Opsware® SAS ISM Development Kit 2.0 Guide

Corporate Headquarters

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

Table of Contents

Preface

Document Date: 8/9/05

Welcome to the Opsware Server Automation System (SAS) — an enterprise-class software solution that enables customers to get all the benefits of Opsware Inc.'s data center automation platform and support services. Opsware SAS provides a core foundation for automating formerly manual tasks associated with the deployment, support, and growth of server and server application infrastructure.

About this Guide

This guide describes how to create and upload Intelligent Software Modules (ISMs) with the ISMTool. An ISM is a set of files and directories that include application bits, installation scripts, and control scripts. With the ISMTool, a command-line utility, you create ISMs and upload them to the Opsware core. After an ISM has been uploaded, it appears in the Opsware Command Center (OCC) as a software node with attached packages.

This guide is intended for developers and advanced Opsware administrators who will create and upload ISMs. To understand the material in this guide, you should already be familiar with script programming and package installation on the OS platforms that you support.

This guide explains how to develop ISMs in a command-line environment. For instructions on using the Opsware Visual Packager, a GUI tool that creates software packages, see the *Opsware® SAS 5.2 User's Guide*.

Contents of this Guide

This guide contains the following chapters:

Chapter 1: Overview - Summarizes the IDK and provides a tutorial for creating and uploading a simple ISM.

Chapter 2: ISM Build Environment - Describes files and directory structure of an ISM.

Chapter 3: ISM Scripts - Explains how to customize ISMs with scripts and parameters.

Chapter 4: ISMTool Commands - Describes the syntax of the ISMTool command, the primary utility of the IDK.

Appendix A: ISMUserTool - Describes the syntax of the ISMUserTool.

Appendix B: Platform Differences - Summarizes IDK differences between operating system platforms.

Conventions in this Guide

This guide uses the following typographical and formatting conventions.

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

Icons in this Guide

This guide uses the following icons to indicate important information.

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

Chapter 1: Overview

IN THIS CHAPTER

This chapter discusses the following topics:

- Introducing the IDK and ISMs
- What's New in This Release
- Installing the IDK
- IDK Quickstart

Introducing the IDK and ISMs

Opsware SAS 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 Opsware core. The upload operation creates a node for the application in the software tree and associates installable packages with the node. After uploading the ISM, you use the Opsware Command Center 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 Opsware SAS, 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 Opsware SAS 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 Opsware SAS 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

What's New in This Release

Version 2.0 of the IDK has the following new features:

- Shared Runtime Packages
- Passthru Packages
- Meta Data Update

Shared Runtime Packages

An ISM requires runtime routines that are provided by Opsware SAS. Prior to IDK 2.0, the ISMTool `--build` operation added the runtime routines to the control package of each ISM. (A separate runtime package was not created.) If you installed more than one ISM onto a managed server, multiple copies of the runtime routines were also installed.

IDK 2.0 enables the sharing of the runtime routines. The ISMTool `--build` operation adds the routines to the runtime package of the ISM. The runtime package resides in `pkg`, the same subdirectory as the control and application packages. If you upload multiple ISMs into a core, just one copy of the runtime package is stored in the Software Repository. Likewise, if you install multiple ISMs onto a managed server, just one copy of the runtime package installed. During uninstallation, the runtime package is removed from the managed server only if no other installed ISM relies on it.

Passthru Packages

Before version 2.0 of the IDK, to include third-party packages (such as RPM and ZIP files) in an ISM, you copied them to the ISM's `bar` subdirectory. During the upload process, the ISMTool unpacked the third-party packages it found in the `bar` subdirectory and repackaged their contents into the ISM.

Starting with the IDK 2.0 release, you can associate third-party packages with an ISM. The ISMTool copies these passthru packages to the ISM's `pkg` subdirectory, does not unpack them, and uploads them unchanged. (The ISMTool still unpacks the third-party packages that are in the `bar` subdirectory.)

The new ISMTool options for passthru packages are as follows:

```
--addPassthruPkg
--removePassthruPkg
--showPkgs
--attachPkg
--showOrder
--orderPkg
```

Meta Data Update

The meta data for packages and software tree nodes is displayed by the Properties tabs in the Opsware Command Center. For example, the Properties tab for a package displays values in the Description and Install Flags fields.

Before the 2.0 version of the IDK, to change the properties you had to log in to the Opsware Command Center after uploading the ISM. Starting with the 2.0 release, you can specify the properties with the ISMTool before uploading the ISM.

The new ISMTool options for meta data updates are as follows:

```
--addPathProp
--showPathProps
--editPkg
```

Installing the IDK

This section discusses the following topics:

- Installing the IDK for the Visual Packager
- Installing the IDK for Command-Line Package Development
- Installing the IDK on Opsware SAS 4.x

Installing the IDK for the Visual Packager

Before using the Visual Packager feature, you must install the IDK with the Install Template Wizard of the Opsware Command Center. To install the IDK with this wizard, perform the following steps:

1. Log in to the Opsware Command Center.
2. Verify that the host where you install the IDK is managed by Opsware SAS.

   For more information, see "Server Search Overview" in the *Opsware® SAS 5.2 User's Guide*.
3. Verify that the host where you install the IDK runs the same operating system version as the managed servers where you will install the packages created with the Visual Packager.

   For example, if you're creating packages for Redhat Linux 7.3 managed servers, install the IDK on a Redhat Linux 7.3 system.
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 In the Opsware Command Center, run the Install Template Wizard. The template to install is located at the following path:

Opsware Tools | Visual Packager

If the Opsware Command Center reports that the IDK (ISMTool package) is already installed, but this is a new version of Opsware SAS, go ahead and re-install the IDK. For more information, see "Installing Templates with the Install Templates Wizard" in the *Opsware® SAS 5.2 User's Guide*.

Installing the IDK for Command-Line Package Development

This guide describes how to build packages with the command-line ISMTool of the IDK. To install the IDK for use with the ISMTool, perform the following steps:

1 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're creating ISMs for applications to be installed on Redhat Linux 7.3 managed servers, install the IDK on a Redhat Linux 7.3 system.

2 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
```

3 Recommended: Verify that the host where you install the IDK is managed by Opsware SAS. (In other words, an Opsware Agent has been installed on the host.)

If you install the IDK on a server without an Opsware Agent, you can build ISMs but you cannot upload them to an Opsware core unless you set the `CRYPTO_PATH` environment variable. (See "CRYPTO_PATH" on page 65 of this guide.)

Although you can build and upload ISMs on an Opsware core server, this practice is not recommended because 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.

4 In the Opsware Command Center, run the Install Template Wizard. The template to install is located at the following path:

Opsware Tools | Visual Packager

If the Opsware Command Center reports that the IDK (ISMTool package) is already installed, but this is a new version of Opsware SAS, go ahead and re-install the IDK. For more information, see "Installing Templates with the Install Templates Wizard" in the *Opsware® SAS 5.2 User's Guide*.

If you have previously installed the ISMtool package with the Install Software Wizard, you can continue to use the ISMTool, but you cannot use the Visual Packager until you install the template.

5. Unix: In a terminal window, log in to the host where you've installed the IDK and set the PATH environment variable to the following value.
   `/usr/local/ismtool/bin/`
   (On Windows the `PATH` is set automatically, but will not take effect until you log in again.)

6. In a terminal window, check the IDK installation by entering the following command:
   `ismtool --myversion`

Installing the IDK on Opsware SAS 4.x

The IDK software is included with Opsware SAS 5.x. To install the IDK on Opsware SAS 4.x, perform the following steps:

1. Obtain the IDK software from the Opsware Inc. download page.
2. Perform the steps in the preceding section, except for step 4.

IDK Quickstart

This section shows how to create, build, and upload a simple ISM. After the upload operation, you can examine the resulting Opsware node and packages by running the Opsware Command Center.

Creating, Building, and Uploading a Simple 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. Grant your Opsware user the privilege to upload ISMs by entering the following command:
   `ismusertool --addUser johndoe`

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:
Command Engine     : d02 192.168.198.91:1004
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 Appendix A, "ISMUsertool."

2 Create a new ISM.

For example, to create an ISM named `foo`, you enter the following at the command-line prompt:

```
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.

3 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
```

4 Set the path to the application node of the Opsware software tree.

This node is created in a later step when you upload the ISM into the Opsware core. The following `ismtool` command sets the path to the node:

Unix:

```
ismtool  --opswpath '/System Utilities/test/abc' foo
```

Windows:

```
ismtool  --opswpath "/System Utilities/test/abc" 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.

5 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-2.0.0-1.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 3. 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-2.0.0-1.i386.rpm` package contains the Opsware shared runtimes that the Opsware 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-2.0.0-1.msi
```

6 Upload the ISM into the Opsware core 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, password, and customer:

```
  Enter Opsware Username: johndoe
  Enter johndoe's Opsware Password:
  Enter Opsware Customer: Customer Independent
.. .
Success!
```

The upload operation creates the software node you specified in step 4 and attaches to the node the packages you built in step 5. The `--upload` command is the last step that you perform with the IDK's tools.

Examining the Node and Packages in the Opsware Command Center

To see the results of the upload operation from the preceding section, perform the following steps:

1. Log in to the Opsware Command Center.
2. In the software tree, navigate to the application node you specified with the `ismtool --opswpath` command in step 4 of the previous section.

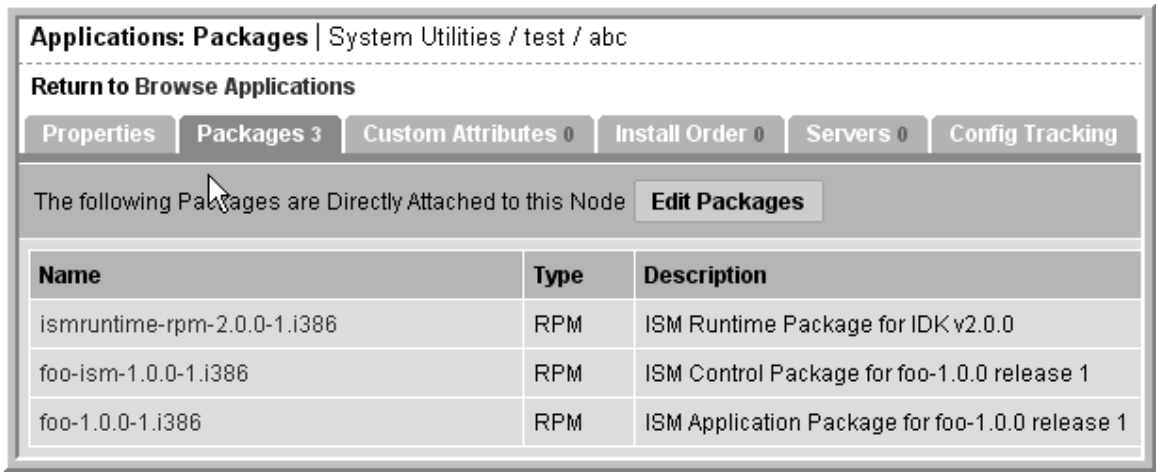
Figure 1-1 shows the Properties tab for the `/System Utilities/test/abc` node.

Figure 1-1: Properties of an Application Node

3. To list the packages attached to this node, select the Packages tab.

Figure 1-2 shows the Packages tab of the `/System Utilities/test/abc` node. These packages were uploaded from the `foo/pkg` subdirectory of the ISM.

Figure 1-2: Packages of an Application Node



Chapter 2: ISM Build Environment

IN THIS CHAPTER

This chapter discusses the following topics:

- ISM File System Structure
- Build Process
- Specifying the Application Files of an ISM
- ISM Name, Version Number, and Release Number

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 `ism/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 Opsware 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 node path to take effect before you upload the ISM into the core.

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. The build cache can be cleaned using the `--clean` command.
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 20 of this guide.)

  ```
  <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 Opsware agent (during installation) and by any control scripts. These runtime routines are for Opsware SAS, 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 "Compiling Source (Unix Only)" on page 17 of this guide.)

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 2-1: Valid Binary Archive Types

FILE EXTENSION	ARCHIVE TYPE
`.cpio`	Unix CPIO Archive
`.msi`	Microsoft Installer
`.rpm`	RPM Package Manager
`.tar`	Tape Archive
`.tar.bz2`	bzip2 compressed Tape Archive
`.tar.gz`	gzip compressed Tape Archive
`.tgz`	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 attachment to the software node:

```
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 “Packages Created by the --build Command” on page 15 of this guide.)

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 Opsware node 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 node at the same level as the previous version.

Unix:

```
ismtool --opswpath 'Application Servers/{$NAME}/{$VERSION}'
```

Windows:

```
ismtool --opswpath "Application Servers/{$NAME}/{$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 55.

4. Build and upload the `foo-1.2.8` ISM with the ISMTool.
5. In the Opsware Command Center, deprecate the `foo-1.2.7` packages.
6. Update templates to include the new node.
7. Reconcile managed servers against the new templates.

Chapter 3: ISM Scripts

IN THIS CHAPTER

This chapter contains the following topics:

Overview of ISM Scripts

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

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 Scripts - 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 tab of a package in the Opsware Command Center.

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 Opsware Command Center, 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 Opsware Command Center, uninstall the application. During the uninstallation, the pre-uninstallation and post-uninstallation hooks are executed on the managed server.

The process for creating and running installation scripts is different. For more information, see "Installation Scripts" on page 37

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 29.)

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.

The following sections of this guide list the default installation hooks created by the --build command:

- "Default Installation Hooks for Unix" on page 26
- "Default Installation Hooks for Windows" on page 27

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 38.

Installation Hook Functions

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

Table 3-1: 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)

Table 3-1: Installation Hook Functions

INSTALL HOOK	COMMON FUNCTIONS
`ism_pre_uninstall`	call `ism_stop` control script (to stop a server)
`ism_post_uninstall`	do any required clean up

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.

End-users can run control scripts from the Control window of the Opsware Command Center. (For more information, see "Control Scripts and Intelligent Software Modules" in the *Opsware® SAS 5.2 User's Guide*.) Advanced end-users can run a control scripts from the command-line in the Opsware 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 Control window of the Opsware Command Center. The Action field of the 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. Unlike Unix, on Windows the control script name is not case sensitive. For example, on Windows, `ism_start.cmd` is the same as `ISM_START.CMD`.

Control Script Functions

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

Table 3-2: 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:

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

To change the default directory, specify the `--ctlprefix` option with ISMTool.

Dynamic Configuration with ISM Parameters

The ISM `parameter` utility enables control scripts and installation hooks to access the values of Opsware 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 Opsware object such as a facility, customer, server, or server group.

Set with the Opsware Command Center, 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 Control window of the Opsware Command Center, an end-user can view the source (Opsware 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 Opsware Command center, install the application contained in the ISM onto a managed server.
5. In the Opsware Command Center, create a custom attribute with the same name as the parameter.
6. In the Opsware Command Center, run the control script on the managed server. At runtime, the script retrieves the parameter (control attribute) value from Opsware SAS.

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 3-3: 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.

For Opsware SAS 5.x, 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
```

For Opsware SAS 4.x, the `parameters` utility has the following syntax:

```
parameters [options]
--scope     <scope>  ; server|customer|facility|software|os|
                     ; install|default (default is all)
--scope     group          ; The 'group' scope needs to use
                           ; --groupname and --grouptype
     --groupname <name>    ; Group name to search
     --grouptype <type>    ; Group type to search
```

```
-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 Opsware SAS. 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 35.

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, each parameter is output in the form of name=value (one per line). The Windows `FOR` command sets each parameter as an environment variable. Finally, the parameters are passed to an NTP configuration script named `WindowsNTPConfigureScript.cmd`.

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

Search Order for Custom Attributes

With the Opsware Command Center, 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.opsware.com`, and you could set the same custom attribute to 80 for the Widget Corp. customer, which is associated with the `foo.opsware.com` server. At runtime, if a control script on `foo.opsware.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.

Opsware SAS 5.x Search Order

For version 5.x, the search order for custom attributes is as follows:

1. Server
2. Server Group
3. Customer
4. Facility
5. Service Level
6. Operating System
7. ISM Node (created during the upload operation)
8. Applications-> Other Applications
9. Applications-> Web Servers

10 Applications-> Application Servers

11 Applications-> Database Servers

12 Applications-> System Utilities

13 Applications-> Operating System Extras

Multiple server 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. Multiple software nodes are searched alphabetically by full node path.

For server group searches, custom attributes from attached nodes are not searched. Also, a server group that is a subgroup does not inherit the custom attributes of its parent group.

Opsware SAS 4.x Search Order

For version 4.x, the search order for custom attributes is as follows:

1 Server

2 Customer

3 Facility

4 Service Level

5 Applications-> OS Extras

6 Applications-> System Utilities

7 Applications-> Database Servers

8 Applications-> Application Servers

9 Applications-> Web Utilities

10 Applications-> Other Applications

11 Operating Systems

Multiple service levels are searched alphabetically by the full path name of the service levels, for example:

```
/ ServiceLevel / foo
/ ServiceLevel / zoo
```


If a managed server is attached to multiple nodes within the same software stack (category), then the search order is determined by the node install order. If the node install order is not set, then the nodes are searched alphabetically by the full path name of the nodes, for example:

```
/ Application Servers / JBoss /
/ Application Servers / WebLogic /
```

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 3-4: Differences Between Installation Scripts and Hooks

INSTALLATION SCRIPTS	INSTALLATION HOOKS
Displayed by the Properties tab of the package in the Opsware Control Center.	Displayed by the Contents tab of the package in the Opsware Control Center. (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 Opsware 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, Opsware SAS 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_post_configure` (control script).
8. Runs `ism_post_start` (control script).

9. Runs `pstinstallscript` (installation script).

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

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

Chapter 4: ISMTool Commands

IN THIS CHAPTER

This chapter discusses the following topics:

- ISMTool Argument Types
- Informational Commands
- Creation Commands
- Build Commands
- Opsware Interface Commands
- Environment Variables

ISMTool Argument Types

Table 4-1defines 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 4-1: 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`

Table 4-1: ISMTool Argument Types

ARGUMENT TYPE	DESCRIPTION	EXAMPLE
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'`

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

Displays the values currently specified for node meta data.

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
```

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 bugs or modify how it operates on an extracted ISMDIR. If the version of the currently installed ISMTool is different than 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 4-2 lists the developer actions if the currently installed and previous versions of ISMTool are not the same.

Table 4-2: ISMTool Upgrade Actions

ISMTOOL VERSION CURRENTLY INSTALLED	ISMTOOL VERSION THAT CREATED 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 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 Unix, the default value of `PATH` is `/usr/local`.

--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`. On Windows the prefix must be in this form: `driveletter:\directoryname` (for example, `D:\mydir`). 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 attached to the node 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 Opsware-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 which 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” on page 71.

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.

Opsware Interface Commands

This section describes the ISMTool commands that interact with Opsware SAS.

--upload

Upload the ISM contained in the ISMDIR to the Opsware core. During the upload process, ISMTool creates the Opsware software node with the path specified by `--opswpath`. To specify which Opsware core to connect to, use either command-line arguments (such as `--softwareRepository`) or the environment variables listed in Table 4-3.

The `--upload` command prompts for an Opsware user name and password. Generally, the user name and password should be those of the default Opsware administrator. To use another Opsware user for the upload, first run ISMUsertool to register the user. For more information, see "ISMUsertool" on page 69.

--opswpath STRING

Specify the path of the Opsware node associated with the uploaded ISM. Note that the Opsware path always contains forward slashes, even on Windows.

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 4-3 lists the variables recognized by the ISMTool.

Table 4-3: 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
${CUSTOMER}	Finance

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"
```

--dataAccesEngine HOST[:PORT]

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

--commandEngine HOST[:PORT]

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

--softwareRepository HOST[:PORT]

For the upload, use the Opsware 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 Opsware node.

--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 Opsware 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 Opsware 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 Opsware package control scripts during the upload. The package control scripts are registered in the Model Repository and are viewable from the Opsware Command Center.

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 Opsware 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.

{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 attach to the software node a passthru package specified by --addPassthruPkg. To attach 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 supported package types and values to use for {PkgType} are as follows:

Table 4-4: Allowed Values for PkgType

PACKAGE	ALLOWED VALUE FOR {PKGTYPE]
AIX LPP	lpp
HP-UX Depot	depot
RPM	rpm
Solaris Package	solpkg
Solaris Patch	solpatch
Solaris Patch Cluster	solcluster
Windows Hotfix	hotfix
Windows MSI	msi
Windows Service Pack	sp
Windows ZIP File	zip

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

```
% 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 attached to the `--opswpath` node, it will be removed.

Note that an ISM remembers all packages that have been removed as a passthru package. If a package was attached to the ISM node via the OCC or a previous upload operation, the attachment will be removed on the next upload operation.

--attachPkg {PkgName} --attachValue BOOLEAN ISMNAME

Specify whether a package managed by an ISM should be attached to the node identified by `--opswpath`.

By default, when a control or application package is built, these packages are marked for attachment to the node. However passthru packages and inner packages are not marked for attachment 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 attachment. If `--attachValue` is false, a package is marked for detachment. A package is attached or detached during an `--upload` operation. The following table lists the package types that can be attached to a node.

Table 4-5: Package Type Properties

PACKAGE TYPE	CAN THIS PACKAGE TYPE CONTAIN SCRIPTS	CAN THIS PACKAGE TYPE BE ATTACHED TO A NODE?
AIX LPP	no	no
AIX Base Fileset	yes	yes
AIX Update Fileset	yes	yes

Table 4-5: Package Type Properties

PACKAGE TYPE	CAN THIS PACKAGE TYPE CONTAIN SCRIPTS	CAN THIS PACKAGE TYPE BE ATTACHED TO A NODE?
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
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 given node meta data property.

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

Table 4-6: Allowed values for {PathProp}

{PATHPROP} ALLOWED VALUE	{PROPVALUE} TYPE	EXAMPLE
`description`	TEXT	`'This does something important'`
`notes`	TEXT	`'And so does this'`
`allowservers`	BOOLEAN	`false`

The following example commands show how to set the `description` and `allowservers` properties:

```
% ismtool --addPathProp description --propValue 'This node does
nothing' ISMNAME
% ismtool --addPathProp allowservers --propValue true ISMNAME

% ismtool --showPathProps ISMNAME
description:   This node does nothing
notes:         None
allowservers:  true
```

--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 4-7: Allowed values for {PkgProp}

{PKGPROP} ALLOWED VALUE	DESCRIPTION	{PROPVALUE} TYPE
`deprecated`	Deprecated status for package	BOOLEAN
`description`	Description for package	TEXT
`endonpreiscriptfail`	Reconcile ends on pre-install script failure	BOOLEAN
`endonpreuscriptfail`	Reconcile ends on pre-uninstall script failure	BOOLEAN
`endonpstiscriptfail`	Reconcile ends on post-install script failure	BOOLEAN
`endonpstuscriptfail`	Reconcile ends on post-uninstall script failure	BOOLEAN
`installflags`	Install flags for package	TEXT
`notes`	Notes for the package	TEXT
`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 OCC. In addition, the following table lists {PkgProp} values supported by specific package types.

Table 4-8: {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	Reconcile ends on post install script failure	BOOLEAN
`preuninstallscriptfilename`	Windows ZIP	Pre uninstall script filename	STRING
`preuninstallscriptfilenamefail`	Windows ZIP	Reconcile 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
```


Environment Variables

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

CRYPTO_PATH

This environment variable indicates the directory that contains the file `ismtool/token.srv`.

`CRYPTO_PATH` and `token.srv` are required only if you are uploading the ISM from a server that is not managed by Opsware SAS (that is, a server that has no Opsware agent).

To connect to the Opsware core during the upload of an ISM, the ISMTool needs the client certificate and key that were generated during the installation of Opsware SAS. The name of the certificate is `token.srv` and it is inside the `opsware-cert.db` that is generated during install. Ask your Opsware Administrator for this certificate.

Keep in mind that using this certificate with the ISMTool invokes a different security mechanism than the one used by the Opsware Command Center. As a result, you might have increased or reduced permissions. You might have access to servers belonging to customers that you normaly do not have access to. Also, you might be able to perform operations that you cannot perform with the Opsware Command Center. Therefore, use the ISMTool with caution to avoid unintended consequences caused by a possible change in security permissions.

After you get the `token.srv` file, on the ISM development machine, copy the file to the following directory:

```
/<some-path>/ismtool
```

The `<some-path>` part of the directory path is your choice, but the subdirectory containing `token.srv` must be `ismtool`. Next, set the `CRYPTO_PATH` environment variable to `<some-path>`, the directory above `ismtool/token.srv`. For example, in `csh` you might copy the file and set the environment variable as follows:

```
% mkdir /home/buzz/dev/ismtool
% cp token.srv /home/buzz/dev/ismtool
% setenv CRYPTO_PATH /home/buzz/dev
```

On Windows, you might might use these commands:

```
mkdir \buzz\dev\ismtool
set CRYPTO_PATH=C:\buzz\dev
copy token.srv \buzz\dev\ismtool
```

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.

ISMTOOLCE

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

ISMTOOLCUSTOMER

This environment variable is a STRING that specifies the Opsware customer during an ISMTool upload.

ISMTOOLDA

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

ISMTOOLPASSWORD

This environment variable is a STRING that specifies the Opsware 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 Opsware Software Repository used by the ISMTool.

ISMTOOLUSERNAME

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

Appendix A: ISMUsertool

IN THIS APPENDIX

This appendix provides a brief overview of the ISMUsertool.

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

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.

The `--upload` command of ISMTool also prompts for the Opsware customer. For users that are granted upload privileges using ISMUsertool, the only Opsware customer allowed is Customer Independent. The `admin` user can specify any customer defined in the Opsware Command Center.

Appendix B: Platform Differences

IN THIS APPENDIX

This appendix discusses the IDK differences for the following platforms:

- Solaris
- Windows

Solaris

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

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 reconcile process to work correctly.

