HP Server Automation

Ultimate Edition

Software Version: 10.23

Content Utilities Guide

Document Release Date: June 2016

Software Release Date: June 2016

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

Intel® and Itanium® are trademarks of Intel Corporation in the U.S. and other countries.

Microsoft®, Windows®, Windows® XP are U.S. registered trademarks of Microsoft Corporation.

Oracle and Java are registered trademarks of Oracle and/or its affiliates.

UNIX® is a registered trademark of The Open Group.

This product includes software developed by the Apache Software Foundation (http://www.apache.org/).

Support

Visit the HP Software Support Online website at:

https://softwaresupport.hp.com/

This website provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support website to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

https://hpp12.passport.hp.com/hppcf/createuser.do

To find more information about access levels, go to:

https://softwaresupport.hp.com/web/softwaresupport/access-levels

Support Matrices

For complete support and compatibility information, see the support matrix for the relevant product release. All support matrices and product manuals are available here on the HP Software Support Online website:

https://softwaresupport.hp.com/group/softwaresupport/support-matrices

You can also download the *HP Server Automation Support and Compatibility Matrix* for this release from the HP Software Support Online Product Manuals website:

https://softwaresupport.hp.com/

This site requires that you register for an HP Passport and sign in. After signing in, click the **Search** button and begin filtering documentation and knowledge documents using the filter panel.

Documentation Updates

All the latest Server Automation product documentation for this release is available from the SA Documentation Library:

https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

Use the SA Documentation Library to access any of the guides, release notes, support matrices, and white papers relevant to this release or to download the full documentation set as a bundle. The SA Documentation Library is updated in each release and whenever the release notes are updated or a new white paper is introduced.

How to Find Information Resources

You can access the information resources for Server Automation using any of the following methods:

Method 1: Access the latest individual documents by title and version with the new SA Documentation Library

Method 2: Use the complete documentation set in a local directory with All Manuals Downloads

Method 3: Search for any HP product document in any supported release on the HP Software Documentation Portal

To access individual documents:

1. Go to the SA 10.x Documentation Library:

   https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

2. Log in using your HP Passport credentials.
3. Locate the document title and version that you want, and then click **go**.

To use the complete documentation set in a local directory:

1. To download the complete documentation set to a local directory:
   a. Go to the SA Documentation Library:

https://softwaresupport.hp.com/group/softwaresupport/search-result/-/facetsearch/document/KM00417675

- b Log in using your HP Passport credentials.
- c Locate the All Manuals Download title for the SA 10.1 version.
- d Click the **go** link to download the ZIP file to a local directory.
- e Unzip the file.

2 To locate a document in the local directory, use the Documentation Catalog (docCatalog.html), which provides an indexed portal to the downloaded documents in your local directory.

3 To search for a keyword across all documents in the documentation set:

- a Open any PDF document in the local directory.
- b Select **Edit > Advanced Search** (or Shift+Ctrl_F).
- c Select the All PDF Documents option and browse for the local directory.
- d Enter your keyword and click Search.

To find additional documents on the HP Software Documentation Search Portal:

Go to the HP Software Documentation Search Portal:

https://softwaresupport.hp.com/

This site requires that you register for an HP Passport and sign in. After signing in, click the **Search** button and begin filtering documentation and knowledge documents using the filter panel.

To register for an HP Passport ID, click the **Register** link on the HP Software Support Online login page.

You can also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details. See Documentation Change Notes for a list of any revisions.

Product Editions

There are two editions of Server Automation:

- Server Automation (SA) is the Ultimate Edition of Server Automation. For information about Server Automation, see the SA Release Notes and the SA User Guide: Server Automation.
- Server Automation Virtual Appliance (SAVA) is the Premium Edition of Server Automation. For more information about what SAVA includes, see the SAVA Release Notes and the SAVA at a Glance Guide.

Documentation Change Notes

The following table indicates changes made to this document since the last released edition.

Date	Changes
December 2014	Original release of this document with SA 210.1

Contents

1 Importing and Exporting SA Content

This is intended for system administrators responsible for specifying SA content. You should be familiar with script programming and SA fundamentals. See the *SA User Guide: Server Automation*.

The DCML Exchange Utility (DET)

DCML, the Data Center Markup Language, is an XML-based language for describing elements and relationships in a data center environment. The DCML Exchange Utility (DET) is a command that exports and imports SA content. It enables you to inject a newly-installed SA Multimaster 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.

The cbt Command

The DET is simply a command, `cbt`, included with SA. For details on the `cbt` command, see The cbt Command Usage on page 13 and The cbt Command Reference on page 55.

The `cbtperm` command lets you set permissions for using DET. For details on the cbtperm command see DET Permissions Command, cbtperm on page 60.

In the context of DET, *content* means user-created SA server management information. This includes the following content types:

- Application Configurations, Application Configuration Templates
- Custom Extensions
- Custom Fields, Custom Attributes
- Customers
- Folders
- Packages
- Patches and Patch Policies
- Server Compliance Criteria
- Device Groups
- User Groups.

Content does not include managed environment type information. For example, facility information and server properties are not included.

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 Server Automation. The exported data.rdf is a valid DCML instance document that is parsable by a compliant DCML processor.

Custom Fields and Custom Attributes

Each custom field exists in a namespace. The DET only has access to (and thus will only export) these objects in the default, user-visible namespace. Objects in other namespaces (OPSWARE, etc.) will not be exported. If objects in other namespaces need to be exported (for example, OS sequences), they will be exported via application-specific APIs (for example, OS sequence APIs).

All custom attributes are exported, including those that are hidden from end-users (those keys starting with `__OPSW`).

For custom fields and attributes, imported values (including nulls) overlay existing values.

2 The cbt Command Usage

Exporting Content

The cbt command exports the content you specify from a target SA mesh to an RDF/XML file that can be imported into another SA mesh. See Importing Content on page 36.

The cbt command is found in the directory:

```
/opt/opsware/cbt/bin
```

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 SA mesh. See the Export Filters on page 13 for information on creating this file.
- target_core_config - a configuration file that tells DET where the various SA 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 50.

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 Command Center core component to prevent users from changing any SA 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 Command Centers in the mesh to prevent users from changing any SA content until the process has completed.

See the *SA Administration Guide* for information about stopping and restarting the Command Center core component.

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 Configuration Export Filter

- Application Configuration Template Export Filter
- Audit Filter
- Custom Extension Export Filter
- Custom Fields Schema Export Filter
- Customer Export Filter
- Folder Export Filter
- OS Build Plan Export Filters
- OS Export Filter
- Package Export Filter
- Patch Export Filter
- Patch Policy Export Filter
- Server Compliance Criteria (Audit Policies) Export Filter
- Server (Device) Group Export Filter
- Service Level Export Filter
- Snapshot 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.   xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
7. <PackageFilter rdf:ID="exportPackages">
8.<packageType rdf:resource="&filter;RPM"/>
9. <packageName>software1.0.0-1.rpm</packageName>
10. </PackageFilter>
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#"
```

```
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<PackageFilter rdf:ID="pkg1">
 <packageType rdf:resource="&filter;RPM"/>
 <packageName>software1.0.0-1.rpm</packageName>
</PackageFilter>
<PackageFilter rdf:ID="pkg2">
    <packageType rdf:resource="&filter;ZIP"/>
</PackageFilter>
<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 SA data types from an SA mesh.

Running CBT export commands requires specifying the absolute path for the selected filter. For example:

```
cbt -e /tmp -f /opt/opsware/cbt/filters/filter.rdf
```

The following sections describe each filter type and their allowed parameters. In general, filter types map to an object type that can be manipulated in the SA Client. The Patch Filter, for example, maps to the SA Client patch object.

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 1 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 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 3 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 4 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>
```

Audit Filter

The audit filter tells DET which audit to export from an SA core/mesh so that you can then import it into another SA core/mesh.

table 5 Audit 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 6 Audit Filter Nested Elements

Element	Description
auditPolicyName (optional)	The name of the audit policy that you want to export.
clearSource (optional)	Used to specify whether the source of the audit policy should be deleted in the exported content or not. Possible values: Yes or N Examples: <clearSource rdf:resource="&filter;Yes"/> <clearSource rdf:resource="&filter;No"/>
osType (optional)	Used to specify to export all Audit policies that have been associated with the specified OS. Possible values: Windows or Unix Examples: <osType rdf:resource="&filter;Windows"/> <osType rdf:resource="&filter;Unix"/>

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
```

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<AuditPolicyFilter rdf:ID="apf1">
  <auditPolicyName>My Audit Policy</auditPolicyName>
</AuditPolicyFilter>
</rdf:RDF>
```

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 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 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 9 Custom Fields 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 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 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>
```

Folder Export Filter

The folder filter tells DET to either export a specific Folder, including the following items that are associated with or contained in the Folder:

- Application Configuration Templates
- Application Configurations
- Attributes and custom attributes
- Contained audit policies
- Contained OS Build Plans
- Contained OS sequences
- Contained packages
- Contained scripts
- Contained software policies
- FolderACLs referencing user groups by name (user groups not exported).

- Placeholders for all folders along the path to the specified Folder
- Subfolders (optional)

table 12 Folder 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 13 Folder Export Filter Nested Elements

Element	Description
path (required)	A required element that specifies the folder path.
recursive (optional)	An optional element that specifies the export of sub-folders.

Folder Export Filter Examples

For example, suppose the following folder hierarchy.

```
/
/A
/A/B
```

The following examples list which folders are exported given the preceding folder hierarchy.

Export folder A:

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;No"/>
</FolderFilter>
```

Export folder B:

```
<FolderFilter rdf:ID="f1">
   <path>/A/B</path>
</FolderFilter>
```

Export folders A and B:

```
<FolderFilter rdf:ID="f1">
   <path>/A</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

Export folders A and B:

```
<FolderFilter rdf:ID="f1">
   <path>/</path>
   <recursive rdf:resource="&filter;Yes"/>
</FolderFilter>
```

OS Build Plan Export Filters

The OS Build Plan export filter tells DET what OS Build Plans to export.

table 14 OS Build Plan 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 15 OS Build Plan Export Filter Nested Elements

Element	Description
osBuildPlanName (optional)	The name of the OS build plan to export.
folderName (optional)	The name of the folder containing the OS Build Plans to export. **Note**: the folderName nested element refers to a folder name and not to a folder path. All folders with the given name will be taken into consideration when executing an OS Build Plan filter with the folderName nested element.

OS Build Plan Export Filter Examples

Exports all OS Build Plans:

```
<OSBuildPlanFilter rdf:ID="osbp1"/>
```

Export all OS Build Plans with "OS Build Plan 1" name:

```
<OSBuildPlanFilter rdf:ID="osbp2">
  <osBuildPlanName>OS Build Plan 1</osBuildPlanName>
</OSBuildPlanFilter>
```

Export all OS Build Plans located in "Build Plan Folder" folder:

```
<OSBuildPlanFilter rdf:ID="osbp3">
  <folderName>Build Plan Folder</folderName>
</OSBuildPlanFilter>
```

OS Export Filter

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

table 16 OS 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 17 OS Export Filter Nested Elements

Element	Description
`osName (optional)`	The name of the OS assigned by the user in the SA Client.
`osPlatform (required)`	A required nested element. This empty element has an `rdf:resource` parameter. This parameter may refer to one of the supported operating systems listed in the *SA Support and Compatibility Matrix*.

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. A placeholder for the containing folder is exported. Placeholders for all folders on the path to the containing folder are also exported.

➤ 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 SA 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 18 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 19 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 • Chef_Cookbook • DEB • HPUX_Depot • HPUX_Fileset • HPUX_Patch_Fileset • HPUX_Patch_Product • HPUX_Product • Relocatable_ZIP • RPM • Solaris_Package • Solaris_Package_Instance • Solaris_Patch • Solaris_Patch_Cluster • Unknown • Windows_Hotfix • Windows_MSI • Windows_OS_Service_Pack • Windows_Update_Rollup • 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 SA Client, not the filename of the package.
osPlatform (optional)	An optional element that allows you to specify the operating system of a named package. This parameter may refer to one of the supported operating systems listed in the *SA Support and Compatibility Matrix*.
customerName (optional)	An optional element that allows you to specify the customer of a named package.

Package Export Filter Example

Export all RPM packages associated to platform SunOS_5.8:

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

A relocatable ZIP file can be installed into different locations on a single server. Because the name of a relocatable ZIP file is the same as that of its parent ZIP file, specifying one will export all relocatable versions of that ZIP file. For example, suppose the ZIP file hierarchy is as follows:

- ZIP hmp.zip (SunOS 5.8)
    - Relocatable ZIP hmp.zip installed in /foo.
    - Relocatable ZIP hmp.zip installed in /bar.

For the preceding ZIP file hierarchy, with the following filter, both relocatable ZIP files will be exported (/foo and /bar).

```
<PackageFilter rdf:ID="p1">
    <packageType rdf:resource="&filter;Relocatable_ZIP"/>
    <packageName>hmp.zip</packageName>
    <osPlatform rdf:resource="&filter;SunOS_5.8"/>
</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 20 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 21 Patch Filter Nested Elements

Element	Description
patchType (required)	A required nested element that has an rdf:resource parameter. This parameter can 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

table 21 Patch Filter Nested Elements (cont'd)

Element	Description
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 SA Client.
patchLocale (optional)	The locale, which identifies the language of the Windows patch. This element is ignored for non-Windows patches. Examples of values for this element are en, ja, and ko. These values represent English, Japanese, and Korean. English is the default. For the list of locales currently supported by Windows patching, see the *SA User Guide: Patching Servers*.

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>
```

Export the patch named Q123456 for the Japanese locale.

```
<PatchFilter rdf:ID="pf1">
  <patchName>Q123456</patchName>
  <patchLocale>ja</patchLocale>
</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.

➤ The Patch Policy filter will not export Solaris Patch Policies. In order to export Solaris Patch Policies, you must export the parent folder using a Folder Export filter. (See Folder Export Filter on page 19.)

table 22 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 23 Patch Policy Export Filter Nested Elements

Element	Description
patchPolicyName	An optional element that specifies the unique name of the patch policy.
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 supported Windows operating systems using the following syntax: `<OS>_<Version>_<Revision(if applicable)>_<Architecture>` For example, for Windows 2008 R2 IA64, the valid OS Platform attribute would be Windows_2008_R2_IA64. Other examples of platforms would be Windows_2008_R2_x64, Windows_2012_R2_x64, and so on.

Patch Policy Export Filter Examples

Export all patch policies from the target mesh:

```
<PatchPolicyFilter rdf:ID="PatchPolicies1"/>
```

Export only the patch policies named “BestWindowsPoliciesNT" on the Windows NT 4.0 operating system, and “BestWindowsPolicies2003" on the Windows 2003 operating system:

```
<PatchPolicyFilter rdf:ID="PatchPolicies2"/>
  <patchPolicyName>BestWindowsPoliciesNT</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_NT_4.0/>
</PatchPolicyFilter>

<PatchPolicyFilter rdf:ID="PatchPolicies3"/>
  <patchPolicyName>BestWindowsPolicies2003</patchPolicyName>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

Export all Patch Policies for the Windows 2003 operating system:

```
<PatchPolicyFilter rdf:ID="PatchPolicies4"/>
  <osPlatform rdf:resource="&filter;Windows_2003"/>
</PatchPolicyFilter>
```

Script Export Filter

Scripts can only be exported by exporting their parent folder. See Folder Export Filter on page 19.

Server Compliance Criteria (Audit Policies) Export Filter

The Server Compliance Criteria export filter instructs DET what Audit Policies you want to export.

table 24 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 25 Server Compliance Criteria Filter Nested Elements

Element	Description
`selectionCriteriaNam (optional)`	Used to specify the name of the Audit Policy. Use this if you want to export specific Audit Policy by name.

`osType rdf:resource (optional)`	Used to specify exporting all Audit Policies that have been associated with the specified OS.
	Possible values: Windows or Unix
	Examples:
	`<osType rdf:resource="&filter;Windows"/>`
	`<osType rdf:resource="&filter;Unix"/>`

`clearSource (optional)`	Used to specify if the source of the Audit Policy should be deleted in the exported content or not.
	Possible values: Yes or No
	Examples:
	`<clearSource rdf:resource="&filter;Yes"/>`
	`<clearSource rdf:resource="&filter;No"/>`

Server Compliance Criteria Export Filter Examples

Export all Audit Policies.

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

Export the specific Audit Policy named “My Audit Policy" that has been associated with the Windows operating system.

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

Server (Device) Group Export Filter

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

table 26 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 27 Server Group Filter Nested Elements

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

`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.

`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.
- All attached software policies are 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 28 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 29 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 “/".
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 Level/A/B/C/D and your path is /Service Level/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 34.

Service Level Export Examples

Export the /Service Level/Foo node only.

```
<ServiceLevelFilter rdf:ID="exportServLevfoo">
  <path>/Service Level/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>/ServiceLevel/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>/ServiceLevel/Gold</path>
  <directive rdf:resource="&filter;Descendants"/>
</ServiceLevelFilter>
```

Snapshot Filter

The snapshot filter tells DET which snapshot you want to export from an SA core/mesh so that you can then import it into another SA core/mesh.

table 30 Snapshot 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 31 Snapshot Filter Nested Elements

Element	Description
`snapshotResultId`	The ID of the snapshot to export. Snapshot names are *not* unique such that an ID must be given instead. The ID is shown in the user interface by opening up the Snapshot browser, where it is displayed on the first screen.

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
  <!ENTITY filter "http://www.opsware.com/ns/cbt/0.1/filter#">
]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns="http://www.opsware.com/ns/cbt/0.1/filter#">
<SnapshotResultFilter rdf:ID="srf1">
  <snapshotResultId>20001</snapshotResultId>
</SnapshotResultFilter>
</rdf:RDF>
```

Template Export Filter

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

table 32 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 33 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 33 Template Export Filter Nested Elements (cont'd)

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 34.

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 *read*, *read & write*, *none* or *yes/no* state of each action permission in the Action Permission view of the SA client Administration User Groups view
- The *read*, *read & write*, *none* state of each customer and device group in the Resource Permission View

table 34 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 35 User Groups Export Filter Nested Elements

Element	**Description**
`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 Device Groups tabs currently list all customers and server groups respectively to allow the read, read & write, none state to be set. Only customers and device 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 Level/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 Level/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 Level/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 Level/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 Level/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 `cbt` command imports content to a target SA mesh.

➤ Content import is supported on a forward compatible basis only. That is, you cannot import content from any version of SA into an older version of SA.

The `cbt` command executable is found in the directory:

```
/opt/opsware/cbt/bin
```

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 SA mesh. See the Policy on Importing Content Types on page 37.
- `target_core_config` - a configuration file that tells DET where the various SA components are located, and what identity it should use to access them. Instructions for creating this file are located at Configuring the cbt Command on page 49.
- --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 The cbt Command Reference on page 55 for a complete list all the available arguments and their meanings.

When Applications are imported using DET, the associated package name in the SA 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 The cbt Command Reference on page 55 for a complete list of all the available arguments and their meanings.

table 36 Policies Used By DET When Importing Each Content Type

Content Type	Associated Content Types	Import Policy (Overwrite)	Import Policy (Duplicate)
Application Configuration	Application Configuration Template	All attributes are updated in overwrite mode.	New Application Configuration is created and named “Oldname-cbt<random>"
Application Configuration Template		All attributes are updated in overwrite mode.	New Application Configuration template is created and named “Oldname-cbt <random>"
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.
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 46 for important information about importing customers.

table 36 Policies Used By DET When Importing Each Content Type (cont'd)

Content Type	Associated Content Types	Import Policy (Overwrite)	Import Policy (Duplicate)
Folder	• Package • Software Policy • OS Sequence • Customer	Placeholders for all Folders along the path to this folder are created. All attributes of this folder are updated. Folder contents and associated customers are overlaid on existing data. If --folderacls is specified, folderACLs to any pre-existing user groups are created.	Skip: folders are not duplicated.
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.
OS	• Custom attributes • Customer • InstallHooks • MRL	Content information overrides existing node in target SA 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.

table 36 Policies Used By DET When Importing Each Content Type (cont’d)

Content Type	Associated Content Types	Import Policy (Overwrite)	Import Policy (Duplicate)
OS Build Plan	• Application Configuration Template • Custom attribute • Customer • Device Group • Script • Software Policy • ZIP Package	Custom attributes are overlaid. Application Configuration Templates, Scripts, Software Policies, Patch Policies and ZIP Packages are overwritten. OS Build Plan description is updated. If the name of a Build Plan is changed or the Build Plan is moved to a different location after export, the exported OS Build Plan will be imported to the original location as a new OS Build Plan leaving the renamed/relocated OS Build Plan unchanged.	Skip: OS Build Plans are not duplicated. OS Build Plan content is duplicated as follows: • Application Configuration Templates, Scripts and Server Groups are duplicated • Zip Packages are overwritten • Customers are skipped • New Software Policies are created with name `<software_policy>_cbt<random>`
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. If the package already exists in a different folder, it is imported as a new package in the new folder -- the existing package is not moved to the new folder.	Same as overwrite.

table 36 Policies Used By DET When Importing Each Content Type (cont'd)

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 Server Automation 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 Server Automation, overwriting the existing database, if there is one. The knowledge created by the import will depend on the patch preference settings in the target SA mesh.	Same as overwrite.
Patch Policy	Patch	Description and list of patches are updated.	New patch policy is created and named "Oldname-cbt<random>"
Scripts[a]	NA	Adds the current version of the imported script to the existing object.	Duplicates the existing object with a new name and adds the current version of the imported script to the duplicate.
Server Compliance Criteria	NA	All attributes are updated in overwrite mode.	New Server Compliance Criteria is created and named "Oldname-cbt<random>"
Server (Device) Group	• Application • Software Policy • 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>".

table 36 Policies Used By DET When Importing Each Content Type (cont'd)

Content Type	Associated Content Types	Import Policy (Overwrite)	Import Policy (Duplicate)
Service Level	• Custom attributes • Customer	Content information overrides existing node in target SA 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.
Template	• Custom attributes • Customer • Application • Patch • OS • Service Level	Content information overrides existing node in target SA 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.
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 SA 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 (Device) 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>".

a. Scripts can only be imported by importing the parent folder.

Import Delete Conditions

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 SA 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 37 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.

Renamed Objects That Cannot Be Found

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 (Device) Group, Service Level, Template.

table 37 Condition for Content Items to Be Deleted Upon Import with the -del Option

object type	Conditions allowing delete
Application Configuration	In use by zero servers or device Groups. In use by zero software policies.
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 37 Condition for Content Items to Be Deleted Upon Import with the -del Option (cont'd)

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. Zero folders. Note: A Customer cannot be deleted if it has any packages still in SA, including those with the status DELETED. When an object has a DELETED status, it means that either a) the package is still needed for remediation operations on at least one server, or b) the Satellite Software Repository Cache 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.
Folder	Zero contained packages, software policies, and sub-folders.
OS	Zero attached devices. Zero child nodes. Zero templates or device Groups include this node.
Package	Is a deletable unit type (see below) Zero Solaris patch clusters or MRLs use this package. Zero software policies use this package. If a ZIP package, it has zero child relocatable ZIPs. Zero OS definitions or application nodes use this package. Zero software policies use this package. If a patch: — Zero devices attached to the patch node. — Zero templates or device Groups include the patch node. — Zero patch policies or patch exceptions include the patch node. If an LPP, HPUX depot, or Solaris package: — Zero sub-packages in use by software policies. — 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 device Groups include the patch node. – Zero patch policies or patch exceptions include the patch node. Deletable package unit types* (see list following this table)
Patch Policy	Zero attached devices. Zero attached device groups.

table 37 Condition for Content Items to Be Deleted Upon Import with the -del Option (cont'd)

object type	Conditions allowing delete
Server (Device) 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 device Groups include this node.
Template	Zero child nodes.
User Group	Always allow delete.

* Detectable 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

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 SA Client to re-assign the permissions. Until doing so, only users who are administrators will see these customers and their associated objects.

Synchronizing Multimaster Meshes with Deltas

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 42.

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:

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

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

- Import exported content into another mesh:

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

  Content is changed and deleted in source mesh.

- 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
  ```

- 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
  ```

- 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.

Content Directory

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

- `data.rdf` - a database of exported SA 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
```

Example Session

The cbt command is pre-configured to be executed as the root user on a managed server. If used in this configuration, you will only have to provide your SA 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. For more information, see Configuring the cbt Command on page 49.

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

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

The cbt command is found in the directory:

```
/opt/opsware/cbt/bin
```

Installing the cbt Command

The cbt command comes installed and ready to use on your SA core servers.

If you want to use the cbt command on any other Unix server, you need to install it manually, as described in this section. Cbt can also be installed on any Unix-based managed server automatically by creating a software policy.

➤ The cbt command can be run on any Unix computer with network access to an SA mesh. Although the cbt command is not supported on the Windows platform, it does support import and export of Windows content.

To install the cbt command on a Unix server other than the SA Core:

1 Log on as root to the Unix server where you want to install the cbt command. The server must have access to an SA mesh.

2 You can obtain the cbt archive, `cbt-<version>.zip`, via the HPSA installation media or on the SA Core:

a On the SA Core, find the cbt archive in the SA Library under: Opsware/Tools/CBT

b On the HP Server Automationinstallation media, find the `cbt-<version>.zip` file in the `packages` subdirectory.

3 Copy the cbt archive to the directory where you want to install it.

4 Unzip the archive.

5 Configure your Java Runtime Environment (JRE):

a If you do not already have them, download JRE 1.4.x or JDK 1.6.x or later from `www.oracle.com`, and install it on the server where you have logged in.

b Set your JAVA_HOME environment variable to point to your Java installation. For example, in csh you would issue the following command:

```
% setenv JAVA_HOME <java installation>
```

c Optionally, you can set the PATH environment variable to include the `cbt` install directory: `<cbt_install_directory>/bin`.

6 To verify that you can run the `cbt` command, enter the following commands:

```
% cd <cbt_install_directory>/bin
% ./cbt -v
```

The `-v` option displays the command's version string.

On some servers, the `cbt` command displays the following error:

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

If this error occurs, edit the `cbt` script, changing the value of the `-Xmx` option in `jargs` to a lower value, for example: `-Xmx512m`.

Configuring the cbt Command

This explains how to configure the `cbt` command. The cbt command is installed on your SA core servers in the following directory:

```
/opt/opsware/cbt/bin
```

To run the cbt command outside the SA Core, see Running the cbt Command on a UNIX Host that Is Not an SA Core on page 50.

The `cbt` command can be run on any Unix computer with network access to an SA mesh. Although the `cbt` command is not supported on the Windows platform, it does support import and export of Windows content.

The `cbt` command is written in Java and uses OWL and RDF for its schema definition and persistent store. It imports and exports SA content by using SA APIs to extract both configuration and large binary content, such as packages and scripts.

1 To verify that you can run the `cbt` command, enter the following commands:

```
% cd <cbt_install_directory>/bin
% ./cbt -v
```

The -v option displays the command's version string.

On some servers, the cbt command displays the following error:

```
Error occurred during initialization of VM
Could not reserve enough space for object heap
```

If this error occurs, then edit the cbt script, changing the value of the -Xmx option in jargs to a lower value, for example: -Xmx512m.

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

a Obtain a copy of the opsware-ca.crt trust certificate from
/var/opt/opsware/crypto/twist/opsware-ca.crt
and save it in a location that the cbt command can access. This step is optional if you are running cbt from the server where the SA Command Center core component is installed.

b Obtain a copy of the spog.pkcs8 client certificate from
/var/opt/opsware/crypto/twist/spog.pkcs8
and save it in a location the cbt command can access. This step is optional if you are running cbt from the server where the SA Command Center core component is installed.

c Obtain the Web Services Data Access Engine (twist) user name and password from your SA administrator. This is set during the Web Services Data Access Engine (twist) installation.

d Create a target mesh configuration file that contains the location and identity information required to access the SA mesh components. For details on this task, see the following section.

Running the cbt Command on a UNIX Host that Is Not an SA Core

When running the cbt command from an UNIX host which is not an SA core, the NSS_FIPS_ENABLED environment variable needs to be set. If the SA core is running in FIPS enabled mode, the NSS_FIPS_ENABLED environment variable should be set to 1.

An example of an cbt command run from an UNIX host:

```
export NSS_FIPS_ENABLED=1 && <cbt_install_directory>/bin/cbt -i <content_dir>
-p <policy> -cf <target_core_config>
```

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 SA 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.

➤ Because the configuration file contains user names and passwords, make sure it is secure.

Table 38 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 38 assume that you are running DET on an SA mesh running the Command Center core component. (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 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 38 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 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 SA 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.
`spike.username`	`admin`	User name for Spike authentication. This is the user who was granted permissions by the `cbtperm` tool. This user name needs to be an admin account that has permissions to create or modify objects. 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.

table 38 Configuration Properties (cont'd)

Property Name	Default Value	Description
spin.protocol	http	Data Access Engine's listener protocol. HTTP if the DET is on the same server as the SA Command Center and is running a clear text spin in a multi-server mesh or HTTPS for any other configuration.
ssl.keyPairs	/var/opt/opsware/crypto/twist/spog.pkcs8	Comma-separated list of client certificates used to communicate with XML-RPC-based servers.
ssl.trustCerts	/var/opt/opsware/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 HTTP Client library instead of JDK's built-in HTTP client.
twist.certPaths	/var/opt/opsware/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 SA 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.
twist.username	detuser	Web Services Data Access Engine's user name. 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. As of SA 7.80, the Software Repository is part of the Slice Component bundle.

table 38 Configuration Properties (cont'd)

Property Name	Default Value	Description
word.path	wordbot-new.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 host name.
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
```

3 The cbt Command Reference

This describes the cbt command and its options. The cbt command is found on SA core servers in the following directory:

```
/opt/opsware/cbt/bin
```

Export Option (-e)

The export option uses the following syntax:

```
cbt -e <content_dir> [<options>]
```

table 39 Export Options

Short Option	long option	description
-e <content_dir>	--export <content_dir>	Export SA data from an SA core and store the data in the given content directory.
-f <filter_file>	--filter <filter_file>	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 14.
-b <content_dir>	--baseline <content_dir>	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 39 Export Options (cont'd)

Short Option	long option	description
`-cf <file>`	`--config <file>`	Specifies the DET configuration file. For more information, see Creating a Target Mesh Configuration File on page 50.
`-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.
`-lx`	`--logxml`	Create log file in XML format.
`-em <addrs>`	`--email <addrs>`	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.

Import Option (-i)

The import option uses the following syntax:

```
cbt -i <content_dir> [<options>]
```

table 40 Import Options

Short Option	long option	description
`-i <content_dir>`	`--import <content_dir>`	Import SA data from the given content directory.
`-p overwrite|duplicate|skip`	`--policy overwrite|duplicate|skip`	Import policy. Default is “overwrite.” “overwrite” means to override objects in the same name space on the target Server Automation 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 Server Automation. “skip” is the most conservative policy. It aborts the import of an object if the same object is detected in the target Server Automation. For more information on import policies for the specific content types, see Policy on Importing Content Types on page 37.
`-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 40 Import Options (cont'd)

Short Option	long option	description
`-fa`	`--folderacls`	Associate imported folders with existing user groups. If this option is not specified, import the folders with the ACLs inherited from the parent folder in the destination mesh. If this option is specified, the DET will attempt to import the ACLs when importing the folder. An ACL will import only if a user group with the same name as in the source mesh already exists or has been imported as part of the current DET run. The ACLs will become associated with the existing user group of the same name. When inserting a folder, the imported ACLs with replace any inherited from a parent folder in the destination mesh. When updating a folder, the ACLs will overlay existing ACLs.
`-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 <file>`	`--config <file>`	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). The utility reports that the package is overwritten, but the package is untouched. Only its unit record is updated.
`-lx`	`--logxml`	Create log file in XML format.

table 40 Import Options (cont'd)

Short Option	long option	description
`-em <addrs>`	`--email <addrs>`	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 Option (-t)

The show export status option uses the following syntax:

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

table 41 Show Export Status Options

short option	Long option	description
`-t`	`--showstatus`	Show status of export of the given content directory.

Configuration File Option (-s)

The configuration file option uses the following syntax:

```
cbt -s [-cf <file>]
```

table 42 Configuration File Options

short option	Long option	description
`-s`	`--showconfig`	Show current configuration values.
`-cf <file>`	`--confi <file>`	Read configuration from the given file.

Show Version Option (-v)

The show version option uses the following syntax:

```
cbt -v
```

table 43 Show Version Options

short option	Long option	description
-v	--version	Show the version of the DET tool.

Show Help Option (-h)

The show help option uses the following options:

```
cbt -h
```

table 44 Show Help Options

short option	Long option	description
-h	--help	Display this help message.

DET Permissions Command, cbtperm

The `cbtperm` command lets you set permissions for using DET. The `cbtperm` 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 45 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 SA Administrator.
-p	--spike.port	Spike's listener port.
-s	--spike.host	Spike's hostname or IP.
-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

4 IDK Overview

Overview of the IDK and ISMs

Server Automation includes the Intelligent Software Module (ISM) Development Kit (IDK). The IDK consists of command-line tools and libraries for creating, building, and uploading ISMs. An ISM is a set of files and directories that include application bits, installation scripts, and control scripts. You build an ISM in a local file system and then upload the ISM into an Server Automation application policy. After uploading the ISM, you use the HP Server Automation Client to install the ISM's application onto managed servers.

Benefits of the IDK

The IDK offers the following benefits:

- Encapsulates best practices for managing software products, enabling standards teams to deliver stable and consistent software builds and manage change in complex data center environments.
- Uploads modules into Server Automation, making them immediately available for installation onto managed servers.
- Separates an application's installation and control scripts from the bits to be installed. You can update the scripts without having to re-install the application bits.
- Enables dynamic configuration by querying Server Automation for custom attributes.
- Automatically builds native packages (such as RPMs) from binary archives.
- Support on Unix platforms for building from source code with a common specification format.
- Provides command-line tools for developers and administrators who prefer building packages and writing installation scripts in a shell environment.

IDK Tools and Environment

The IDK includes the following:

- ISMTool - A command-line tool that creates, builds, and uploads ISMs.
- ISMUserTool - A command-line tool that specifies the users allowed to upload ISMs.
- Environment variables - Shell environment variables accessed by the ISMTool.
- Runtime libraries - The Server Automation routines that support the IDK tools.

Supported Package Types

You can use the IDK to create the following types of packages:

- AIX LPP

- HP-UX Depot
- RPM
- Solaris Package
- Windows MSI
- ZIP (Windows and Unix)

Installing the IDK

It is recommended that you install and run the IDK on a managed server (a server running a server agent). For instructions, see Installing the IDK on a Managed Server on page 62. For more information on server agents, see the *SA User Guide: Server Automation*.

You can install the IDK on a core server, but do so with care. The core components share the CRYPTO_PATH environment variable with the IDK tools. If you set the CRYPTO_PATH environment variable incorrectly, the core components might cease to function.

You can install the IDK on an unmanaged server (a server that does not run a core component or an agent), but the functionality of the IDK will be limited. On such a server, you can build ISMs but you cannot upload them to the core unless you set the CRYPTO_PATH environment variable. See CRYPTO_PATH on page 108 for information on this variable. See Installing the IDK on an Unmanaged Server on page 63.

Installing the IDK on a Managed Server

To install the IDK and the ISMTool on a managed server, perform the following steps:

1. Choose a managed server to run the IDK.
2. Verify that the host where you install the IDK runs the same operating system version as the managed servers where the ISM's application will be installed.

   For example, if you are creating ISMs for applications to be installed on Redhat Linux 7.3 managed servers, install the IDK on a Redhat Linux 7.3 system.
3. If you are installing the IDK on a Redhat Linux Application Server, Enterprise Server, or Workstation, then make sure that the `rpm-build` package is already installed. To verify that this package is installed, enter the following command:

   ```
   rpm -qa | grep rpm-build
   ```
4. If you are installing the IDK on a Solaris zone, make sure that the `/usr/local` directory exists and has write access. (This directory might not exist in a sparse root zone.) You can perform this task either with Server Automation or with the following `zonecfg` commands, where *path* is the file system on the global zone:

   ```
   zonecfg:zone-name:fs> add fs
   zonecfg:zone-name:fs> set dir=/usr/local
   zonecfg:zone-name:fs> set special=path
   zonecfg:zone-name:fs> set type=lofs
   ```
5. In the SA Client, search for a software policy with a name that contains “ismtool.”
6. In the list of software policies displayed, right-click the policy for the platform where you will run the IDK, and then select Attach Server.
7. On the Attach Server window, select the managed server where you will run the IDK.

8. Make sure that the Remediate Servers Immediately check box is selected.
9. Click Attach.
10. Unix: In a terminal window, log in as `root` to the host where you are installing the IDK and set the PATH environment variable to include the following value.

    ```
    /usr/local/ismtool/bin
    ```

    (On Windows the PATH is set automatically, but will not take effect until you log in again.)
11. In a terminal window, check the IDK installation by entering the following command:

    ```
    ismtool --myversion
    ```

Installing the IDK on an Unmanaged Server

It is recommended that you install and run the IDK on a managed server (a server running a server agent). For instructions, see Installing the IDK on a Managed Server on page 62.

You can install the IDK on an unmanaged server (a server that does not run a core component or an agent), but the functionality of the IDK will be limited. On such a server, you can build ISMs but you cannot upload them to the core unless you set the CRYPTO_PATH environment variable. See CRYPTO_PATH on page 108 for information on this variable.

To install the IDK and the ISMTool on an unmanaged server, perform the following steps:

1. Choose a managed server to run the IDK.
2. Verify that the host where you install the IDK runs the same operating system version as the managed servers where the ISM's application will be installed.

    For example, if you are creating ISMs for applications to be installed on Redhat Linux 7.3 managed servers, install the IDK on a Redhat Linux 7.3 system.
3. If you are installing the IDK on a Windows managed server, set the CRYPTO_PATH environment variable as described in CRYPTO_PATH on page 108.
4. If you are installing the IDK on a Redhat Linux Application Server, Enterprise Server, or Workstation, then make sure that the `rpm-build` package is already installed. To verify that this package is installed, enter the following command:

    ```
    rpm -qa | grep rpm-build
    ```
5. If you are installing the IDK on a Solaris zone, make sure that the `/usr/local` directory exists and has write access. (This directory might not exist in a sparse root zone.) You can perform this task either with Server Automation or with the following `zonecfg` commands, where *path* is the file system on the global zone:

    ```
    zonecfg:zone-name:fs> add fs
    zonecfg:zone-name:fs> set dir=/usr/local
    zonecfg:zone-name:fs> set special=path
    zonecfg:zone-name:fs> set type=lofs
    ```
6. In the SA Client, search for a software policy with a name that contains "ismtool."
7. Locate and open the policy that matches the target server platform.
8. Select the Policy Items view to see all the packages in the policy.
9. Locate and open the package that matches the OS and architecture of the target server.
10. Download the package to the target server using the **Actions > Export Software** menu.
11. Manually install the package onto the target server.

12 Unix: In a terminal window, log in as `root` to the host where you are installing the IDK and set the PATH environment variable to include the following value.

```
/usr/local/ismtool/bin
```

(On Windows the PATH is set automatically, but will not take effect until you log in again.)

13 In a terminal window, check the IDK installation by entering the following command:

```
ismtool --myversion
```

IDK Quick Start

This section shows how to create, build, and upload a simple ISM. After the upload operation, you can run the SA Client and examine the software policy containing the uploaded ISM.

Perform the following steps in a terminal window of the host where you've installed the IDK. Unless otherwise noted, the commands are the same on Unix and Windows.

1 Unix: Log in as `root` to the server where you installed the IDK.

If you cannot log in as `root`, then log in as another Unix user and set the CRYPTO_PATH environment variable as described in CRYPTO_PATH on page 108.

2 Windows: Open a terminal window and make sure that the CRYPTO_PATH environment variable is set.

3 Grant your user the privilege to upload ISMs by entering the `ismusertool` command, for example:

```
ismusertool --addUser johndoe
```

This command asks you to confirm that you are contacting the core through an agent gateway:

```
Using an agent gateway to reach an Opsware Core.
Is this correct? [y/n]: y
```

Next, the command prompts for the Opsware `admin` user name and password:

```
Enter Opsware Admin Username: admin
Enter admin's Opsware Password:
```

For more information, see ISMUserTool on page 110.

4 Create a new ISM.

For example, to create an ISM named `foo`, enter the following command:

```
ismtool --new foo
```

This command creates a directory named `foo` at the current directory level. The ISM is made up of the contents of the `foo` directory. You'll specify the `foo` ISM in the subsequent `ismtool` commands.

5 Add the application files to the ISM.

One way to add the application files is to copy one or more archives to the `bar` subdirectory. For example, if the application bits are in a file named `mytest.zip`, you might add them to the ISM as follows:

Unix:

```
cp /tmp/mytest.zip foo/bar
```

Windows:

```
copy c:\temp\mytest.zip foo\bar
```

6. Set the path to the software policy that will contain the ISM you upload in a later step.

   Note: You must have Write Objects Within Folder permission to the folder that contains the software policy. Folder permissions are set on the Folder Properties window of the SA Client.

   The following `ismtool` command sets the path to the software policy named `Quote Policy`:

   Unix:

   ```
   ismtool  --opswpath '/My Kit/Service/Quote Policy' foo
   ```

   Windows:

   ```
   ismtool  --opswpath "/My Kit/Service/Quote Policy" foo
   ```

   On Unix you enclose the path in single quotes, but on Windows you use double quotes. For both Unix and Windows, the path contains forward slashes.

7. Build the packages within the ISM by entering the following command:

   ```
   ismtool --build foo
   ```

   This command creates three packages in the `foo/pkg` subdirectory. On a Linux system, these packages are as follows:

   ```
   foo-1.0.0-1.i386.rpm
   foo-ism-1.0.0-1.i386.rpm
   ismruntime-rpm-3.0.0-1.i386.rpm
   ```

   The `foo-1.0.0-1.i386.rpm` package contains the application bits, which in this example were copied to the `foo/bar` subdirectory in step 5. The `foo-ism-1.0.0-1.i386.rpm` package holds the installation hooks and control scripts. (Because this example is simple, it has no control scripts.) The `ismruntime-rpm-3.0.0-1.i386.rpm` package contains the SA shared runtimes that the Server Agent will use when it installs the package on a managed server.

   Note that the package type (RPM) corresponds to the native packaging engine of a Linux System. On Windows, the `--build` command creates following MSI packages in the `foo\pkg` subdirectory:

   ```
   foo-1.0.0-1.msi
   foo-ism-1.0.0-1.msi
   ismruntime-msi-3.0.0-1.msi
   ```

8. Upload the ISM into the software policy by entering the following command:

   ```
   ismtool --upload foo
   ```

   This command generates several prompts. First, it asks you to confirm the core into which you are uploading the ISM:

   ```
   Using the following Opsware Core:

     Data Access Engine : d02      192.168.198.91:1004
     Software Repository: d02      192.168.198.91:1003
     Command Engine     : d02      192.168.198.91:1018

     Is this correct? [y/n]: y
   ```

   Next, the `--upload` command prompts for the Opsware user and password:

   ```
     Enter Opsware Username: johndoe
     Enter johndoe’s Opsware Password:
   .. .
   Success!
   ```

9. In the SA Client, open the software policy and verify that it contains the ISM you uploaded in the preceding step.

Platform Differences

In general, the IDK functions the same on packages from different platforms (operating systems). However, there are a few differences, as explained in the following sections.

Solaris Differences

Solaris package names have a 9 character limit. By convention, the format is a set of capital letters, followed by a set of lower case letters that identify the application. Optionally, the final character may have a special meaning. Note that this format is a convention, not a requirement. Here are some examples of Solaris package names:

```
SPROcc
SPROcmpl
SPROcodmg
SUNWgssx
SUNWgzip
SUNWhea
SUNWhiu8x
SUNWhmd
SUNWhmdu
SUNWhmdx
```

When the ISMTool creates a Solaris package, it must use a package name that is no more than 9 characters in length. The package name constructed by ISMTool begins with `ISM`, followed by the five first characters of the ISM's name, followed by the letter `c` for the control package or a digit `0` for the first part of an application package, `1` for the second part, and so forth. For example, if the ISM name is `foobar`, the package names would be the following:

```
ISMfooba0
ISMfoobac
```

If truncation occurs, ISMTool generates a warning so that the developer can rename the ISM to avoid naming conflicts. To view the package names, use the Solaris `pkginfo` command.

If you upload a Solaris passthru package, the response file is not uploaded. You must manually upload the response file.

Windows Differences

On Windows, when ISMTool creates the application and control Windows Installer (MSI) packages, it encodes the `ProductName` and `ProductVersion` as follows:

```
ProductName:     <name>-<version>
ProductVersion:  0.0.<app|ctl release>
```

The `<name>`, `<version>`, and `<release>` correspond to an ISM's internal information, which can be viewed with the ISMTool’s `--info` command. This encoding scheme is by design and is required for the remediation process to work correctly.

5 IDK Build Environment

ISM File System Structure

The ISMTool `--build` and `--upload` commands operate on the ISM directory, which you create with either the `--unpack` or `--new` commands. The `--unpack` command unzips a file (containing the ISM directory contents) that was previously zipped with `--pack`. The `--new` command initially creates the ISM directory. For example, the following command creates a new directory named `ntp-4.1.2`:

```
ismtool --new ntp-4.1.2
```

This command creates the following subdirectories under the `ntp-4.1.2` directory:

- `bar` - Contains binary archives, the contents of which are used to create the application package.
- `doc` - A location for documentation (HTML) generated automatically during ISM build. You can also create other documentation files in the directory.
- `ism` - Contains all the files needed to create the control package of the ISM. The `ism` directory is where you can edit the default package hooks (pre-install, post-install, pre-uninstall, post-uninstall), as well as add control scripts to i`sm/control`.
- `log` - Holds files which keep track of the output from source transformations (compilation or local installs), output from native packaging engines such as msi, rpm, pkgtrans, swpackage, or an SA upload.
- `pad` - Contains the installation scripts (pre-install, post-install, pre-uninstall, post-uninstall) specified by the ISMTool `--addPkgProp` option.
- `pkg` - Contains the application, control, and shared runtime packages, all of which are generated by `--build`. This subdirectory also contains copies of passthru packages.
- `tmp` - Used as scratch space for ISMTool operations.
- `src` - May optionally contain files that can control the compilation of sources into binary archives.

The following listing shows the contents of the ISM subdirectories after the following command:

```
ismtool --build ntp-4.1.2
```

The output of the source build is in the binary archive directory with the generated name `__ntp-4.1.2_src_ntp.spec.cpio`. The build creates the files in the `log`, `pkg`, and `tmp` subdirectories, in addition to the other files with names beginning with two underscores.

```
ntp-4.1.2/
            src/
                ntp-4.1.2.tar.gz
                ntp.spec
            bar/
                __ntp-4.1.2_src_ntp.spec.cpio
                __ntp-4.1.2_src_ntp.spec.cpio.meta
            pkg/
               ntp-4.1.2-3.i386.rpm
               ntp-ism-4.1.2-7.i386.rpm
```

```
   ismruntime-rpm-2.0.rpm
log/
    . . .
doc/
    index.html
    index/
        ntp-4.1.2-3.i386.rpm.html
        ntp-ism-4.1.2-7.i386.rpm.html
tmp/
   . . .
ism/
    ism.conf
    bin/
        ismget
        parameters
        platform
        python
    env/
        ism.sh
        ism.py
        ism.pl
    pkg/
        ism_check_install
        ism_post_install
        ism_post_uninstall
        ism_pre_install
        ism_pre_uninstall
    control/
pad/
    ismruntime-rpm-2.0.0.i386.rpm
     . . .
    ntp-4.1.2-3.i386.rpm/
                         pkg.conf
                         scripts/
    ntp-ism-4.1.2-7.i386.rpm/
   . . .
```

Build Process

This section describes the following:

- When to Invoke the –build Command
- Multiple Command-Line Options
- Actions Performed by the –build Command
- Packages Created by the –build Command

When to Invoke the --build Command

You run the ISMTool `--build` command after `--new` and before `--upload`. Whenever you change an ISM with an option, you must invoke `--build` before `--upload` for the change to take effect. For example, if you specify `--opswpath`, you must invoke `--build` for the new software policy path to take effect before you upload the ISM.

Multiple Command-Line Options

You may invoke multiple ISMTool options on the same command-line, or you may invoke the options separately. In the following Unix example, the command changes the native package engine to `rpm3`, the version to `2.0.47b`, the default install user to `root`, and the default install group to `root` for the ISM directory named `apache`:

```
ismtool --pkgengine rpm3 --version 2.0.47b --user root --group root apache
```

The next sequence of commands is equivalent:

```
ismtool --pkgengine rpm3     apache
ismtool --version   2.0.47b  apache
ismtool --user      root     apache
ismtool --group     root     apache
```

The ISMTool sorts command actions into the proper logical order for execution. The following command, for example, will change the version of `apache` to `3.0` before the build is executed.

```
ismtool --build --version 3.0 apache
```

Actions Performed by the --build Command

The ISMTool `--build` command performs the following steps.

1. Performs a pre-build clean by removing all side-effect build products. However, this step will leave any `cpio` archives generated during a previous build as a form of build cache.
2. Runs the optional script `ism/build/ism_clean`. The scripts in the `ism/build` subdirectory are hooks into the build process. To use these scripts, you must create them manually.
3. Runs a checksum on the application sources and increment the application release number if the current checksum does not match the previous checksum.
4. Runs a checksum on the control sources (the contents of the `ism` subdirectory) and increment the control release number if the current checksum does not match the previous checksum.
5. Runs the optional script `ism/build/ism_pre`.
6. For source builds, recursively searches for `.spec` files in the `src` subdirectory, compiling and executing each.
7. Creates the shared runtime package.
8. Creates the control package.
9. Creates the application package.
10. Generates the automatic HTML document `doc/index/index.html`.
11. Runs the optional script `ism/buid/ism-post`.

Packages Created by the --build Command

The `--build` command creates the following packages in the `pkg` subdirectory:

- Application package - Created from the contents of the `bar` (binary archive) subdirectory, this package contains the application bits. You copy the application archives to the `bar` subdirectory before invoking the `--build` command. The file name of the application package has the following syntax. The `<version>` is for the entire ISM, and the `<release>` is specific to the application package. See ISM Name, Version Number, and Release Number on page 74 for more information.

  ```
  <name>-<version>-<release>.<package-extension>
  ```

- Control package - This package contains the control and installation scripts from the `ism` subdirectory. The control package file name has the following syntax:

  ```
  <name>-ism-<version>-<release>.<package-extension>
  ```

- Shared runtime package - This package holds the shared runtime routines that are invoked by the Server Agent (during installation) and by any control scripts. These runtime routines are for Server Automation, not for the application itself. The file name of the shared runtime package has the following syntax. (The `<ctl-prefix>` is included in the file name only if you've specified a non-default value with the `--ctlprefix` option.)

  ```
  ismruntime-<ctl-prefix>-<package-type>-<idk-version>.<package-extension>
  ```

- Passthru packages - You specify these packages with the `--addPassthruPkg` option, which copies them into the `pkg` subdirectory unchanged.

Specifying the Application Files of an ISM

This section discusses the methods for getting application files into an ISM:

- Placing Archives in the bar Subdirectory
- Specifying Passthru Packages
- Compiling Source (Unix Only)

Placing Archives in the bar Subdirectory

Before running `--build`, you may manually copy file archives to the ISM's `bar` (binary archive) subdirectory. Alternatively, the archives in the `bar` subdirectory may be generated as `cpio` files by the directives in the `%files` section of the specfile. See also Compiling Source (Unix Only) on page 71.

The `--build` command repackages the archives in the `bar` subdirectory into the application package of the `pkg` subdirectory. The following table lists the types of archives that may reside in the `bar` subdirectory.

table 46 Valid Binary Archive Types

File Extension	Archive Type
`.cpio`	Unix CPIO Archive
`.msi`	Microsoft Installer
`.rpm`	RPM Package Manager

table 46 Valid Binary Archive Types

File Extension	Archive Type
.tar	Tape Archive
.tar.bz2	bzip2 compressed Tape Archive
.tar.gz	gzip compressed Tape Archive
.zip	Info-Zip compatible Zip

Specifying Passthru Packages

Unlike an archive in the bar subdirectory, a passthru package is not extracted and re-packaged. The --addPassthruPkg command copies a passthru package unchanged into the pkg subdirectory. The package specified by --addPassthruPkg cannot reside in the ISM directory. The following example adds a passthru package to an ISM and designates the package for addition to the software policy:

```
ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

Compiling Source (Unix Only)

The --build command recursively searches the src subdirectory for specfiles (files ending in .spec)). If found, a specfile is compiled into Bourne Shell and executed. Specfiles are written in a simplified derivative of the RPM specfile language. The ISMTool's specfile-like language compiler allows you to use existing RPM specfiles with minimal modifications.

For more information about the specfile language, see the Maximum RPM document, located at the following URL:

```
http://www.rpm.org/max-rpm/index.html
```

Example Specfile

Here is an example of a simple ISM specfile for NTP 4.1.2:

```
##############################################
# Common Preamble
##############################################

%define ismname %(../ism/bin/ismget name)
%define version %(../ism/bin/ismget version)
%define prefix %(../ism/bin/ismget prefix)

Name: %{ismname}
Version: %{version}

##############################################
# prep, build, install, files
##############################################

Source: http://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.1.2.tar.gz

%prep
```

```
%setup -n ntp-4.1.2

%build

%ifos Solaris2.7
echo ``do something Solaris2.7 specific''
%endif

%ifos Linux
echo ``do something Linux specific''
%endif

./configure --prefix=%prefix
make

%install
/bin/rm -rf $ISM_BUILD_ROOT
make install prefix=$ISM_BUILD_ROOT/%{prefix}

%files
%defattr(-,root,root)
%prefix
```

Specfile Preamble

The preamble specifies information to be fetched from the ISM with the program `ismget`. The following lines fetch the name, version, and prefix of the ISM.

```
%define ismname    %(../ism/bin/ismget name)
%define version    %(../ism/bin/ismget version)
%define prefix     %(../ism/bin/ismget prefix)
```

This fetched information can be useful in the set up and compilation of sources. However, the `%define` commands are optional. The only required tags in the preamble are `Name` and `Version`.

%prep

The `%prep` section is designed to prepare sources for compilation. This involves uncompressing and untaring source distributions. A single source file is identified with the `Source` tag. A list of sources are identified by a vector of tags: `Source0`, `Source1`, ... Similarly, patches are identified by either a `Patch` tag or a vector of tags: `Patch0`, `Patch1`, ... The ISMTool duplicates the macro functionality as documented in Maximum RPM. The `%setup` macro controls how sources are unpacked. The %prep section can also manage patching using the %patch macro.

%build

The shell script commands in the `%build` section will transform the sources into binaries. Compiling from source usually involves running `./configure -prefix=%{prefix}` and `make`. It is possible to perform configuration switching based on the platform (operating system). The platform tags are designed for backward compatibility to RPMs found in real-world installations. The following platform strings are some examples that can be used in ISMTool specfiles for platform branching:

```
Linux
RedHat
RedHat-Linux-7.2
```

```
RedHat-Linux-AS2.1
Solaris
Solaris2.8
Solaris-2.8
SunOS
SunOS5.7
SunOS-5.7
hpux
hpux11.00
hpux-11.00
HPUX
HPUX11.00
HPUX-11.00
aix
aix4.3
aix-4.3
AIX
AIX4.3
AIX-4.3
```

%install

The `%install` section specifies the copying of files from the build to a virtual install location. For example, if the `%prefix` is set to `/usr/local`, the following line would install NTP into `/usr/local/bin`:

```
make install prefix=$ISM_BUILD_ROOT/%{prefix}
```

The variable `$ISM_BUILD_ROOT` (or equivalently `$RPM_BUILD_ROOT`) is the location of a temporary directory inside the ISM's `tmp` directory. This temporary directory will serve as the virtual install root where the directives in the `%files` section will be applied.

The `%install` section also indicates where the files from a binary install could be extracted. In a binary install, the files resulting from a binary install on a development server can be packaged into the virtual install location. However, if that is not possible then a binary installer could be transported to the end system and installed with an ISM post-install hook. In this case, you would create a binary archive of the installer and copy it to the ISM's `bar` subdirectory.

%files

In the specfile, the output of the source transformation phase is a set of files indicated by the directives in the `%files` section. These files are archived into a `cpio` in the ISM’s `bar` subdirectory.

The final phase of the source transformation is to select the files installed into the `$ISM_BUILD_ROOT`. The directives in the `%files` section are a subset of the selection mechanisms documented in Maximum RPM. These directives specify a list of files or directories (which are recursively gathered) relative to `$ISM_BUILD_ROOT`. In this example, the install is into the path `$ISM_BUILD_ROOT/%{prefix}`. To select these files for packaging, you would simply give the `%prefix` as the directory to package.

In addition to selecting files by naming files or directories, meta information can be described. The line `%defattr(-,root,root)` tells the archive engine to use the modes it finds in the file system, but to create the archive replacing the file ownerships it finds in the file system with `root,root`. For full documentation of `%defattr()` and `%attr()`, see Maximum RPM.

ISM Name, Version Number, and Release Number

This section includes the following:

- Initial Values for the ISM Name, Version, and Release
- ISM Version and Release Numbers Compared
- Upgrading the ISM Version

Initial Values for the ISM Name, Version, and Release

The `--new` command creates a directory for the new ISM and specifies the internal base name of the ISM. For example, the following command creates the `mystuff` directory in the file system, sets the internal base name to `mystuff`, and sets the version number to 1.0.0.

```
ismtool --new mystuff
```

In most cases, you specify the version number with `--new`. The following command creates a directory named `ntp-1.4.2`, sets the internal base name to `ntp`, and sets the version number to 1.4.2:

```
ismtool --new ntp-1.4.2
```

To view the internal base name, version number, and release numbers, use the `--info` command:

```
ismtool --info ntp-1.4.2.
```

The output generated by the preceding command includes the following:

```
. . .
name:                    ntp
version:                 4.2.1
appRelease:              0
. . .
ctlRelease:              0
. . .
```

ISM Version and Release Numbers Compared

ISM version and release numbers differ in several ways. You may specify the version number with either the `--new` or `--version` commands. The ISMTool automatically generates the release numbers; you cannot specify them. The version number applies to the entire ISM. The application and control packages each have separate release numbers. The `--build` command increments the release numbers whenever it re-generates the packages. Because application and control packages can be built independently, the packages may have different release numbers.

The names of the application and control packages include the internal base name, version number, and release number. For example, the `ntp-4.1.2-3.i386.rpm` application package has a version number of 4.1.2 and a release number of 3. See also Packages Created by the -build Command on page 70.

To display the version of the IDK (not the ISM), enter the following:

```
ismtool --myversion
```

Upgrading the ISM Version

Although you may modify the internal base name (with `--name`) and the version number (with `--version`), this practice is not recommended because it does not automatically change the directory name. If you change the internal base name or version, to avoid confusion you should also rename the directory containing the ISM.

The recommended practice is to use a matching internal base name, version number, directory name, and software policy path. For example, to upgrade `foo-1.2.7` to `foo-1.2.8`, you would follow these steps:

1. At the same directory level as `foo-1.2.7`, create a new ISM directory:

   ```
   ismtool --new foo-1.2.8
   ```

2. Copy archives to the `foo-1.2.8/bar` directory or specify passthru packages.

3. Set the path to the software policy at the same level as the previous version.

   Unix:

   ```
   ismtool --opswpath ‘MyFolder/{$NAME}/{$VERSION}’
   ```

   Windows:

   ```
   ismtool --opswpath "MyFolder/{$NAME}/{$VERSION}"
   ```

   The `--opswpath` command replaces the `NAME` variable with `foo` and the `VERSION` variable with 1.2.8. To see the current values of the variables, use the `--info` command. For more information on variable substitution, see ISMTool Variables on page 100.

4. Build and upload the `foo-1.2.8` ISM with the ISMTool.

5. In the SA Client, detach the `foo-1.2.7` policy from the managed servers.

6. (Optional) Remove the `foo-1.2.7` policy.

7. Remediate managed servers against the new software policy.

6 IDK Scripts

Overview of ISM Scripts

ISM scripts are Unix shell or Windows command-line scripts that reside in the ISM directory. The sections that follow describe the different type of ISM scripts:

- **Installation Hooks:** Bundled into the ISM's control package by the ISMTool `--build` command, the installation hooks are run by the native packaging engine (such as `rpm`) on the managed server. Installation hooks may invoke control scripts.
- **Control Hooks**: Also bundled into the ISM's control package, the control scripts perform day-to-day, application-specific tasks such as starting software servers.
- **Installation Scripts**: Not contained in the control package, but instead stored in the Software Repository, installation scripts can be viewed on the Properties of a package in the SA Client.

The overall process for developing and running installation hooks and control scripts follows:

1. invoke the ISMTool `--new` command, which creates the default installation hooks.
2. With a text editor, create the control scripts.
3. With a text editor, modify the default installation hooks, which may call control scripts.
4. With the ISMTool, build and upload the ISM.
5. In the SA Client, install the application contained in the ISM onto a managed server. During the installation, the pre-installation and post-installation hooks are run on the managed server.
6. During the production lifetime of the application, run or schedule the control scripts.
7. At the end of the application's life cycle, with the SA Client, uninstall the application. During the uninstallation, the pre-uninstallation and post-uninstallation hooks are executed on the managed server.

Installation scripts have a different overall process than installation hooks and control scripts. For more information, see Installation Scripts on page 87.

An ISM script cannot call program (such as `rpm` or `pkgadd`) that locks the package associated with the script.

Installation Hooks

The installation hooks are scripts that reside in the `ism/pkg` subdirectory. (Some documents refer to the installation hooks as "packaging scripts.") The installation hooks are run at certain stages during the installation and uninstallation of applications on managed servers.

Creating Installation Hooks

The ISMTool --new command creates the following installation hooks:

Unix:

```
ism/pkg/
       ism_check_install
       ism_post_install
       ism_post_uninstall
       ism_pre_install
       ism_pre_uninstall
```

Windows:

```
ism\pkg\
       ism_post_install.cmd
       ism_post_uninstall.cmd
       ism_pre_install.cmd
       ism_pre_uninstall.cmd
```

To customize the installation hooks, you modify them with a text editor. Although you may edit the installation hooks, you cannot change their file names.

The default ism_pre_install and ism_post_uninstall hooks are just stubs; they perform no actions. The default ism_post_install hook calls the ism_configure and ism_start control scripts. The default ism_pre_uninstall hook calls the ism_stop control script. Note that the control scripts are not created automatically by the ISMTool; you must create them with a text editor. (See Control Scripts on page 82.)

For the contents of the default installation hooks created by the --build command, see the following sections:

- Default Installation Hooks for Unix on page 80
- Default Installation Hooks for Windows on page 81

Check Installation Hook

Some native packaging engines support the ism_check_install hook directly; others do so implicitly with the ism_pre_install hook. The ISMTool maps the check_install feature onto the native packaging engine. If the check_install script returns a non-zero code, the install is halted.

Invocation of Installation Hooks

When you install (or uninstall) the application of an ISM onto a managed server, the native packaging engine on the server invokes the installation hooks. (You do not run the installation hooks directly.) For example, on a Linux system, the rpm utility invokes ism_pre_install immediately before it installs the application bits and invokes ism_post_uninstall right after it removes the bits.

See also Invocation of Installation Scripts and Hooks on page 88.

Installation Hooks and ZIP Packages

Unlike some other packaging engines, the ZIP packaging engine used by Server Automation does not support installation hooks. If the ZIP packaging engine is specified and the installation hook files are not empty, the ISMTool generates a warning and ignores the installation hook files.

ZIP Packages and Installation Directories

The ZIP packages created by the IDK are not relocatable. In other words, the same ZIP package cannot be used to install multiple instances of an application in different directories on a single managed server. Therefore, if the end user changes the ZIP package's Install Path field in the SA Client, the package installation will fail. To change the installation directory, the ISM developer specifies a new path with the ––prefix or ––ctlprefix option, builds a new ISM, and uploads the new ISM to the core. (For Windows NT4, these options are required and cannot specify variables.)

As a best practice for ZIP packages, the ISM developer should include a warning in the ISM's description similar to the following: "WARNING: Do not change the Install Path of this package."

Installation Hook Functions

You can customize the installation hooks to perform actions such as those listed in the following table.

table 47 Installation Hook Functions

Install hook	common functions
ism_pre_install	create required directories, create users, set directory permissions
ism_post_install	call ism_configure control script, call ism_start control script (to start a web server, for example)
ism_pre_uninstall	call ism_stop control script (to stop a server)
ism_post_uninstall	do any required clean up

Scripts for Control-Only ISMs

If you specify the ––skipApplicationPkg option, the ISMTool will not build the application package, enabling the creation of a control-only ISM. You can use this feature to build a controller for an application that is not installed or packaged with the ISMTool. Examples are controllers for core operating system functions, currently running applications that cannot be packaged, and specialized hardware.

During the installation and uninstallation of a control-only ISM, the ism_ctl_post_install and ism_ctl_pre_uninstall scripts are run. (The scripts are run for all ISMs, but typically you specify them only for control-only ISMs.) Because these scripts are not generated by the ISMTool, you must create them before running the ––build command. The following listing shows the required names and locations of these scripts:

Unix:

```
ism/pkg/
        . . .
        ism_ctl_post_install
        ism_ctl_pre_uninstall
```

Windows:

```
ism\pkg\
        . . .
        ism_ctl_post_install.cmd
        ism_ctl_pre_uninstall.cmd
```

Location of Installation Hooks on Managed Servers

On your development system, the `--build` command bundles the installation hooks into the ISM's control package. On the managed server, the contents of the control package are installed into the directory indicated by the `ctlprefix` of the ISM. By default, the installation hooks are installed into the following directory:

Unix:

```
/var/opt/OPSWism/<ism-name>/pkg
```

Windows:

```
%ProgramFiles%\OPSWism\<ism-name>\pkg
```

To change the default directory of the installation hooks, specify the `--ctlprefix` option before building and uploading the ISM. If you specify the `ctlprefix` as follows, for example, the installation hooks will be installed in `/usr/local/ntp-4.1.2/pkg`:

```
ismtool --ctlprefix /usr/local ntp-4.1.2
```

Default Installation Hooks for Unix

The default `ism_pre_install` hook:

```
#!/bin/sh
#
# ISM Pre Install Script
#
. 'dirname $0'/../env/ism.sh
```

The default `ism_post_install` hook:

```
#!/bin/sh
#
# ISM Post Install Script
#
. 'dirname $0'/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_configure ]; then
${ISMDIR}/control/ism_configure
fi
if [ -x ${ISMDIR}/control/ism_start ]; then
${ISMDIR}/control/ism_start
fi
```

The default `ism_pre_uninstall` hook:

```
#!/bin/sh
#
# ISM Pre Uninstall Script
#
. 'dirname $0'/../env/ism.sh
if [ -x ${ISMDIR}/control/ism_stop ]; then
${ISMDIR}/control/ism_stop
fi
```

The default `ism_post_unininstall` hook:

```
#!/bin/sh
#
```

```
# ISM Post Uninstall Script
#
. ‘dirname $0‘/../env/ism.sh
```

Default Installation Hooks for Windows

The default `ism_pre_install.cmd` hook:

```
@echo off
REM
REM ISM Pre Install Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
ENDLOCAL
```

The default `ism_post_install.cmd` hook:

```
@echo off
REM
REM ISM Post Install Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
REM
REM Call the ISM’s configure script
REM
IF EXIST "%ISMDIR%\control\ism_configure.cmd"
call "%ISMDIR%\control\ism_configure.cmd"
REM
REM Call the ISM’s start script
REM


IF EXIST "%ISMDIR%\control\ism_start.cmd"
call "%ISMDIR%\control\ism_start.cmd"
ENDLOCAL
```

The default `ism_pre_uninstall.cmd` hook:

```
@echo off
REM
REM ISM Pre Uninstall Hook
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
```

```
call %1
REM
REM Call the ISM’s stop script
REM
IF EXIST "%ISMDIR%\control\ism_stop.cmd"
call "%ISMDIR%\control\ism_stop.cmd"
ENDLOCAL
```

The default `ism_post_unininstall.cmd` hook:

```
@echo off
REM
REM ISM Post Uninstall Script
REM
SETLOCAL
REM
REM %1 specifies the full path to the ISM.CMD file
REM Call ISM.CMD to define ISM environment variables
REM
call %1
```

Control Scripts

The ISM control scripts reside in the `ism/control` directory. Control scripts perform housekeeping or maintenance tasks for an application after it has been installed.

Installation hooks can run control scripts. If a task is performed during an installation (or uninstallation) but might also be performed on a regular basis, it should be coded as a control script. For example, the `ism_post_install` hook can invoke the `ism_start` control script to start an application immediately after installation. Also, the `ism_pre_uninstall` hook can invoke the `ism_stop` control script to shutdown the application.

Users can run control scripts from the ISM Control window of the SA Client. Advanced users can run control scripts from the command line in the Global Shell.

Creating Control Scripts

Unlike installation hooks, control scripts are not created by the ISMTool; you create control scripts with a text editor. You may add any number of control scripts to the `ism/control` subdirectory. By convention, the file names for control scripts are as follows:

Unix:

```
ism/control/
           ism_start
           ism_stop
           ism_configure
           ism_reconfigure
```

Windows:

```
ism\control\
           ism_start.cmd
           ism_stop.cmd
           ism_configure.cmd
```

```
ism_reconfigure.cmd
```

The control script name might appear differently in the ISM Control window of the SA Client. The Action field of the ISM Control window displays the name of the control script, but without the leading `ism_` or the file type extension. For example, a control script named `ism_start.cmd` appears in Action field as `start`. The Action field displays only the first 25 characters of a control script name. Therefore, the first 25 characters of the names should be unique. For both Unix and Windows, the leading `ism_` must be lower case; otherwise, the Action field displays the prefix.

For Unix, make sure that the control scripts under `ism/control` are executable. Otherwise, they will not appear in the SA Client.

Control Script Functions

Control scripts are for repetitive tasks needed to manage an application. The following table summarizes typical uses for control scripts.

table 48 Control Script Functions

Control Script	common functions
`ism_start`	notifies any companion or dependent servers, starts the application
`ism_stop`	notifies any companion or dependent servers, stops the application
`ism_configure`	performs configuration operations
`ism_reconfigure`	similar `to ism_configure`, but calls `ism_stop` first and `ism_start` afterwards

Location of Control Scripts on Managed Servers

Like installation hooks, control scripts are bundled into the control package by the `--build` command. On the managed server, control scripts reside in the directory indicated by the ISM `ctlprefix` value. By default, control scripts are installed in the following directory on a managed server:

Unix:

```
/var/opt/OPSWism/<ism-name>/control
```

Windows (except for NT4):

```
%ProgramFiles%\OPSWism\<ism-name>\control
```

To change the default directory, specify the `--ctlprefix` option with ISMTool. For Windows NT4, `--ctlprefix` must be specified and cannot contain variables.

Dynamic Configuration with ISM Parameters

The ISM `parameter` utility enables control scripts and installation hooks to access the values of SA custom attributes. The key of an ISM parameter matches the name of its corresponding custom attribute. The value of a custom attribute determines the value of the parameter. The source of a custom attribute is an SA object such as a facility, customer, server, or device group.

Set with the SA Client, a custom attribute is a name-value pair that holds configuration information. For example, to designate the port number of an Apache web server, a custom attribute named APACHE_1.3_PORT could have a value of 80. If an ISM has a parameter named APACHE_1.3_PORT, a control script could access the current value of the custom attribute.

Using the ISM Control window of the SA Client, an end-user can view the source (SA object) of a parameter, view the parameter value, and override the parameter value.

Development Process for ISM Parameters

The overall process for developing and using ISM parameters follows:

1 With the ISMTool, add a new parameter.

2 With a text editor, write a control script (or modify an installation hook) to access the parameter.

3 With the ISMTool, build and upload the ISM.

4 In the SA Client, install the application contained in the ISM onto a managed server.

5 In the SA Client, create a custom attribute with the same name as the parameter.

6 In the SA Client, run the control script on the managed server. At runtime, the script retrieves the parameter (control attribute) value from Server Automation.

Adding, Viewing, and Removing ISM Parameters

The ISMTool --addParam command creates a new parameter, which may be fetched by any script in the ISM. A parameter is a tuple with four fields, each specified by an ISMTool option. The following table lists the fields and their corresponding options.

table 49 ISM Parameter Fields

Parameter field	ISMTool option	Description
Name	--addParam	The name of the ISM parameter, which must match the name of the custom attribute.
Default Value	--paramValue	The default value of the parameter. The script uses the default value if a matching custom attribute is not found.
Type	--paramType	The data type of the parameter. Allowed values: 'String' 'Template'
Description	--paramDesc	Text describing the parameter.

The following Unix command adds a parameter named NTP_SERVER to the ntp-4.2.1 ISM:

```
ismtool --addParam NTP_SERVER \
     --paramValue 127.0.0.1 \
     --paramType 'String' \
     --paramDesc 'NTP server, default to loopback' ntp-4.2.1
```

To view the parameters that have been added to the ntp-4.2.1 ISM, enter the following:

```
ismtool --showParams ntp-4.2.1
```

To remove the parameter added in this example, you enter the following command:

```
ismtool --removeParam NTP_SERVER ntp-4.2.1
```

Accessing Parameters in Scripts

After you've added a parameter with ISMTool, you can write an ISM control script to access the parameters. The supported scripting languages follow:

- Bourne Shell
- Korn Shell
- Windows command shell
- Python
- Perl

Shell scripts access the parameters through environment variables, Python scripts through dictionaries, and Perl scripts through hash tables.

The ISM parameters Utility

To fetch parameters, a control script runs the `parameters` utility, which resides in the ISM shared runtime package. Only those parameters defined with the `--addParam` command can be fetched.

The `parameters` utility has the following syntax:

```
parameters [options]
--scope <scope> ; server|servergroup|customer|facility|
                ; servicelevel|os|custapps|webserver|appserver|
                ; dbserver|systemutilities|osextras|install|
                ; default (default is all)
-s/--sh                 ; Bourne Shell syntax
-k/--ksh                ; Korn-Shell syntax
-p/--python             ; Python repr'ed dictionary
-l/--perl               ; PERL map
-c/--cmd                ; Windows Cmd syntax
-b/--vbscript           ; Windows VBScript syntax
-h/--help               ; Help
-v/--version            ; Version
```

The `--scope` option limits the search for the custom attribute to the specified area of Server Automation. For example, if you specify `--scope facility` and a custom attribute has been defined for both the facility and the customer, then the custom attribute of the customer is not considered. See also: Search Order for Custom Attributes on page 86.

If the `parameters` utility encounters an error during retrieval, it returns a special parameter named `_OPSW_ISMERR`, which contains a brief description of the error encountered.

Example Scripts

The following Bourne Shell example is a control script that configures the NTP time service on Unix. The parameters utility retrieves two parameters, `NTP_CONF_TEMPLATE` and `NTP_SERVER`, that have been defined for the ISM.

```
    #!/bin/sh
    . `dirname $0`/../env/ism.sh
```

```
eval `${ISMDIR}/bin/parameters`
echo $NTP_CONF_TEMPLATE | \
sed "s/NTP_SERVER_TAG/$NTP_SERVER/" > /etc/ntp.conf
```

The following control script, written in Python, also configures NTP.

```
#!/usr/bin/env python
import os
import sys
import string
ismdir=os.path.split(sys.argv[0])[0]
cmd = ’%s --python’ % (os.path.join(ismdir,’bin’,’parameters’))
params = eval(os.popen(cmd,’r’).read())
template = params[’NTP_CONF_TEMPLATE’]
value = params[’NTP_SERVER’]
conf = string.replace(template,’NTP_SERVER_TAG’,value)
fd=open(’/etc/ntp.conf’,’w’)
fd.write(conf)
fd.close()
```

The following example shows a configuration control script for Windows. In this example, for 32 bit Windows operating systems, each parameter is output in the form of name=value (one per line).

The Windows `FOR` command sets each parameter as an environment variable. (In the listing that follows, the `FOR` command is split into two lines, but in the actual script, the `FOR` command must be on a single line.) Finally, the parameters are passed to an NTP configuration script named `WindowsNTPConfigureScript.cmd`.

```
@echo off
SETLOCAL
cd /d %ISMDIR%
for /f "delims== tokens=1,2" %%i in ('""bin\parameters.cmd""') do set
%%i=%%j WindowsNTPConfigureScript.cmd %NTP_CONF_TEMPLATE% %NTP_SERVER%
ENDLOCAL
```

Search Order for Custom Attributes

With the SA Client, you can set a custom attribute in several places. For example, you could set a custom attribute named `APACHE_1.3_PORT` to 8085 for a managed server named `foo.hp.com`, and you could set the same custom attribute to 80 for the Widget Corp. customer, which is associated with the `foo.hp.com` server. At runtime, if a control script on f`oo.hp.com` accesses the `APACHE_1.3_PORT` parameter, which value will it fetch? In this case, the value will be 8085 because a custom attribute for a server occurs first in the search order.

Note that if a custom attribute is not found, the script uses the default parameter value that you set with the ISMTool `--paramValue` option.

Default Search Order

The default search order for custom attributes is as follows:

1. Server
2. Device Group
3. Customer
4. Realm

5 Facility

6 Operating system.

7 ISM (created in the software policy during the upload operation)

8 Patch Policy

9 Software Policy

Multiple device groups and service levels are searched alphabetically. For example, if a server belongs to the ABC and XYZ groups, the ABC group is searched for the custom attribute before the XYZ group. A server group that is a subgroup does not inherit the custom attributes of its parent group.

Installation Scripts

The installation scripts reside in the `pad` subdirectory. Like installation hooks, the installation scripts are run at specific stages during the installation and uninstallation of an application on a managed server.

Differences Between Installation Scripts and Hooks

Although they serve a similar purpose, installation scripts and hooks have several differences, as noted in the following table.

table 50 Differences Between Installation Scripts and Hooks

installation scripts	installation hooks
Displayed by the Properties of the package in the SA Client.	Displayed by the Contents of the package in the SA Client. (Only RPMs are displayed.)
Reside in the `pad` subdirectory.	Reside in the `ism/pkg` subdirectory.
Stored in Model Repository (after an upload).	Bundled in the control package, installed on the managed server in the directory specified by `ctlprefix`.
Run by the Server Agent.	Run by the native packaging engine.
Can be defined for each package in the ISM.	Defined for the entire ISM.

Creating Installation Scripts

Although the ISMTool creates the `pad` subdirectory structure, it does not create default installation scripts. For each package created with `--build` or added with `--addPassthruPkg`, the ISMTool creates a subdirectory as follows:

```
pad/<package-name>/scripts
```

For example, on Linux the `--build` command would create the following subdirectories for an ISM named `ntp-1.4.2`:

```
pad/ismruntime-rpm-2.0.0-1.i386.rpm/scripts
pad/ntp-ism-4.2.1-1.i386.rpm/scripts
pad/ntp-4.2.1-1.i386.rpm/scripts
```

With a text editor, you create the installation scripts in the `scripts` subdirectory. For example, you could create installation scripts for the `ntp-4.2.1-1.i386.rpm` package as follows:

```
pad/ntp-4.2.1-1.i386.rpm/scripts/
                                preinstallscript
                                pstinstallscript
                                preuninstallscript
                                pstuninstallscript
```

The file names of the installation scripts must match the preceding example. For example, the script invoked immediately after the installation must be named `pstinstallscript`.

Invocation of Installation Scripts and Hooks

If an ISM has both installation scripts and hooks, when an application is installed on a managed server, Server Automation performs tasks in the following order:

1. Installs the ISM runtime package.
2. Installs the ISM control package.
3. Runs `preinstallscript` (installation script).
4. Runs `ism_pre_install` (installation hook).
5. Installs the application package (the application bits).
6. Runs `ism_post_install` (installation hook).
7. Runs `ism_configure` (control script).
8. Runs `ism_start` (control script).
9. Runs `pstinstallscript` (installation script).

During the uninstallation of an application on a managed server, Server Automation performs actions in the following order:

1. Runs `preuninstallscript` (uninstallation script).
2. Runs `ism_pre_uninstall` (uninstallation hook).
3. Runs `ism_stop` (control script).
4. Uninstall the application package (the application bits).
5. Runs `ism_post_uninstall` (uninstallation hook).
6. Runs `pstuninstallscript` (uninstallation script).
7. Uninstalls the ISM control package.
8. Uninstalls the ISM runtime package.

7 IDK Commands

ISMTool Argument Types

Table 51 defines the argument types that are used in the ISMTool commands defined in the rest of this chapter. The ISMNAME argument type, for example, is specified by the syntax of the ISMTool --new command.

table 51 ISMTool Argument Types

Argument Type	Description	Example
PATH	Absolute file system path.	/foo/bar
STRING	Text string with no spaces.	foobar
TEXT	Arbitrary quoted text. On Unix you enclose the text in single quotes; on Windows use double quotes.	'This is some text'
BOOL	Boolean.	true or false
ISMFILE	Path to a valid .ism file in the file system. This file would unpack into an ISMDIR.	/foo/bar/name.ism
ISMDIR	Path to a valid extracted ISMFILE or to a newly created ISM.	xyz /home/sam/xyz
ISMNAME	Name for a newly-created ISM. The ISMNAME can have the format STRING or STRING-VERSION.	ntp ntp-4.1.2
VERSION	A STRING that represents the version of the ISM. The VERSION cannot contain spaces and must be a legal version string for the native packaging engine.	1.2.3 4.13 0.9.7b
HOST[:PORT]	Host and optional port.	www.foo.com www.foo.com:8000 192.168.1.2:8000
BYTES	Integer number of bytes.	42
SECONDS	Integer number of seconds.	300
PARAMTYPE	Expected type of the parameter data. The only allowed values are the constants ‘String’ and ‘Template’. On Unix you enclose the values in single quotes; on Windows use double quotes.	‘String’ ‘Template’

ISMTool Informational Commands

This section describes the ISMTool commands that provide information about the build environment.

--help

Display the ISMTool command-line help.

--env

Display the locations of system-level tools found in the environment. This command is helpful for investigating build problem and for verifying that the environment variable `ISMTOOLBINPATH` is set correctly. For example, on a Unix system `--env` might display the following:

```
% ismtool --env
bzip2: /usr/local/ismtool/lib/tools/bin/bzip2
cpio: /usr/local/ismtool/lib/tools/bin/cpio
gzip: /usr/local/ismtool/lib/tools/bin/gzip
install: /usr/local/ismtool/lib/tools/bin/install
17
patch: /usr/local/ismtool/lib/tools/bin/patch
python: /usr/local/ismtool/lib/tools/bin/python
pythonlib: /usr/local/ismtool/lib/tools/lib/python1.5
rpm2cpio: /usr/bin/rpm2cpio
rpm: /bin/rpm
rpmbuild: /usr/bin/rpmbuild
tar: /usr/local/ismtool/lib/tools/bin/tar
unzip: /usr/local/ismtool/lib/tools/bin/unzip
wget: /usr/local/ismtool/lib/tools/bin/wget
zip: /usr/local/ismtool/lib/tools/bin/zip
zipinfo: /usr/local/ismtool/lib/tools/bin/zipinfo
pkgengines: ['rpm4']
```

--myversion

Display the version of the ISMTool.

--info ISMDIR

Display an overview of the internal information about the ISM contained in the directory `ISMDIR`. After the build is completed, more detailed information is available, which can be viewed in browser at this URL:

```
<ISMDIR>/doc/index/index.html
```

--showParams ISMDIR

Display the name, default value, type, and description for each control parameter.

--showPkgs ISMNAME

Display the list of all packages managed by the ISM. This list includes the control package, the application package, all passthru packages, and all inner packages contained in passthru packages. Examples of inner packages are Solaris package instances contained in Solaris packages, or an update fileset contained in a AIX LPP package. For each managed package, the package name, type, attached status and all meta data that can be set will be listed.

--showOrder ISMNAME

Display the current install order of attached packages managed by the ISM.

--showPathProps ISMNAME

This option is deprecated in Server Automation 6.0.

Displays the values currently specified for software policy meta data.

ISMTool Creation Commands

This section describes the ISMTool commands that generate the ISM directory structure.

--new ISMNAME

Create a new ISM, which consists of directory that contains subdirectories and files. The value of ISMNAME specifies the name of the newly-created ISM directory. The internal ISM name varies with the format of ISNAME.

For example, the following command creates an ISM directory called foobar. The internal name of the ISM is foobar and the initial version of the ISM defaults to 1.0.0.

```
% ismtool --new foobar
```

The next command creates an ISM directory called ntp-4.1.2. The internal name of the ISM is ntp and the initial version of the ISM is 4.1.2. Note that the internal name of the ISM does not include -VERSION.

```
% ismtool --new ntp-4.1.2
```

The name of the ISM directory is independent of the internal ISM name. For example, if the developer renames the ntp-4.1.2 directory to myntp, the internal name of the ISM is still ntp and the version of the ISM remains 4.1.2.

--pack ISMDIR

Creates a ZIP archive of the ISM contained in ISMDIR. The name of the archive will be <ismname-version>.ism. Note that the contents of ISMDIR must be less than 2GB. (If the size is greater than 2 GB, then use the zip or tar utility instead.) An example of --pack follows:

Unix:

```
% ismtool --new tick
% ismtool --version 3.14 tick
% ls
tick/
% mv tick spooon
% ls
spooon/
% ismtool --pack spooon
% ls
spooon/ tick-3.14.ism
```

Windows:

```
% ismtool --new tick
% ismtool --version 3.14 tick
% dir
11/21/2003 10:17a <DIR> tick
% move tick spoon
% dir
11/21/2003 10:17a <DIR> spoon
% ismtool --pack spoon
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

--unpack ISMFILE

Unpacks the ISM contained in the ZIP file named ISMFILE. The ISM is unpacked into the `ISMDIR` that was specified when the `ISMFILE` was created with the `--pack` command. The following example uses the `ISMFILE` created in the `--pack` example:

Unix:

```
% ls
spooon/ tick-3.14.ism
% rm -rf spooon
% ls
tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% ls
spooon/ tick-3.14.ism
```

Windows:

```
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% rmdir /s /q spoon
% dir
11/21/2003 10:17a 1,927,339 tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% dir
11/21/2003 10:17a <DIR> spoon
11/21/2003 10:17a 1,927,339 tick-3.14.ism
```

ISMTool Build Commands

This section describes the ISMTool commands that build and modify an ISM.

--verbose

Display extra debugging information.

--banner

Suppress the display of the output banner.

--clean

Clean up all files generated as a result of a build. This removes temporary files and all build products.

--build

Builds the ISM, creating the packages in the `pkg` subdirectory.

The primary purpose of the build command is to create the packages contained in the ISM. Optionally, the build command may invoke source compilation and run pre-build and post-build scripts.

--upgrade

Upgrade the ISM to match the currently installed version of the ISMTool.

New releases of the ISMTool may fix defects or modify how it operates on an extracted ISMDIR. If the version of the currently installed ISMTool is different from the version of the ISMTool that created the ISM, the developer may need to perform certain actions. Note that minor and major downgrades are not allowed. For example, if version 2.0.0 of the ISMTool created the ISM, then version 1.0.0 of the ISMTool cannot process the ISM. Table 52 lists the developer actions if the currently installed and previous versions of ISMTool are not the same.

table 52 ISMTool Upgrade Actions

ISMTool Version Currently Installed	ISMTool Version Used to Create the ISM	Developer Action
1.0.1	1.0.0	PATCH increment. Developer action is not needed. This is considered a simple automatic upgrade which is forward AND backward compatible.
1.0.0	1.0.1	PATCH decrement. Automatic downgrade. No action needed.
1.1.0	1.0.0	MINOR increment. The developer must apply the `--upgrade` command to the ISM. There may be small operational differences or enhanced capability. Warning: This operation is not reversible. Minor upgrades are designed to be as transparent as possible.
2.0.0	1.0.0	MAJOR increment. The developer must apply the `--upgrade` command to the ISM. There may be large operational differences. The developer will probably need to perform other actions specified in release notes.
1.0.0	2.0.0 or 1.1.0	MAJOR or MINOR decrement. This downgrade path is not allowed. The ISM cannot be processed with the installed version of the ISMTool.

--name STRING

Change the internal name of the ISM to `STRING`. The `ISMDIR`, the top level directory of an extracted ISM, can have a different name than the internal name of the ISM. To change both names, use the ISMTool `--name` command to change the internal name and a file system command to change the directory name. If the STRING format is not valid for the native packaging engine, the problem will not be found until a `--build` is issued and the packaging engine throws an error.

--version STRING

Change the internal version field of the ISM. The STRING cannot contain spaces. The `--version` command performs no other checks on the STRING format. If the STRING format is not valid for the native packaging engine, the problem will not be found until a `--build` is issued and the packaging engine throws an error.

--prefix PATH

Change the install prefix of an ISM. The `PATH` is used by the build-from-source feature of the ISMTool and also by the drivers for the packaging engines. During installation on a managed server, the application files packaged in the ISM are installed in the location relative to the `PATH`. In the SA Client, the `PATH` appears in the Install Path field in the package's properties. In the following Unix example, the developer begins with this `.tar` file:

```
% tar tvf ntp/bar/ntp.tar
```

```
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 etc/ntp.conf
drwxr-xr-x ntp/ntp           0 2003-07-08 16:22:38 etc/ntp/
-rw-r--r-- root/root        22 2002-11-22 09:22:08 etc/ntp/step-tickers
-rw-r--r-- ntp/ntp           7 2003-07-08 16:22:38 etc/ntp/drift
-rw------- root/root       266 2001-09-05 03:54:42 etc/ntp/keys
-rwxr-xr-x root/root    252044 2001-09-05 03:54:43 usr/sbin/ntpd
-rwxr-xr-x root/root     40460 2001-09-05 03:54:43 usr/sbin/ntpdate
-rwxr-xr-x root/root     70284 2001-09-05 03:54:43 usr/sbin/ntpdc
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntp-genkeys
-rwxr-xr-x root/root     66892 2001-09-05 03:54:43 usr/sbin/ntpq
-rwxr-xr-x root/root     12012 2001-09-05 03:54:43 usr/sbin/ntptime
-rwxr-xr-x root/root     40908 2001-09-05 03:54:43 usr/sbin/ntptimeset
-rwxr-xr-x root/root     19244 2001-09-05 03:54:43 usr/sbin/ntptrace
-rwxr-xr-x root/root      1019 2001-09-05 03:54:39 usr/sbin/ntp-wait
```

In this example, a `--prefix` of `'/'` would build an application package such that all the files would be installed relative to the file system root.

```
% ismtool --build --prefix '/' --pkgengine rpm4 ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-1.i386.rpm
drwxr-xr-x   2 ntp   ntp         0 Jul  8 16:22 /etc/ntp
-rw-r--r--   1 root  root     1808 Nov 22  2002 /etc/ntp.conf
-rw-r--r--   1 ntp   ntp         7 Jul  8 16:22 /etc/ntp/drift
-rw-------   1 root  root      266 Sep  5  2001 /etc/ntp/keys
-rw-r--r--   1 root  root       22 Nov 22  2002 /etc/ntp/step-tickers
-rwxr-xr-x   1 root  root    40908 Sep  5  2001 /usr/sbin/ntp-genkeys
-rwxr-xr-x   1 root  root     1019 Sep  5  2001 /usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root   252044 Sep  5  2001 /usr/sbin/ntpd
-rwxr-xr-x   1 root  root    40460 Sep  5  2001 /usr/sbin/ntpdate
-rwxr-xr-x   1 root  root    70284 Sep  5  2001 /usr/sbin/ntpdc
-rwxr-xr-x   1 root  root    66892 Sep  5  2001 /usr/sbin/ntpq
-rwxr-xr-x   1 root  root    12012 Sep  5  2001 /usr/sbin/ntptime
-rwxr-xr-x   1 root  root    40908 Sep  5  2001 /usr/sbin/ntptimeset
-rwxr-xr-x   1 root  root    19244 Sep  5  2001 /usr/sbin/ntptrace
```

It is easy to change the install prefix to `'/usr/local'`:

```
% ismtool --build --prefix '/usr/local' ntp
.
.
.
% rpm -qlpv ntp/pkg/ntp-1.0.0-2.i386.rpm
drwxr-xr-x   2 ntp   ntp         0 Jul  8 16:22 /usr/local/etc/ntp
-rw-r--r--   1 root  root     1808 Nov 22  2002 /usr/local/etc/ntp.conf
-rw-r--r--   1 ntp   ntp         7 Jul  8 16:22 /usr/local/etc/ntp/drift
-rw-------   1 root  root      266 Sep  5  2001 /usr/local/etc/ntp/keys
-rw-r--r--   1 root  root       22 Nov 22  2002 /usr/local/etc/ntp/step-
tickers
-rwxr-xr-x   1 root  root    40908 Sep  5  2001 /usr/local/usr/sbin/ntp-
genkeys
-rwxr-xr-x   1 root  root     1019 Sep  5  2001 /usr/local/usr/sbin/ntp-wait
-rwxr-xr-x   1 root  root   252044 Sep  5  2001 /usr/local/usr/sbin/ntpd
-rwxr-xr-x   1 root  root    40460 Sep  5  2001 /usr/local/usr/sbin/ntpdate
-rwxr-xr-x   1 root  root    70284 Sep  5  2001 /usr/local/usr/sbin/ntpdc
```

```
-rwxr-xr-x   1 root  root   66892 Sep  5  2001 /usr/local/usr/sbin/ntpq
-rwxr-xr-x   1 root  root   12012 Sep  5  2001 /usr/local/usr/sbin/ntptime
-rwxr-xr-x   1 root  root   40908 Sep  5  2001 /usr/local/usr/sbin/
ntptimeset
-rwxr-xr-x   1 root  root   19244 Sep  5  2001 /usr/local/usr/sbin/ntptrace
```

On Windows, there is no standard way to tell an MSI where to install itself. Therefore, application packages built from MSI files found in the `bar` directory will ignore the `--prefix` setting. However, for Windows application packages built from ZIP files, the ISMTool will use the `--prefix` setting. On Windows the prefix must be in this form: `driveletter:\directoryname` (for example, `D:\mydir`). On Windows NT4, `--prefix` is required and cannot contain variables.

On Unix, the default value of `PATH` is `/usr/local`. However, on Solaris 11, the default value is /usr/app.

--ctlprefix PATH

Change the install prefix of the control files. Note that this command is not recommended and that you should instead rely on the default values. During installation on a managed server, the control files packaged in the ISM are installed in the location relative to the `PATH`. In the SA Client, the `PATH` appears in the Install Path field in the package's properties. On Windows the prefix must be in this form: `driveletter:\directoryname` (for example, `D:\mydir`). On Windows NT4, `--ctlprefix` is required and cannot contain variables.

The default value for `PATH` follows:

Unix:

`/var/opt/OPSWism`

Windows:

`%ProgramFiles%\OPSWism`

On Solaris, if you specify `--ctlprefix` on Solaris, you will be prompted for the name of the shared run-time package.

--user STRING (Unix only)

Change the Unix user owner of the files in the application package to `STRING`. When the files in the package are installed on the managed server, they will be owned by the specified Unix user.

--group STRING (Unix only)

Change the Unix group owner of the files in the application package `STRING`.

--ctluser STRING (Unix only)

Change the Unix user owner of the files in the control package to `STRING`. The default value is `root`. When the files in the package are installed on the managed server, they will be owned by the specified Unix user.

--ctlgroup STRING (Unix only)

Change the Unix group owner of the files in the control package to `STRING`. The default value is `bin`.

--pkgengine STRING (Unix only)

Change the native packaging engine. On systems that have multiple packaging engines available, use this command to switch between them. To view the available engines, issue the `--help` or `--env` commands.

Note that if you change the native packaging engine, no packages will be added to the software policy during the `--upload` operation.

--ignoreAbsolutePaths BOOL (Unix only)

Ignore the absolute paths in the archive. For example, the following is a binary archive with absolute paths:

```
% tar tvf test/bar/foo.tar
-rw-r--r-- root/root      1808 2002-11-22 09:20:36 /foo/bar/baz.conf
```

If the `--prefix` is set to `/usr/local` then the install path is ambiguous: Should ISMTool install `baz.conf` as `/foo/bar/baz.conf` or `/usr/local/foo/bar/baz.conf`? If the answer is `/foo/bar/baz.conf`, then the developer must set the `--prefix` of the ISM to `'/'`. However, if the answer is `/usr/local/foo/bar/baz.conf`, then the developer must specify the `--ignoreAbsolutePaths` command.

--addCurrentPlatform (Unix only)

Add the current platform to the ISM's supported list. Note: This command does not make the ISM cross-platform. ISMs can be constructed on different SA-supported platforms. A platform is the combination of OS type and version. Example platforms are: Redhat-Linux-7.2, SunOS-5.9, Windows-2000. To view the currently supported platforms for an ISM use the `--info` command.

--removeCurrentPlatform (Unix only)

Removes the current platform from the ISM's supported platform list.

--addPlatform TEXT (Unix only)

Add to the ISM's supported platform list the platform specified by the TEXT. Because platform support and identification are dynamic, no error checking is done for `--addPlatform`. For this reason, the recommendation is to use `--addCurrentPlatform` instead of `--addPlatform`.

--removePlatform TEXT (Unix only)

Removes from the ISM's supported platform list the platform specified by the TEXT.

--target STRING (Unix only)

Warning: This command should only be used by experts.

Allow cross-platform packaging of the application package for the RPM packaging engine. The `--target` command must be used with `--skipControlPkg`. The format of the `STRING` is `<arch-os>`, for example, `i686-linux` or `sparc-solaris2.7`.

--skipControlPkg BOOL

Prevent the building of the control package. This command allows the ISMTool to support the packaging of files that have no need for a structured application control package.

--skipApplicationPkg BOOL

Prevent the building of the application package. This command allows the ISMTool to support the creation of a control-only ISM package. This feature can be used to build a controller for an application that is not installed or packaged with the ISMTool. Examples are controllers for core operating system functions, currently running applications that cannot be packaged, and specialized hardware.

--chunksize BYTES (Unix only)

Limits the number of bytes that will be inserted into an application package. (Heuristics are used to compensate for compression factors.) The binary archive (`bar`) directory may contain many archives from which to build the application package. If the chunksize is exceeded, then the application archives are grouped into several bins and each bin is turned into a-sub application package. The algorithm is a standard bin-packing heuristic. The movable units are binary archives within the `bar` directory.

For example, suppose that the output package format is an RPM and has five binary archives: `a.tgz` (100M), b.tgz(100M), `c.tgz` (200M), `d.tgz` (300M), and e.tgz(50M). If the chunksize is set to 314572800 (300M) then the output application bins will be:

```
part1( a.tgz, b.tgz, e.tgz ) == 250M
part2( c.tgz )               == 200M
part3( d.tgz )               == 300M
```

This would result in three application packages:

```
foobar-part0-1.0.0.i386.rpm
foobar-part1-1.0.0.i386.rpm
foobar-part2-1.0.0.i386.rpm
```

In general, the chunksize is not a problem unless the application package is almost a gigabyte in size. At that point, some package engines start breaking. The default chunksize is one gigabyte (2 ^30 bytes).

--solpkgMangle BOOL (SunOS only)

Prevent the ISMTool from changing the name of the application package to conform to Solaris requirements. For more information, see Solaris Differences on page 66.

When creating a Solaris package, ISMTool must use a package name that conforms to the 9-character limit. However, it may be desirable to prevent ISMTool from changing ("mangling") the package name during the `--build` process. When `--solpkgMangle false` is specified, ISMTool will use the ISM name when creating the application package. The control package name will continue to be mangled. Note that when `--solpkgMangle` is `false`, the ISM name must be 9 characters or less and there cannot be multiple application packages.

--embedPkgScripts BOOL

Embed the contents of the ISM packaging scripts (installation hooks) in the application package. This option must be used with `--skipControlPkg` and `--skipRunTimePkg`.

By default, the application package is built to call out to the ISM packaging scripts installed by the control package. The `--embedPkgScripts` option overrides this behavior by embedding the contents of the scripts found in the `ism/pkg` directory inside the application package. These scripts are invoked during the pre and post phases of the application package install and uninstall.

If one or more of the scripts in the `ism/pkg` directory are not needed, delete the scripts before the `--build` process. Note that RPM and LPP packaging engines do not have a `checkinstall` phase so the `ism_check_install` file is ignored when building RPMs and LPPs.

--skipRuntimePkg BOOL

Specify whether to build runtime packages during subsequent `--build` operations.

A runtime package is built by default. If `--skipRuntimePkg` true is specified, the runtime package will not be built during subsequent operations until `--skipRuntimePkg` false is specified. ISM utilities such as the `parameters` interface will fail if the runtime package cannot be located. Do not specify `--skipRuntimePkg` true unless you are sure the runtime package already exists on the managed server on which you'll install the ISM.

ISMTool Interface Commands

This section describes the ISMTool commands that interact with SA.

--upload

Upload the ISM contained in the ISMDIR to the software policy specified by `--opswpath`. If you specify a software policy that does not exist, it will be created automatically during the upload process. To specify which SA core to connect to, use either command-line arguments (such as `--softwareRepository`) or the environment variables listed in Table 53.

The `--upload` command prompts for an SA user name and password. Before the upload operation, this user must be granted permission with `ismusertool`. Also, this user must have write permission on the folder containing the software policy.

--noconfirm

Suppress confirmation prompts, which require a `y` or `n` reply. For example, the ISMTool has the following confirmation prompt:

```
Do you wish to proceed with upload? [y/n]:
```

If `--noconfirm` is set, the prompts are suppressed and the ISMTool behaves as if the answer is `y`. The `--noconfirm` option affects only the current invocation of the ISMTool.

--opswpath STRING

Specify the path of the software policy that will contain the uploaded ISM. Note that the path always contains forward slashes, even on Windows.

If you specify a software policy that does not exist, it will be created automatically during the upload process. If you specify a folder (a path not terminated by a policy), an error occurs because you cannot upload an ISM into a folder.

The ISMTool supports the construction of cross-platform ISMs. An example of such an ISM is the Network Time Protocol (NTP) daemon, which can be built from source on a variety of platforms. To make uploading of cross-platform ISMs easier, the ISMTool supports variable substitution within the `--opswpath STRING`. These variables represent the internal settings of the ISM. Table 53 lists the variables recognized by the ISMTool.

table 53 ISMTool Variables

Variable	Example
${NAME}	ntp
${VERSION}	4.1.2
${APPRELEASE}	3
${CTLRELEASE}	7
${PLATFORM}	Redhat Linux 7.2
${OSTYPE}	Redhat Linux
${OSVERSION}	7.2

Unix example:

```
% ismtool --opswpath '/System Utilities/${NAME}/${VERSION}/${PLATFORM}'
ntp
```

Possible expansion:

```
'/System Utilities/ntp/4.1.2/Redhat Linux 7.2'
```

Windows example:

```
% ismtool --opswpath "/System Utilities/${NAME}/${VERSION}/${PLATFORM}"
ntp
```

Possible expansion:

```
"/System Utilities/ntp/4.1.2/Windows 2000"
```

--commandCenter HOST[:PORT]

For an upload to a folder, use the Opsware Command Center core component located at `HOST[:PORT]`.

--dataAccessEngine HOST[:PORT]

For the upload, use the SA Data Access Engine located at `HOST[:PORT]`.

--commandEngine HOST[:PORT]

For the upload, use the SA Command Engine located at `HOST[:PORT]`.

--softwareRepository HOST[:PORT]

For the upload, use the SA Software Repository located at `HOST[:PORT]`.

--description TEXT

Provide descriptive text for the ISM. During the upload, this text is copied to the description field on the software policy.

--addParam STRING

Add a parameter named STRING to the ISM. Usually, the commands `--paramValue`, --`paramDesc`, and –`-paramType` are also specified. For example:

```
% ismtool --addParam NTP_SERVER \
          --paramValue 127.0.0.1 \
          --paramType 'String' \
          --paramDesc 'NTP server, default to loopback' ntp

% ismtool --addParam NTP_CONF_TEMPLATE \
          --paramValue /some/path/ntp.conf.template \
          --paramType 'Template' \
          --paramDesc 'Template for the /etc/ntp.conf file' ntp
```

--paramValue TEXT

Set the default value for the parameter. The `--addParam` command must also be specified. If the parameter type is `'String'` then the value is the string specified by `TEXT`. If the parameter type is `'Template'` then `TEXT` is interpreted as a PATH to a configuration template file. The data in the template file is loaded as the default value. If the `--paramValue` and `--paramType` are not specified, then the default value is the empty string.

--paramType PARAMTYPE

Set the type of the parameter. The `--addParam` command must also be specified. The `PARAMTYPE` must be either `'String'` or `'Template'`. The default type is `'String'`.

--paramDesc TEXT

Set the descriptive text for the parameter. The `--addParam` command must also be specified. The default value is an empty string.

--removeParam STRING

Remove the parameter named STRING.

--rebootOnInstall BOOL

Tag the application package with the SA package control flag `reboot_on_install`. If `--rebootOnInstall` is set to true, then the managed server will be rebooted after the package is installed. If the ISM has multiple application packages, the last package in the list is tagged.

--rebootOnUninstall BOOL

Tag the application package with the SA package control flag `reboot_on_uninstall`. If `--rebootOnUninstall` is set to true, then the managed server will be rebooted after the package is uninstalled. If the ISM has multiple application packages, the first package in the list is tagged.

--registerAppScripts BOOL (Windows only)

Register the ISM packaging scripts (installation hooks) with the application package.

By default, ISM packaging scripts are encoded in the application MSI to run at pre-installation, post-installation, pre-uninstallation, and post-uninstallation. When `--registerAppScripts` is specified, the ISM packaging scripts are instead registered as SA package control scripts during the upload. The package control scripts are registered in the Model Repository and are viewable from theHP Server Automation Client.

The `--registerAppScripts` command is required if the ISM packaging scripts contain actions that conflict with the application MSI installation. For example, a conflict could occur if a post-install script contains a call to `msiexec.exe`. Since the Microsoft Installer does not allow concurrent installs, a script containing a call to `msiexec.exe` will not complete successfully. By registering the ISM packaging scripts as SA package control scripts, the scripts are called outside of the MSI installation and uninstallation.

--endOnPreIScriptFail BOOL (Windows only)

Register to end subsequent installs with the application package.

If `--endOnPreIScriptFail` and `--registerAppScripts` are both set to true, then the installation will abort if the ISM pre-install script returns a non-zero exit code.

--endOnPstIScriptFail BOOL (Windows only)

Register to end subsequent installs with the application package.

If `--endOnPstIScriptFail` and `--registerAppScripts` are both set to true, then the installation will abort if the ISM post-install script returns a non-zero exit code.

--endOnPreUScriptFail BOOL (Windows only)

Register to end subsequent uninstalls with the application package.

If `--endOnPreUScriptFail` and `--registerAppScripts` are both set to true, then the uninstall will abort if the ISM pre-uninstall script returns a non-zero exit code.

--endOnPstUScriptFail BOOL (Windows only)

Register to end uninstalls with the application package.

If `--endOnPstUScriptFail` and `--registerAppScripts` are both set to true, then the uninstall will abort if the ISM post-uninstall script returns a non-zero exit code.

--addPassthruPkg {PathToPkg} --pkgType {PkgType} ISMNAME

Specifies that the package identified by `{PathToPkg}` should be treated as a passthru package. The supported package type `{PkgType}` depends on the platform, as shown by Table 54.

`{PathToPkg}` can be either a full or relative path to the package, but the package must exist at the time the `--addPassthruPkg` option is specified. `{PathToPkg}` cannot specify a package in the current ISM's directory structure. For example, the control package, the application package, or a package in the bar directory cannot be specified as a passthru package.

Note that by default, the upload operation does not add the passthru package (specified by `--addPassthruPkg`) to the software policy. To add the passthru package, you must specify the `--attachPkg` option.

If you upload a Solaris passthru package, the response file is not uploaded. You must manually upload the response file.

The following table lists the allowed values of `{PkgType}` (package type) for each platform.

table 54 Supported Package Types for Passthru Option

Platform (OS)	Allowed Value for {Pkgtype}
AIX	lpp rpm zip
HP-UX	depot zip
Linux	rpm zip
SunOS	rpm solcluster solpatch solpkg ips zip
Windows	hotfix msi sp zip

The following example shows how to add a passthru package to an ISM and specify the package for addition to the software policy:

```
% ismtool --addPassthruPkg /tmp/bos.rte.libs.5.1.0.50.U --pkgType lpp ISMNAME
Inspecting specified package: ...
bos.rte.libs.5.1.0.50.U (lpp)
```

```
  bos.rte.libs-5.1.0.50 (update fileset)
  IY42527 (apar)
Done.
% ismtool --attachPkg bos.rte.libs-5.1.0.50 --attachValue true ISMNAME
```

--removePassthruPkg {PassthruPkgFileName} ISMNAME

Specify that an already registered passthru package is no longer a passthru package.

ISMTool will do the following:

1. Delete {PassthruPkgFileName} from the ISMs directory structure.
2. Record in ism.conf that {PassthruPkgFileName} is no longer a passthru package.
3. During the next upload and all subsequent uploads, if the package is added to the `--opswpat`h software policy, it will be removed.

Note that an ISM remembers all packages that have been removed as a passthru package. If a package was added to the software policy via the SA Client or a previous upload operation, the package will be removed from the policy on the next upload operation.

--attachPkg {PkgName} --attachValue BOOLEAN ISMNAME

Specify whether a package managed by an ISM should be added to the software policy identified by `--opswpath`.

By default, when control or application packages are built, these types of packages are marked for addition to the software policy. However passthru packages and inner packages are not marked for addition until the `--attachPkg` option is specified.

{PkgName} is the name of the package as listed by the `--showPkgs` command. If `--attachValue` is true, a package is marked for addition to the softare policy. If `--attachValue` is false, a package will be uploaded into the Software Repository but it will not be added to the software policy. If `--attachValue` is false and the package already resides in the software policy, the package is marked for removal from the policy. A package is added or removed during an `--upload` operation. The following table lists the package types that can be added to a softare policy.

table 55 Package Type Properties

Package type	Can this package type contain scripts?	Can this package type be added to a software policy?
AIX LPP	no	no
AIX Base Fileset	yes	yes
AIX Update Fileset	yes	yes
AIX APAR	no	yes
HP-UX Depot	no	no
HP-UX Fileset	yes	yes
HP-UX Patch Fileset	no	no
HP-UX Product	no	yes
HP-UX Patch Product	no	yes

table 55 Package Type Properties (cont'd)

Package type	Can this package type contain scripts?	Can this package type be added to a software policy?
IPS Package	no	yes
RPM	yes	yes
Solaris Package	no	no
Solaris Package Instance	yes	yes
Solaris Patch	yes	yes
Solaris Patch Cluster	no	yes
Windows Hotfix	yes	yes
Windows MSI	yes	yes
Windows Service Pack	yes	yes
Windows ZIP File	yes	yes

--orderPkg {PkgName} --orderPos {OrderPos} ISMNAME

Change the install order of attached packages managed by the ISM.

{OrderPos} is an integer that specifies the new install order for the package identified by {PkgName}. {OrderPos} is 1 (not 0) or the first package to be installed. To display the install order, use the `ismtool --showOrder` command.

The following example shows how to display and change the install order:

```
% ismtool --showOrder ISMNAME
[1] test-ism-1.0.0-1.rpm
[2] test-1.0.0-1.rpm
[3] bos.rte.libs-5.1.0.50
[4] IY42527

% ismtool --orderPkg IY42527 --orderPos 1 ISMNAME
[1] IY42527
[2] test-ism-1.0.0-1.rpm
[3] test-1.0.0-1.rpm
[4] bos.rte.libs-5.1.0.50
```

--addPathProp {PathProp} --propValue {PropValue} ISMNAME

Specific a value for a property (meta data) of the software policy.

To display the current values, use the --showPathProps command. The following table lists the allowed values and types for the --addPathProp command.

table 56 Allowed values for {PathProp}

{PathProp} Allowed Value	{PropValue} Type	Example
description	TEXT	'This does something important'
Deprecated: notes	TEXT	'And so does this'
Deprecated: allowservers	BOOLEAN	false

The following example commands show how to set the description property:

```
% ismtool --addPathProp description --propValue 'This policy does something'
ISMNAME
% ismtool --showPathProps ISMNAME
description:   This policy does something
```

--editPkg {PkgName} --addPkgProp {PkgProp} --propValue {PropValue} ISMNAME

Specify a value for a given package meta data property.

{PkgName} identifies the package to update; it can be any of the package names listed using the --showPkgs command. The following table lists the allowed values for {PkgProp}.

table 57 Allowed values for {PkgProp}

{PkgProp} allowed value	Description	{PropValue} type
deprecated	Deprecated status for package	BOOLEAN
description	Description for package	TEXT
endonpreiscriptfail	Remediation ends on pre- install script failure	BOOLEAN
endonpreuscriptfail	Remediation ends on pre-uninstall script failure	BOOLEAN
endonpstiscriptfail	Remediation ends on post-install script failure	BOOLEAN
endonpstuscriptfail	Remediation ends on post-uninstall script failure	BOOLEAN
installflags	Install flags for package	TEXT
notes	Notes for the package	TEXT

table 57 Allowed values for {PkgProp}

{PkgProp} allowed value	Description	{PropValue} type
`rebootoninstall`	Package requires a reboot after install	BOOLEAN
`rebootonuninstall`	Package requires a reboot after uninstall	BOOLEAN
`uninstallflags`	Uninstall flags for package	TEXT

The endonXXXscriptfail values are set only if a pre/post install/uninstall script has been defined for a package. These scripts reside in the ISMNAME/pad subdirectory.

Note that not all package types support all the {PkgProp} values listed in the preceding table. The supported {PkgProp} values for each package type can be seen by viewing the package property details in the SA Client. In addition, the following table lists {PkgProp} values supported by specific package types.

table 58 {PkgProp} Allowed Values by Package Type

{PkgProp} allowed value	Package Type	Description	{PropValue}
`upgradeable`	RPM	Package is upgradeable	BOOLEAN
`productname`	Windows MSI	MSI product name	STRING
`productversion`	Windows MSI	MSI version number	STRING
`servicepacklevel`	Windows OS Service Pack	Service Pack version number	INTEGER
`installdir`	Windows ZIP	Installation directory	STRING
`postinstallscriptfilename`	Windows ZIP	Post install script filename	STRING
`postinstallscriptfilenamefail`	Windows ZIP	Remediation ends on post install script failure	BOOLEAN
`preuninstallscriptfilename`	Windows ZIP	Pre uninstall script filename	STRING
`preuninstallscriptfilenamefail`	Windows ZIP	Remediation ends on pre uninstall script failure	BOOLEAN

The `productversion`, `productname, and servicepacklevel` must be set before performing an ––`upload` operation. The `productname` and `productversion` cannot be changed after an `--upload` operation. If you modify the `productname` or `productversion` and then perform another `--upload` operation, the modified values will not be applied.

The following example shows how to specify the description of a package:

```
% ismtool --editPkg bos.rte.libs.5.1.0.50 --addPkgProp description --
propValue 'This is a fileset' ISMNAME
```

ISMTool Environment Variables

The ISMTool references the shell environment variables described in this section.

CRYPTO_PATH

This environment variable indicates the directory that contains the files `agent/agent.srv` and `agent/opsware-ca.crt`.

`CRYPTO_PATH`, `agent.srv` and `opsware-ca.crt` are required only if you are uploading the ISM from a server not managed by SA (that is, a server that has no Server Agent.)

To connect to the SA core during the upload of an ISM, the ISMTool needs the client certificates (the `agent.srv` and `opsware-ca.crt` files) that were generated during the installation of HP Server Automation.

Keep in mind that using these certificates with the ISMTool invokes a different security mechanism than the one used by the SA Client. As a result, you might have increased or reduced permissions. You might have access to servers belonging to customers that you usually do not have access to.

Also, you might be able to perform operations that you cannot perform with the SA Client. Therefore, in this situation use the ISMTool with caution to avoid unintended consequences caused by a possible change in security permissions.

To obtain the `agent.srv`, `opsware-ca.crt` files and set the CRYPTO_PATH environment variable, perform the following steps:

1. Log in to the SA core server as `root` and locate the following file:

   ```
   /var/opt/opsware/crypto/agent/agent.srv
   /var/opt/opsware/crypto/agent/opsware-ca.crt
   ```

2. Copy `agent.srv` and `opsware-ca.crt` to the server where you have installed the IDK, to the following directory:

   ```
   <some-path>/agent
   ```

   The `<some-path>` part of the directory path is your choice, but the subdirectory containing `agent.srv` and `opsware-ca.crt` must be `agent`.

3. Set the `CRYPTO_PATH` environment variable to `<some-path>`.

   For example, on a Unix server, suppose that the full path name of `agent.srv` and `opsware-ca.crt` is as follows:

   ```
   /home/jdoe/dev/
   ```

   In csh you would set the environment variable as follows:

   ```
   setenv CRYPTO_PATH /home/jdoe/dev/
   ```

   On Windows, perhaps agent\agent.srv and agent\opsware-ca.crt reside here:

   ```
   C:\jdoe\dev\
   ```

   You could set the environment variable as follows:

   ```
   set CRYPTO_PATH=C:\jdoe\dev\
   ```

ISMTOOLBINPATH

This environment variable is a list of directory names, separated by colons, where the ISMTool searches for system-level tools (such as tar and cpio). The following search strategy is used:

1. Search the paths from the environment variable ISMTOOLBINPATH.
2. Search the complied-in binaries (if any) in /usr/local/ismtool/lib/tools/bin.
3. Search within the user's path.

ISMTOOLCC

This environment variable is the HOST[:PORT] of the Opsware Command Center core component used during an ISMTool upload to a folder.

ISMTOOLCE

This environment variable is the HOST[:PORT] of the SA Command Engine used by the ISMTool.

ISMTOOLDA

This environment variable is the HOST[:PORT] of the SA Data Access Engine used by the ISMTool.

ISMTOOLPASSWORD

This environment variable is a STRING that specifies the SA password during an ISMTool upload.

ISMTOOLSITEPATH

This environment variable is a PATH for a “site” directory.

The ISMTool contains certain default scripts and attribute values (for example, the install prefix) which are referenced when a new ISM is created. A developer can override the default scripts and a selected set of attribute values by using a site directory.

The defaults.conf File

Within the site directory, a developer can create the defaults.conf file, which contains overrides for attribute values. A line in defaults.conf has the format: <tag>:<value>. A line starting with the # character is a comment. The following example shows the values that can be set in defaults.conf:

Unix:

```
prefix:    /usr/local
ctlprefix: /var/opt/OPSWism
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
ctluser:   root
ctlgroup:  bin
```

Windows:

```
prefix:    ???
ctlprefix: ???
opswpath:  /System Utilities/${NAME}/${VERSION}/${PLATFORM}
version:   1.0.0
```

The templates Subdirectory

Developers can override the files in the /usr/local/ismtool/lib/ismtoollib/templates directory by placing their own copies in a templates subdirectory located within the ISMTOOLSITEPATH. For example, developers can override the files that are the default packaging hooks for Windows or Unix.

The control Subdirectory

Sometimes, developers need to install a common set of tools into an ISM's control directory. The ISMTool supports this requirement by copying all files from a control subdirectory of the ISMTOOLSITEPATH to the ISM's control directory. If a file already exists in the ISM's control directory, it will not be overwritten.

ISMTOOLSR

This environment variable is the HOST[:PORT] of the SA Software Repository used by the ISMTool.

ISMTOOLUSERNAME

This environment variable is a STRING that specifies the SA user name during an ISMTool upload.

ISMUserTool

The --upload command of the ISMTool prompts for an SA user name. To enable SA users to perform an upload, run the ISMUsertool to assign privileges.

To list the users that have upload privileges:

```
% ismusertool --showUsers
```

To grant a user users upload privileges:

```
% ismusertool --addUser johndoe
```

To revoke upload privileges:

```
% ismusertool --removeUser johndoe
```

ISMUsertool allows you to specify multiple options on a single command line. For more information, specify the --help option:

```
% ismusertool --help
```

By default, the Opsware admin user has upload privileges, which cannot be revoked.

Folders are new in version 6.0 of Server Automation. To upload an ISM into a folder, the user must have folder privileges. By default, the admin user does not have folder privileges. In a production environment, admin should not have folder privileges, so you should not use admin for uploads.