private. So all public groups can be exported by specifying a path of /Group/Public (and Descendants directive).
- Private groups cannot be exported, so a path of /Group/Private will result in an error during export.
- It is possible for an imported dynamic server group to not have any rules. This can happen if the source group only had rules like “Facility is C07” or “Realm is Sat02”. Since Facility and Realm are core specific, these rules are not exported.
- Also, any rules that reference Server IDs will not be exported. For example rules like “Server ID equals 55500001" will not be exported.

Server Group Export Filter Example

Export all public server groups from a mesh:

```
<ServerGroupFilter rdf:ID="exportPubServGroups">
    <path>/Group/Public/</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServerGroupFilter>
```

Export the public server group named “NT Servers” including all sub groups that belong to it:

```
<ServerGroupFilter rdf:ID="exportNTServGroups">
    <path>/Group/Public/NT Servers</path>
    <directive rdf:resource="&filter;Descendants"/>

</ServerGroupFilter>
```

Export only the public server group named “Production Web Servers” (but none of its subgroups):

```
<ServerGroupFilter rdf:ID="exportProdWebServGroups">
    <path>/Group/Public/Production Web Servers</path>
    <directive rdf:resource="&filter;Node"/>

</ServerGroupFilter>
```

Export the public group named “Production Web Servers” and its subgroup named “iPlanet”, but no other subgroups.

```
<ServerGroupFilter rdf:ID="exportProdWebServGroupsIP">
  <path>/Group/Public/Production Web Servers/iPlanet</path>
  <directive rdf:resource="&filter;Path"/>
</ServerGroupFilter>
```

Service Level Export Filter

The service level export filter tells DET what service level nodes to export.

Table 2-26: Service Level Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID=“unique name”.`

Table 2-27: Service Level Export Filter Nested Elements

ELEMENT	DESCRIPTION
`path (required)`	An absolute path from the top level node to the node to be exported. The path separator is “/”.

Table 2-27: Service Level Export Filter Nested Elements (continued)

ELEMENT	DESCRIPTION

`directive (required)`	An empty content element with a single `rdf:resource` parameter. The parameter refers to one of three constants:
	• Descendants: Export all descendants of the given path including the leaf of the path.
	• Node: Only export the given node.
	• Path: Export all nodes along the path and no other nodes.
	For example, given the following path:
	`/Service Levels/A/B/C/D`
	and your path is
	`/Service Levels/A/B`
	If the `rdf:resource` parameter is Node, node B is exported.
	If the `rdf:resource` parameter is Path, nodes A and B are exported.
	If the `rdf:resource` parameter is Descendants, nodes B, C and D are exported.

`customerName (optional)`	This optional element restricts the export to nodes owned by this customer at or below the specified path. If the node specified by the path is not owned by the specified customer, nothing is exported and a warning is logged. For examples of how this element works in a filter file, see “customerName Element Examples” on page 55.

Service Level Export Examples

Export the /Service Levels/Foo node only.

```
<ServiceLevelsFilter rdf:ID="exportServLevfoo">
  <path>/Service Levels/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</ServiceLevelFilter>
```

Export Bar and Baz nodes along the given path. Note that the stack root is not exported.

```
<ServiceLevelFilter rdf:ID="exportServLevBarBaz">
  <path>/ServiceLevels/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</ServiceLevelFilter>
```

Export the Gold Service Level node and all of its descendants, including the leaf node.

```
<ServiceLevelFilter rdf:ID="exportServLevGold">
  <path>/ServiceLevels/Gold</path>
  <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

Template Export Filter

The template export filter tells DET what template nodes to export.

Table 2-28: Template Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name".`

Table 2-29: Template Export Filter Nested Elements

ELEMENT	DESCRIPTION
`path (required)`	An absolute path from the top level node to the node to be exported. The path separator is "/".

Table 2-29: Template Export Filter Nested Elements (continued)

ELEMENT	DESCRIPTION

`directive (required)`	An empty content element with a single `rdf:resource` parameter. The parameter refers to one of three constants:
	• Descendants - export all descendants of the given path including the leaf of the path.
	• Node - only export the given node.
	• Path - export all node along the path and no other nodes.
	For example, given the following path:
	`/Templates/A/B/C/D`
	and your path is
	`/Templates/A/B`
	If the `rdf:resource` parameter is Node, node B is exported.
	If the `rdf:resource` parameter is Path, nodes A and B are exported.
	If the `rdf:resource` parameter is Descendants, nodes B, C, and D are exported.

`customerName (optional)`	This optional element restricts the export to nodes owned by this customer at or below the specified path. If the node specified by the path is not owned by the specified customer, nothing is exported and a warning is logged. For examples of how this element works in a filter file, see “customerName Element Examples” on page 55.

Template Export Filter Examples

Export the /Templates/Foo node only.

```
<TemplateFilter rdf:ID="exportTemplatesfoo">
  <path>/Templates/Foo</path>
  <directive rdf:resource="&filter;Node"/>
</TemplateFilter>
```

Export Bar and Baz nodes along the given path. Note that the stack root is not exported.

```
<TemplateFilter rdf:ID="exportTemplatesBarBaz">
  <path>/Templates/Bar/Baz</path>
  <directive rdf:resource="&filter;Path"/>
</TemplateFilter>
```

Export the Alpha Template node and all of its descendants, including the leaf node.

```
<TemplateFilter rdf:ID="exportTemplatesAlpha">
  <path>/Templates/Alpha</path>
  <directive rdf:resource="&filter;Descendants"/>
</TemplateFilter>
```

User Group Export Filter

The User Group export filter tells DET what user groups to export. A user group export includes the following information for each user group:

- Name
- Description
- The checked state of each feature in the Features tab (of the OCC user group administration page)
- The checked state of each permission in the Other tab.
- The read, read & write, none state of each role class stack in the Node Stacks tab
- The read, read & write, none state of each customer in the Customers tab
- The read, read & write state of each server group in the Server Groups tab

- The read, read & write, none or yes, no state of each feature in the Client Features tab.

Table 2-30: User Groups Export Filter Parameters

PARAMETER	DESCRIPTION
`rdf:ID`	Each filter has a unique name that is specified in the filter file using the format `rdf:ID="unique name"`

Table 2-31: User Groups Export Filter Nested Elements

`groupName (optional)`	An optional element that allows you to export specific user groups by name. If `groupName` is not specified, then all user groups will be exported.

Notes

- The membership of users and facility permissions (as seen in the Users and Facilities tabs) are not exported.
- The Customers and Server Groups tabs currently list all customers and server groups respectively to allow the read, read & write, none state to be set. Only customers and server groups that are configured with read or read & write will be exported.

User Group Export Filter Examples

Export all user groups from a mesh.

```
<UserGroupFilter rdf:ID="exportAllUserGroups"/>
```

Export the group named “SuperUsers”:

```
<UserGroupFilter rdf:ID="exportUserGroups">
  <groupName>SuperUsers</groupName>
</UserGroupFilter>
```

Export three user groups with the names “AdvancedUsers,” “OpswareAdministrators,” and “BasicUsers”:

```
<UserGroupFilter rdf:ID="exportAdvUsersGroup">
  <groupName>AdvancedUsers</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportOpsUsersGroup">
```

```
  <groupName>OpswareAdministrators</groupName>
</UserGroupFilter>
<UserGroupFilter rdf:ID="exportBasicUsersGroup">
  <groupName>BasicUsers</groupName>
</UserGroupFilter>
```

customerName Element Examples

These examples illustrate how the customerName element works for the Application, Service Level, Template, and Server Group export filters.

This section contains two topics:

- customerName Examples for Applications, Service Levels, Templates
- customerName Examples for Server Groups

customerName Examples for Applications, Service Levels, Templates

Given this node hierarchy:

```
Service Levels (owned by Customer Independent)
    A (Customer Independent)
        B (Customer Independent)
        C (Nike & Adidas)
            D (Nike)
```

- If your file specifies the following filter definition, then A, B, C, and D will be exported. In other words, the service levels of all customers.

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Levels/A</path>
    <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

- If your file specifies the following filter definition, then A and B will be exported. C and D will be skipped

```
ServiceLevelFilter rdf:ID="a1">
    <path>/Service Levels/A</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Customer Independent</customerName>
</ServiceLevelFilter>
```

- If your file specifies the following filter definition, then C and D will be exported.

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Levels/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Nike</customerName>
</ServiceLevelFilter>
```

- If your file specifies the following filter definition, then only C will be exported. D will be skipped because it is not owned by Adidas.

```
<ServiceLevelFilter rdf:ID="a1">
    <path>/Service Levels/A/C</path>
    <directive rdf:resource="&filter;Descendants"/>
    <customerName>Adidas</customerName>
</ServiceLevelFilter>
```

- If your file specifies the following filter definition, then nothing will be exported:

```
<ServiceLevelFilter rdf:ID="a1">
<path>/Service Levels/A</path>
<directive rdf:resource="&filter;Descendants"/>
<customerName>Nike</customerName>
</ServiceLevelFilter>
```

customerName Examples for Server Groups

The examples illustrate how the customerName Element works for the Server Group filter.

For example, if your core had this server group hierarchy:

Server Groups

Public

SG1

+ /Application Servers/A (owned by Customer Independent)

+ /System Utilities/B (Nike)

+ /Web Servers/C (Adidas)

- If your file specifies the following filter definition, then SG1, A, B, and C will be exported.

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
</ServerGroupFilter>
```

- If your file specifies the following filter definition, then SG1 and A will be exported.

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    customerName>Customer Independent</customerName>
</ServerGroupFilter>
```

- If your file specifies the following filter definition, then SG1 and B will be exported.

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Nike</customerName>
</ServerGroupFilter>
```

- If your file specifies the following filter definition, then server group SG1 will be exported.

```
<ServerGroupFilter rdf:ID="a1">
    <path>/Group/Public/SG1</path>
    <directive rdf:resource="&filter;Node"/>
    <customerName>Acme</customerName>
</ServerGroupFilter>
```

Importing Content

The Import process imports content to a target Opsware mesh.

Content import using the DET into an Opsware mesh is supported on a forward compatible basis. For example, you can import content from an Opsware SAS 4.7 mesh into an Opsware SAS 5.5 mesh. (But you cannot do this in reverse.)

The import command is:

```
cbt -i <content_dir> -p <policy> -cf <target_core_config> --noop
```

The command and its arguments indicate:

- `content_dir` – the directory containing the previously-exported content
- `policy` – the import policy that DET should use when it detects duplicates in the target Opsware mesh. See the "Policy on Importing Content Types" on page 58.

- `target_core_config` - a configuration file that tells DET where the various Opsware components are located, and what identity it should use to access them. Instructions for creating this file are located at “Installing and Configuring DET” on page 13.
- --noop – Run the import in a “dry run” mode. In other words, don't modify any data. Instead, output a summary of what changes would be made if run normally.

See “DET Command Line” on page 69 for a complete list all the available arguments and their meanings.

When Applications are imported using DET, the associated package name in the Opsware mesh receives a “cbt” suffix. For example:

```
openssh-3.8p1-sol8-sparc-local_cbt796213986
```

Policy on Importing Content Types

The following table shows the affect of the policy you specify on the command-line for each content type when duplicates are found.

The choices are:

- `overwrite` - the default if no policy is specified. The effect of this option is different for each content type as described in the table.
- `duplicate` - the effect of this option is different for each content type as described in the following table.
- `skip` - for all content types, specifying “skip” means that if a duplicate is found, a message is entered in the session log and the import continues.

See “DET Command Line” on page 69 for a complete list of all the available arguments and their meanings.

Table 2-32: Policies Used By DET When Importing Each Content Type

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
Application	• Custom attributes • Config tracking policy • Install order • Software list • Customer	Content information overrides existing node in target Opsware mesh without changing its node ID. Content information is overlaid on the existing node.	Content information is renamed by applying a “cbt-<random>” suffix to the application name.
Application Configuration	Account Application Configuration File	All attributes are updated in overwrite mode.	New Application Configuration is created and named “Oldname-cbt-<random>”
Application Configuration Template	Account	All attributes are updated in overwrite mode.	New Application Configuration template is created and named “Oldname-cbt-<random>”
Config Tracking Policy	NA	Creates and overrides existing policy.	Same as Install Order Relationship.
Custom Attributes	NA	Creates and overrides existing keys. The result is the union of the imported key and existing keys.	Same as Install Order Relationship.
Custom Extension	NA	A new version of the script is created.	Same as overwrite policy.

Table 2-32: Policies Used By DET When Importing Each Content Type (continued)

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
Custom Field Schema	NA	Display name is the only field that is updated.	Do nothing on duplication.
Customer	NA	Do nothing on duplication.	Do nothing on duplication. Please see “Synchronizing Multimaster Meshes with Deltas” on page 65 for important information about importing customers.
Distributed Script	NA	A new version of the script is created.	Same as overwrite policy.
Install Hooks	NA	See Unit.	See Unit.
Install Order Relationship	NA	Creates the relationship regardless and override the existing relationship.	Since this is done in the context of the parent node, a new relationship is always created because a parent node is always created - albeit with a different name.
MRL	NA	Always create an MRL in the target mesh using the identical name as in the source mesh.	Same as overwrite.

Table 2-32: Policies Used By DET When Importing Each Content Type (continued)

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
OS	• Custom attributes • Config tracking policy • Customer • Software list • InstallHooks • MRL	Content information overrides existing node in target Opsware mesh without changing its node ID. Content information is overlaid on the existing node.	Content information is renamed by applying a "cbt-<random>" suffix to the application name.
Package	NA	Package is uploaded over the existing package and will overwrite the "container" package types: LPP, HPUX Depot, and Solaris Package. These package types will be overwritten with the new data if their new contents (contained packages) are a superset of the old contents. If not, DET will revert to the existing "rename" mode.	Same as overwrite.

Table 2-32: Policies Used By DET When Importing Each Content Type (continued)

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
Patch	NA	Physical patch package is uploaded and contained units are created in the Software Repository. AIX LPPs and HPUX Depots, package types will be overwritten with the new data if their new contents (contained packages) are a superset of the old contents. If not, DET will revert to the existing "rename" mode.	Same as overwrite. This is because Opsware SAS cannot reliably and efficiently determine whether a package in the Software Repository is equivalent to the package being uploaded.
Patch Knowledge (PATCH_ META_DATA)	NA	The patch database is imported into Opsware SAS, overwriting the existing database, if there is one. The knowledge created by the import will depend on the patch preference settings in the target Opsware mesh.	Same as overwrite.
Patch Policy	Patch	Description and list of patches are updated.	New patch policy is created and named "Oldname-cbt<random>"
Server Compliance Criteria	NA	All attributes are updated in overwrite mode.	New Server Compliance Criteria is created and named "Oldname-cbt-<random>"

Table 2-32: Policies Used By DET When Importing Each Content Type (continued)

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
Server Group	• Application • Custom Attribute • Custom Field Schema • Patch • Server Group • Service Level	Group description and type are updated. Dynamic group rules are overwritten. The match “if any rules are met” and “if all rules are met” setting will be updated to reflect what is defined in the export. Custom attributes are overlaid. Attachments to patches, applications and service levels are overwritten.	New server group is created and named “Oldname-cbt<random>”.
Service Level	• Custom attributes • Config tracking policy • Customer	Content information overrides existing node in target Opsware mesh without changing its node ID. Content information is overlaid on the existing node.	Content information is renamed by applying a “cbt-<random>” suffix to the template name.
Software List	• Unit	Creates and overrides existing list.	Same as Install Order Relationship.
Template	• Custom attributes • Customer • Application • Patch • OS • Service Level	Content information overrides existing node in target Opsware mesh without changing its node ID. Content information is overlaid on the existing node.	Content information is renamed by applying a “cbt-<random>” suffix to the template name.

Table 2-32: Policies Used By DET When Importing Each Content Type (continued)

CONTENT TYPE	ASSOCIATED CONTENT TYPES	IMPORT POLICY (OVERWRITE)	IMPORT POLICY (DUPLICATE)
Unit	Unit script	Units are associated with a physical package, see Package content type above. Virtual units are always associated with existing units in the target Opsware mesh - this is presumably created as a side effect of uploading the physical package that is also part of the same import session.	Same as overwrite.
Unit Script	NA	Created and overrides existing unit scripts.	Same as overwrite.
User Group	Customer Server Group	User group description is updated. In addition, the checked state of features (as seen in the Features and Other tabs) will be updated to reflect what is in the export. The Read, Read & Write, and None settings of customers, node stacks, and client features will be updated to reflect what is in the export. The Read and Read & Write settings of server groups are updated as well.	New user group is created and named “Oldname-cbt<random>”.

Synchronizing Multimaster Meshes with Deltas

This release of DET provides the means of performing ‘incremental” exports and imports, which helps you keep the content in your multimaster mesh synchronized and up to date.

For example, you can run regular exports from your “source” mesh that represents all the content you want other meshes to contain. Using the new options allows you to export only content that has been modified or deleted so that your target mesh are consistent with the source mesh.

Delta Exports

These command-line options allow you to perform an delta export:

- `--baseline` (short form: `-b`)

  Specifies a baseline export against which to compare the current export. This requires that either `--incremental` or `--delete` be specified during export.

- `--incremental` (short form: `-incr`)

  Of the content specified by the filter file, export only that which has been added or modified since the baseline. If this option is not given, all content specified by the filter file is exported. Must be used with `--baseline`.

- `--delete` (short form: `-del`)

  Include in the export any content in the baseline that is not specified by the filter file, marked “as deleted”. If this option is not given, nothing is exported “as deleted”. Must be used with `--baseline`.

Here is what happens when you use `--delete` and `--incremental` in combination with `--baseline` during an export:

- No incremental export options.

  All content specified by filter file is exported.

- `-incr`

  All content specified by filter file that is new or changed since the baseline is exported.

- `-del`

  All content specified by filter file is exported (since `-incr` is not given), plus all content in the baseline that is not specified by the filter file (“as deleted”).

- `-incr -del`

  All content specified by filter file that is new or changed since the baseline is exported, plus all content in the baseline that's not specified by the filter file ("as deleted").

Delta Imports

This command-line options allows you to perform a delta import (if certain options were given during export):

- `--delete` (short form: `-del`)

  If the `--baseline` option was given with `--delete` during export, then using the `--delete` option during import will delete objects that have been marked for deletion from the export.

  If the `--baseline` option was given with `--delete` during export, but you do not use --delete during import, the items marked for deletion will not be deleted but rather renamed. For more information on cases in which some content may never get deleted and always renamed (for example, if the object has a dependency elsewhere in the mesh) then see "Import Delete Conditions" on page 79.

Mesh Synchronization Usage Scenario

Here is what a typical incremental export and import cycle might look like when content in the source mesh has been both deleted and modified:

1. Initial, full export of a filter that exports Application Configuration content:

```
cbt -e content/appConfig.0 -f ac_Filter.rdf -cf meshA_Config
```

2. Import exported content into another mesh:

```
cbt -i content/appConfig.0 -p overwrite -cf meshB_Config
```

   Content is changed and deleted in source mesh.

3. Export the modified and deleted content from the source mesh using `-b` and `-incr` and `-del`:

```
cbt -e content/appConfig.1 -f ac_Filter.rdf -b content/
appConfig.0 -incr -del -cf meshA_Config
```

4 Import the delta into the destination mesh, updating the modified content and deleting the deleted content:

```
cbt -i content/appConfig.1 -p overwrite -cf meshB_Config -
del
```

5 Repeat steps four and five every time you want to update content, using the most recent export as your baseline. For example, on the next round you would use:

- Export content/appConfig.2 with -b content/appConfig.1.
- Import content/appConfig.2.

Considerations When Importing Customers

Currently, DET does not support the export of user group permissions that are associated with customers, except in cases when the customer name being exported has the same name as a customer in the target mesh (the mesh you are importing the customer into).

For example, let's say that in your source mesh, you had a software application node named iPlanet, and that software application node iPlanet was accessible for reading and writing to all groups associated with a customer named Computing Machines. One of these groups associated with the customer Computing Machines was named groupA.

Next, you export a software application node iPlanet from the source mesh, and then import that node into a new mesh – and this mesh does not have a customer named Computing Machines. The result would be that any users in groupA would not be able to see software application node iPlanet in the target mesh.

However, if the mesh you imported the customer Computing Machines into already has a customer with exactly the same name, then all permissions are untouched in the new mesh and all users groupA would be able to access the software application node named iPlanet – in other words, all permissions associated with the Computing Machines customer (the ability to read and write the software node iPlanet) will remain in tact.

Importing Customers Workaround

If a user group loses permissions to access objects (such as servers associated with a customer), then use the Opsware Command Center to re-assign the permissions. Until doing so, only users who are administrators will see these customers and their associated objects.

Content Directory

The content directory is the persistent store of exported Opsware content. The content directory contains:

- `data.rdf` - a database of exported Opsware configuration content.
- `filter.rdf` - a database of filters provided by the user and generated by DET.
- `blob/` - a directory containing exported software packages and scripts.
- `var/` - a directory containing logs for each of the last ten import and export sessions. Logs are named `cbtexport {0-9}.log` and `cbtimport {0-9}.log`. The 0 log is always the most recent and the 9 log file is always the oldest of the ten session logs.

The following is an example content directory.

```
% ls -R
.:
blob
data.rdf
filter.rdf
var

./blob:
unitid_140270007.pkg
unitid_166510007.pkg
unitid_166540007.pkg
unitid_2090007.pkg

./var:
cbtexport0.log
cbtexport0.log.lck
cbtimport0.log
```

Appendix A: DET Command Line

IN THIS APPENDIX

This Appendix contains the following topics:

DET Command Line Overview

The DET command line is pre-configured to be executed as the user root on a managed server. If used in this configuration, you will only have to provide your Opsware user name and password to perform an export or an import. The following is an example session: (The example below assumes the user has been granted import and export permission. See “Installing and Configuring DET” on page 13 in Chapter 1 for more information.)

The following is an example csh session on the Opsware Command Center server.

```
% setenv JAVA_HOME <j2re 1.4.x installation>
% <cbt install dir>/bin/cbt -e /tmp/foo -f <cbt install dir>/
filters/app.rdf --spike.username hermaime
Enter password for hermaime: ********
...
```

Commands and Options

The following sections describe the syntax of the DET command line interface. The DET command line tool consists of the following commands:

- Export Command
- Import Command
- Show Export Status Command
- Configuration File Command
- Show Version Command
- Show Help Command
- DET Permissions Command

Export Command

The export command uses the following syntax:

```
cbt -e <content_dir> [-f <filter_file>] [-b <content_dir>] [-
incr] [-del] [-cf file] [-c] [-d] [-np] [-nd] [-lx] [-em
<addrs>] [--emaillog]
```

Table A-1: Export Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-e`	`--export`	Export Opsware data from an Opsware core and store the data in the given content directory.

Table A-1: Export Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-f`	`--filter`	The first time you export, you must specify a filter file describing what data to export. After that, if no filter is specified, then any previously-used filter in the content directory is used. For more information on the DET filter file, see “Example Export Filter File” on page 26.
`-b`	`--baseline`	Specifies a baseline export against which to compare the current export. This requires that either `--incremental` or `--delete` be specified during export.
`-incr`	`--incremental`	Performs an incremental export. Of the content specified by the filter file, export only that which has been added or modified since the baseline. If this option is not given, all content specified by the filter file is exported.

Table A-1: Export Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-cf`	`--config`	Specifies the DET configuration file. For more information about the DET configuration file, see “Creating a Target Mesh Configuration File” on page 14.
`-c`	`--clean`	Remove previously exported data from the content directory given by `-e`.
`-d`	`--debug`	Show more detailed debug information.
`-del`	`--delete`	Include in the export any content in the baseline that's not specified by the filter file, marked “as deleted”. If this option is not given, nothing is exported “as deleted”. If used, `--baseline` must also be used to specify the baseline export.
`-np`	`--noprogress`	Don't show the progress on the console.
`-nd`	`--nodownload`	Don't download the units from Software Repository (the word). IMPORTANT: Exports using this option cannot be imported.

Table A-1: Export Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-lx`	`--logxml`	Create log file in XML format.
`-em`	`--email`	Email a summary of the export to this comma-separated list of addresses. In order for this option to work, you must have added the email notification parameters to the DET configuration file.
`(none)`	`--emaillog`	Include the entire log file in the email.

Change for -c Option

The -c option can no longer be used on its own and must instead be used with the export command.

For example, previously, you would have cleaned out the content directory using the following command:

```
$ cbt -c <content-dir>
```

Now, the -c (or --clean) flag can only be used with the export command. For example:

```
$ cbt -e <content-dir> -c (or --clean)
```

Import Command

The import command uses the following syntax:

```
cbt -i <content_dir> [-p <overwrite|duplicate|skip>] [-del] [-n]
[-cf file] [-d] [-np] [-nu] [-lx] [-em <addrs>] [--emaillog]
```

Table A-2: Import Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-i`	`--import`	Import Opsware data from the given content directory.

Table A-2: Import Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-p`	`--policy`	Import policy. Default is "overwrite." "overwrite" means to override objects in the same name space on the target Opsware SAS without affecting its object IDs. "duplicate" means to create a duplicate copy of an object with a synthetic name when a duplicate is detected on the target Opsware SAS. "skip" is the most conservative policy. It aborts the import of an object if the same object is detected in the target Opsware SAS. For more information on import policies for the specific content types, see See "Policy on Importing Content Types" on page 58.
`-del`	`--delete`	Delete objects marked deleted by the export. (In other words, this option will only work if the `-del` option was given during export. If this option is not given, the objects will be renamed.

Table A-2: Import Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-n`	`--noop`	Run the import in a “dry run” mode. In other words, don't modify any data. Instead, output a summary of what changes would be made if run normally.
`-cf`	`--config`	Read configuration from the given file.
`-d`	`--debug`	Show more detailed debug information.
`-np`	`--noprogress`	Don't show the progress on the console.
`-nu`	`--noupload`	Don't upload unchanged packages to the Software Repository (the word).
`-lx`	`--logxml`	Create log file in XML format.
`-em`	`--email`	Email a summary of the import to this comma-separated list of addresses. In order for this option to work, you must have added the email notification parameters to the DET configuration file.
`(none)`	`--emaillog`	Include the entire log file in the email.

Show Export Status Command

The show export status command uses the following syntax:

```
cbt -t <content_dir>
```

Table A-3: Show Export Status Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-t`	`--showstatus`	Show status of export of the given content directory.

Configuration File Command

The configuration file command option uses the following syntax:

```
cbt -s [-cf config]
```

Table A-4: Configuration File Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-s`	`--showconfig`	Show current configuration values.
`-cf`	`--config`	Read configuration from the given file.

Show Version Command

The show version command uses the following syntax:

```
cbt -v
```

Table A-5: Show Version Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-v`	`--version`	Show the version of the DET tool.

Show Help Command

The show help command uses the following options:

```
cbt -h
```

Table A-6: Show Help Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-h`	`--help`	Display this help message.

DET Permissions Command

The DET permissions command uses the following syntax:

```
cbtperm -u [user] -a [spike.username] -p [spike.port] -s
[spike.host] -c [ssl.trustCerts] -k [ssl.keyPairs]
```

Table A-7: DET Permissions Command Options

SHORT OPTION	LONG OPTION	DESCRIPTION
`-u`	`N/A`	The user to whom you want to grant permission to use the DCML Exchange Tool.
`-a`	`--spike.username`	User name for Spike authentication, such as the Opsware administrator.
`-p`	`--spike.port`	Spike's listener port.
`-s`	`--spike.host`	Spike's hostname or IP.

Table A-7: DET Permissions Command Options (continued)

SHORT OPTION	LONG OPTION	DESCRIPTION
`-c`	`--ssl.trustCerts`	Comma-separated list of trust certificate files to be used to communicate with XML-RPC servers
`-k`	`--ssl.keyPairs`	Comma-separated list of client certificates to be used to communicate with XML-RPC servers

Appendix B: Import Delete Conditions

IN THIS APPENDIX

This appendix contains the following topic:

- Import Delete Conditions for 'Marked As Deleted' Objects
- When Objects Get Renamed and Become Un-findable

Import Delete Conditions for 'Marked As Deleted' Objects

If you have specified that content be marked as deleted during an export, running the `--delete` option on import will delete those marked items from the destination mesh.

In some cases, however, if the content marked for deletion in the destination mesh is being used by parts of the Opsware model, DET will take a 'no harm' approach by renaming the content item instead of deleting it. Or, if you used the `-del` option during export but did not use the `-del` option during import, then any content items marked for deletion in the export will not be deleted in the destination mesh – they will instead be renamed.

When a content item is renamed in the destination mesh, the following naming convention is used for the renamed item:

```
<item_name>-cbtDeleted<12345>
```

For example, if Application Configuration "foo" is renamed during one DET run, it would be renamed to "`foo-cbtDeleted134234`".

Table B-1 describes all conditions that must be met for a content item to be deleted on an import, and those cases in which the content item will be renamed. If the conditions for allowing delete are not met, then the item will be renamed according to the renaming convention.

For some content items, there are no restrictions and they will always be marked as deleted when the delete option is used for both import and export. For other content items, deletion will never be allowed.

When Objects Get Renamed and Become Un-findable

When a content item is renamed for any reason (no `-del` or “do no harm’”), it may become un-findable by DET on subsequent imports. This reason for this is that the name by which the item is located in the destination mesh has been changed due to the rename.

For example, if Application Configuration “foo” is renamed during one DET run to “foo-cbtDeleted134234”, on subsequent runs the DET will attempt and fail to find an Application Configuration named “foo”. This will prevent the DET from re-renaming or deleting the Application Configuration.

Types of objects with dependencies that can become unfindable after they get renamed include Application, Application Configuration, Application Configuration file, Compliance Selection Criteria, Custom Extension, Distributed Script, OS, Patch Policy, Server Group, Service Level, Template.

Table B-1: Condition for Content Items to Be Deleted Upon Import with the -del Option

OBJECT TYPE	CONDITIONS ALLOWING DELETE
Application	Zero attached devices. Zero child nodes. Zero templates or server groups include this node.
Application Configuration	In use by zero servers or server groups.
Application Configuration File	In use by zero application configurations.
Compliance Selection Criteria	Always allow delete.
Custom Extension	Never allow delete; always rename.
Custom Field Schema	Always allow delete.

Table B-1: Condition for Content Items to Be Deleted Upon Import with the -del Option

OBJECT TYPE	CONDITIONS ALLOWING DELETE
Customer	Zero application, service level, and template nodes. Zero non-deactivated devices. Zero packages (including those with status DELETED). Zero IP range groups. Note: A Customer cannot be deleted if it has any packages still in Opsware, including those with the status DELETED. When an object has a DELETED status, it means that either a) the package is still needed for reconcile operations on at least one server, or b) the Satellite Software Repository has not yet flushed the package. If this is the case, then the Customer marked for deletion will not be deleted, but renamed.
Deployment Stage Value	Zero devices using this value.
Disturbed Script	Always allow delete.
OS	Zero attached devices. Zero child nodes. Zero templates or server groups include this node.

Table B-1: Condition for Content Items to Be Deleted Upon Import with the -del Option

OBJECT TYPE	CONDITIONS ALLOWING DELETE
Package	Is a deletable unit type (see below) Zero Solaris patch clusters or MRLs use this package. Zero OS definitions or application nodes use this package. If a patch: • Zero devices attached to the patch node. • Zero templates or server groups include the patch node. • Zero patch policies or patch exceptions include the patch node. If an LPP, HPUX depot, or Solaris package: • Zero OS definitions or application nodes use any sub-package. • For any sub-package that is a patch: – Zero devices attached to the patch node. – Zero templates or server groups include the patch node. – Zero patch policies or patch exceptions include the patch node. Deletable package unit types: • HOTFIX • HPUX_DEPOT • LPP • MRL • MSI • PROV_INSTALL_HOOKS • RPM • SERVICE_PACK • SOL_PATCH • SOL_PATCH_CLUSTER • SOL_PKG • SP_RESPONSE_FILE • UNKNOWN • UPDATE_ROLLUP • WINDOWS_UTILITY • ZIP

Table B-1: Condition for Content Items to Be Deleted Upon Import with the -del Option

OBJECT TYPE	CONDITIONS ALLOWING DELETE
Patch Policy	Zero attached devices. Zero attached device groups.
Server Group	Zero attached devices. Zero child nodes. Not used by access control. Zero dynamically bound jobs.
Server Use Value	Zero devices using this value.
Service Level	Zero attached devices. Zero child nodes. Zero templates or server groups include this node.
Template	Zero child nodes.
User Group	Always allow delete.