



Opsware® System Intelligent Software Module (ISM) Development Kit 1.0.6 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

Copyright © 2000-2005 Opsware Inc.

Opsware Inc. Confidential Information.

NOT for Redistribution. All Rights Reserved.

Opsware, Opsware Command Center, Model Repository, Data Access Engine, Web Services Data Access Engine, Software Repository, Command Engine, Opsware Agent, Multimaster Replication Engine, and Code Deployment & Rollback are trademarks and service marks of Opsware Inc. All other marks mentioned in this document are the property of their respective owners.

The Opsware System is protected by US and international copyrights and patents pending.

Table of Contents

Preface	vii
About this Guide	vii
Contents of this Guide	vii
Conventions in this Guide	viii
Icons in this Guide	ix
Guides in the Documentation Set and Who Should Read Them	ix
Chapter 1: ISMTool Overview	1
About ISMTool	1
ISMTool Quick Start	2
Build Process	3
ISM File System Structure	5
Chapter 2: ISM Packages	9
ISM Application Package	9
Source Transformation (UNIX only)	9
Binary Archive to Package Transform	13
File Attribute Control (UNIX only)	13
ISM Control Package	15

ISM Packaging Scripts	15
Default Packaging Scripts for UNIX	17
Default Packaging Scripts for Windows	18
ISM Control Scripts	19
ISM Parameter Interface	20
Chapter 3: ISMTool Commands	25
ISMtool Argument Types	25

Informational Commands	26
--help	26
--env	26
--myversion	27
--info ISMDIR	27
--showParams ISMDIR	27
Creation Commands	27
--new ISMNAME	27
--pack ISMDIR	28
--unpack ISMFILE	29
Build Commands	29
--quiet	30
--verbose	30
--banner	30
--clean	30
--build	30
--upgrade	31
--name STRING	32
--version STRING	32
--prefix PATH	33
--ctlprefix PATH	35
--user STRING (UNIX only)	35
--group STRING (UNIX only)	35
--ctluser STRING (UNIX only)	35
--ctlgroup STRING (UNIX only)	35
--pkgengine STRING (UNIX only)	36
--ignoreAbsolutePaths BOOL (UNIX only)	36

--addCurrentPlatform (UNIX only)	36
--removeCurrentPlatform (UNIX only)	36
--addPlatform TEXT (UNIX only)	36
--removePlatform TEXT (UNIX only)	36
--target STRING (UNIX only)	37
--skipControlPkg BOOL	37
--skipApplicationPkg BOOL	37
--chunksize BYTES (UNIX only)	37
--solpkgMangle BOOL (SunOS only)	38
Opware Interface Commands	38
--upload	38
--opswpath STRING	38
--dataAccessEngine HOST[:PORT]	39
--commandEngine HOST[:PORT]	39
--softwareRepository HOST[:PORT]	39
--description TEXT	40
--addParam STRING	40
--paramValue TEXT	40
--paramType PARAMTYPE	40
--paramDesc TEXT	40
--removeParam STRING	40
--rebootOnInstall BOOL	41
--rebootOnUninstall BOOL	41
--registerAppScripts BOOL (Windows only)	41
--endOnPreIScriptFail BOOL (Windows only)	41
--endOnPstIScriptFail BOOL (Windows only)	41
--endOnPreUScriptFail BOOL (Windows only)	42

--endOnPstUScriptFail BOOL (Windows only)	42
Environment Variables	42
CRYPTO_PATH	42
ISMTOOLBINPATH	43
ISMTOOLCE	43
ISMTOOLCUSTOMER	44
ISMTOOLDA	44
ISMTOOLPASSWORD	44
ISMTOOLSITEPATH	44
ISMTOOLSRL	45
ISMTOOLUSERNAME	45
Appendix A: ISMUsertool	47
Appendix B: Platform Differences	49
Solaris	49
Windows	50

Preface

Welcome to Opsware System 4.7 – 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 System 4.7 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 and control scripts. (An ISM is similar to a Linux RPM or Solaris package.) 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.

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.

Contents of this Guide

This guide contains the following chapters:

Chapter 1: ISMTool Overview

Chapter 2: ISM Packages

Chapter 3: ISMTool Commands

Chapter 3: ISMTool Commands

Appendix A: ISMUserTool

Appendix B: Platform Differences





Conventions in this Guide

This guide uses the following typographical and formatting conventions.

NOTATION	DESCRIPTION
Bold	Defines terms.
<i>Italics</i>	Identifies guide titles and provides emphasis.
Courier	Identifies text of displayed messages and other output from Opsware System programs or tools.
Courier Bold	Identifies user-entered text (commands or information).
<i>Courier Italics</i>	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.

Guides in the Documentation Set and Who Should Read Them

The *Opsware System 4.7 Administration Guide* is intended to be used by the Opsware administrator who is responsible for performing the tasks described herein.

The *Opsware System 4.7 User's Guide* is intended to be used by system administrators who perform the day-to-day functions of managing servers, provisioning operating systems, uploading packages, setting up software trees and node hierarchies, attaching software applications and installing them on servers, managing patches, reconciling servers with software, creating and executing scripts, tracking configuration, and deploying and rolling back code and content.

The *Opsware System 4.7 Installation Guide* is intended to be used by system administrators who are responsible for the installation of Opsware System 4.7 in a facility. It documents how to run the Opsware System Installer and how to configure each of the components.

Chapter 1: ISMTool Overview

IN THIS CHAPTER

This chapter contains the following topics:

- About ISMTool
- ISMTool Quick Start
- Build Process
- ISM File System Structure

About ISMTool

Within a large organization, it is usually the responsibility of a small standards team to define the installation and configuration details of software that is used within a data center. Without access to automation, such a team usually has a difficult time implementing the standards due to the rising number of servers in the data center environment.

With the Opsware System, the standards team can define software packages and configurations, which are then tested and uploaded to the Opsware core in preparation for distribution throughout the data center. To support centralized management and to leverage the packaging effort, the Opsware System supports package configuration based on local conditions at installation time as well as reconfiguration at a later time. A key element to this feature is the Intelligent Software Module (ISM), a set of directories and files which contain the raw bits of the application as well as scripts controllable by the Opsware System. These scripts can be run during initial configuration, reconfiguration, application start, application stop, and application-specific actions.

The ISMtool is a command line program which manages the creation of ISMs for use with the Opsware System. The ISMtool can support creation of ISMs from source (for example, Apache, OpenSSH, NTP) or enterprise applications that use an application specific binary installer. After creation of an ISM, the ISMtool supports the uploading of the ISM to the Opsware core so that the ISM is immediately available for installation onto managed servers.

The ISMtool leverages existing packaging technology (in other words, RPM, MSI, Solaris Packages, HP-UX depot, and AIX LPP). This decision was made for two reasons. First, developers should not have to learn yet another packaging technology. Second, operational problems are rarely the fault of a packaging technology. Problems are usually caused by a lack of standards or weak enforcement of those standards. An operational standard may state, "If an operating system is installed in DC1 through DC6 then the Network Time Protocol service should point to time servers at `ntp.dc1.big.com` through `ntp.dc6.big.com` respectively". Although this operational rule is simple, it would be difficult to implement without the Opware features that provide a way to deliver and control NTP services throughout a distributed data center environment.

ISMTool Quick Start

In general, the ISMtool operates on the extracted form of the ISM. The extracted form is a directory structure containing special subdirectories used by the ISMtool to manage builds and to store built packages, temporary files, and its logs. ISMs are self-contained units, separate from the ISMtool itself. This separation allows the ISMtool to archive the extracted ISM directory into a portable ZIP file for transport to another location. The life-cycle of an ISM is shown in the below ISMtool commands.

UNIX:

```
location1% ismtool --new foobar
location1% ls
foobar/
location1% ismtool [options] foobar
location1% ismtool --pack foobar
location1% ls
foobar/   foobar-1.0.0.ism
```

Windows:

```
C:\location1>ismtool --new foobar
C:\location1>dir
11/21/2003  10:17a      <DIR>          foobar
C:\location1>ismtool [options] foobar
C:\location1>ismtool --pack foobar
C:\location1>dir
11/21/2003  10:17a      <DIR>          foobar
11/21/2003  10:17a                1,927,339 foobar-1.0.0.ism
```

The `foobar-1.0.0.ism` file (which is just a ZIP archive) can be emailed or copied to another server. At the remote location a developer could unpack the ISM and continue working on it, as shown by the following commands.

UNIX:

```
location2% ls
foobar.ism
location2% ismtool --unpack foobar-1.0.0.ism
location2% ls
foobar/ foobar-1.0.0.ism
```

Windows:

```
C:\location2>dir
11/21/2003  10:17a                1,927,339 foobar-1.0.0.ism
C:\location2>ismtool --unpack foobar
C:\location2>dir
11/21/2003  10:17a                <DIR>          foobar
11/21/2003  10:17a                1,927,339 foobar-1.0.0.ism
```

Now the ISMtool can operate on the `foobar` ISM at the remote location, and the developer can upload it to the local Opsware installation.

UNIX:

```
location2% ismtool --build foobar
location2% ismtool --upload foobar
```

Windows:

```
C:\location2>ismtool --build foobar
C:\location2>ismtool --upload foobar
```

Note that all changes happen to the extracted ISM directory. The only ISMtool operation that will work against the ZIP form of an ISM is `--unpack`.

Build Process

The ISMtool manages the ISM creation process. The tool creates, modifies, builds, examines, and uploads an ISM to the Opsware core. Once built, an ISM has three components: an application package, a control package and an XML descriptor. The application package contains the raw bits of an application, but unlike traditional packaging systems, the application package does not contain the packaging scripts. (Example of these scripts include pre-install, post-install, pre-uninstall.) In an ISM, a separate control package contains the packaging scripts as well as the runtime interface

between the application package and the Opware System. Because the ISM packaging scripts are external to the application package, a developer could install an update to the control package (for example, to fix a bug in a packaging script) without reinstalling a running application. This flexibility is an important feature in 24 by 7 operational environments that run business-critical applications. The XML descriptor in an ISM is used by the Opware System when the ISM is uploaded. The descriptor provides information such as the name and default values for centrally managed configuration parameters, as well as the application control hooks (such as configure, reconfigure, start, stop). The application and control packages can be generated several formats, depending on the development platform. Currently supported formats are MSI, RPM, Solaris Data Stream and HP-UX depots, and AIX LPPs.

The ISMtool has a three-phase build process. In the first phase, which is for UNIX platforms, is optional. The ISMtool searches the `src` subdirectory of the ISM for components that need a build transformation. The most common type of transformation is the compilation of program source code. Another example is the running of a binary installer on a build machine to extract files necessary for packaging. As a result of the first phase, zero or more binary archives are placed into the ISM's binary archive (`bar`) subdirectory.

In the second phase of the build process, the ISMtool creates the application package from the archives it finds `bar` subdirectory. These archives may have been produced by the first phase of the build process, or the developer may directly copy the archives into the `bar` subdirectory. A developer can use this mechanism to quickly create an application package for a simple archive such as `zip`, `tar`, `cpio`, or `msi`. The resulting application package is placed into the ISM's package subdirectory (`pkg`).

In the third phase of the build process, the ISMtool creates the control package by packaging up the files inside the `ism` subdirectory of the ISM. The `ism` subdirectory contains the Opware runtime interface, the ISM control scripts (such as `configure`, `start`, `stop`), and the packing scripts (such as `pre-install`, `post-install`, `pre-uninstall`, `post-uninstall`). The control package is also placed into the `pkg` subdirectory.

The ISMtool supports several back-end packaging engines, all of which use a package name and version. Some engines also keep track of the release, which is really just a version augmentation. The ISMtool allows the version string of an ISM to be any string that is legal for the back-end packaging engine. Internally, however, the ISMtool needs to some way to keep track of changes to the application and control sources. The ISMtool does this by keeping track of two release numbers, one for the application package and the other for the control package. The ISMtool does recursive checksums of all data that

is used to create the output packages. If the tool notices any changes, it increments the appropriate internal release number. The expression of these release numbers in the final package depends on the packing technology. For most engines, the release number is appended to the developer-supplied version string. RPM supports a release field, which is used to store the appropriate release number. For example, given an ISM named `ntp` with version 4.1.2, an application release number of 3, and control release number 7, the output packages for RPM on an x86 would be named:

```
Applicaton Package:    ntp-4.1.2-3.i386.rpm
Control Package:      ntp-ism-4.1.2-7.i386.rpm
```

On Windows, the output MSI packages would be named:

```
Application Package:  ntp-4.1.2-3.msi
Control Package:      ntp-ism-4.1.2-7.msi
```

The release numbers are NOT designed to be editable by a developer. They are designed so that when a developer makes a bug fix to the control or application sources of an ISM the resulting packages, if changed, will have a different release. This allows a developer to update Opsware with the patched package using the `--upload` command.

ISM File System Structure

When an ISM is initially created using `--new`, the following directory structure is created.

UNIX:

```
% ismtool --new ntp-4.1.2
% ls ntp-4.1.2/
tmp/ log/ doc/ src/ bar/ ism/ pkg/
```

Windows:

```
% ismtool --new foobar
% dir foobar
11/21/2003  10:17a    <DIR>      .
11/21/2003  10:17a    <DIR>      ..
11/21/2003  10:17a    <DIR>      bar
11/21/2003  10:17a    <DIR>      doc
11/21/2003  10:17a    <DIR>      ism
11/21/2003  10:17a    <DIR>      log
11/21/2003  10:17a    <DIR>      pkg
11/21/2003  10:17a    <DIR>      src
11/21/2003  10:17a    <DIR>      tmp
```

An ISM has these subdirectories:

- `tmp` - Used as scratch space for ISMtool operations.
- `log` - Holds files which keep track of the output from source transformations (compilation or local installs), output from packaging engines such as `msi`, `rpm`, `pkgtrans`, `swpackage`, or an Opsware upload.
- `doc` - A location for optional developer supplied documentation as well as documentation generated automatically during ISM build.
- `src` - May optionally contain files that can control the compilation of sources into binary archives.
- `bar` - Contains binary archives, the contents of which are used to create the application package.
- `ism` - Contains all the files needed to create the control package of the ISM. The `ism` directory is where a developer can edit the default package hooks (i.e., pre-install, post-install, pre-uninstall, post-uninstall), as well as add ISM controls to the package.
- `pkg` - Contains the application and control packages, which are generated by the build.

The following listing shows the contents of the ISM subdirectories after an ISMtool build. 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 ISMtool build creates the files in the `log` and `tmp` subdirectories, in addition to any 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
  log/
    __ntp_0_app__.spec.log
    __ntp_ism__.spec.log
    upload.log
  doc/
    index.html
    index/
```

```
ntp-4.1.2-3.i386.rpm.html
ntp-ism-4.1.2-7.i386.rpm.html
tmp/
  macros
  __ntp_0_app__.spec
  __ntp_ism__.spec
  rpmrc
  BUILD/
  BUILDROOT/
  RPMS/
  SOURCES/
  SPECS/
  TEMP/
ism/
  ism.conf
  bin/
    ismget
    parameters
    platform
    python
  lib/
    <ism runtime>
  env/
    ism.sh
    ism.py
    ism.pl
  build/
    ism_pre
    ism_post
    ism_clean
  pkg/
    ism_check_install
    ism_post_install
    ism_post_uninstall
    ism_pre_install
    ism_pre_uninstall
  control/
    ism_start
    ism_stop
    ism_configure
    ism_reconfigure
```


Chapter 2: ISM Packages

IN THIS CHAPTER

This chapter contains the following topics:

- ISM Application Package
- ISM Application Package
- ISM Control Scripts
- ISM Parameter Interface

ISM Application Package

The ISMtool supports the creation of an application package either by scripted construction (compile from source or use of a binary installer) or by the direct packaging of one or more binary archives. The easiest way to use the ISMtool is to simply zip or tar up a directory, then place it in the ISM's binary archive directory (`bar`) and start a build with the `--build` command. More sophisticated ISMs can control the build of the application package from the source code (for example, NTP). The result from such a build will produce a binary archive, which is placed in the `bar` directory for packaging.

Source Transformation (UNIX only)

Every ISM has a source (`src`) subdirectory. The ISMtool recursively searches the `src` subdirectory for files ending in `.spec` (called specfiles). 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 a developer 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>

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, the developer would create a binary archive of the installer and copy it to the ISM's `bar` directory.

%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` directory.

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, the developer 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 the Maximum RPM document.

Binary Archive to Package Transform

After the source transformations have occurred, the ISMtool searches the internal `bar` directory for binary archives to package. Binary archives can be copied by the developer into the `bin` directory or they may be generated as `cpio` archives by the directives in the `%files` section of the specfile. Regardless of their origin, all files inside the binary archives in the `bar` directory are transformed into the ISM's application package. Warning: The ISMtool uses certain file extensions to figure out how to process an archive. Valid archive file extensions are shown in Table 2-1.

Table 2-1: Valid Binary Archive Types

FILE EXTENSION	ARCHIVE TYPE
<code>.cpio</code>	UNIX CPIO Archive
<code>.msi</code>	Microsoft Installer
<code>.rpm</code>	RPM Package Manager
<code>.tar</code>	Tape Archive
<code>.tar.bz2</code>	bzip2 compressed Tape Archive
<code>.tar.gz</code>	gzip compressed Tape Archive
<code>.tgz</code>	gzip compressed Tape Archive
<code>.zip</code>	Info-Zip compatible Zip

File Attribute Control (UNIX only)

Often, an application package should be installed using a certain UNIX user and group. Also, the individual files in an application package may require specific ownership and UNIX mode bit settings.

This task becomes more difficult if the developer does not have the permissions (`setuid root`) to create files on a development server with the needed ownership and mode bits. Packaging engines contain a variety of mechanisms to get around this difficulty. The ISMTool hides the complexity of these various mechanisms by providing a common way to specify the ownership and mode bits. The ISMTool allows the developer to control the meta information about each file that is packaged.

In the ISMtool, a tiered approach allows increasing amount of control. First, given no other information, the ISMtool extracts file ownership and mode bits from the binary archive itself. Sometimes, a developer may have an archive with files owned by a user not

intended to own files on the target system. To override the ownership of all files in all binary archives, the developer can use the `--user` and `--group` options described later in this document. If file-by-file control is needed, then the developer can add an additional file to the binary archive directory with the file-specific attribute information. The format of the attribute information is based on the `specfile` attribute language documented in Maximum RPM.

The developer indicates file-specific attributes as follows. Given a binary archive named `foobar.tar.gz`, the developer adds a file called `foobar.tar.gz.meta` to the `bar` directory. The ISMtool will process the `.meta` file as it would the `%files` section of a `specfile`. For example, suppose the developer begins with the follow TAR archive:

```
% 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/
ntp-timeset
-rwxr-xr-x root/root    19244 2001-09-05 03:54:43 usr/sbin/
ntp-trace
-rwxr-xr-x root/root     1019 2001-09-05 03:54:39 usr/sbin/
ntp-wait
```

If the developer wants the `etc/ntp` directory and the files inside it to be owned by the user and group `ntp/ntp`, then the following `.meta` file would be used:

```
% tar tvf ntp/bar/ntp.tar.meta
%attr(-,ntp,ntp) etc/ntp/
```

```
%attr(-,ntp,ntp) etc/ntp/step-tickers
%attr(-,ntp,ntp) etc/ntp/drift
%attr(-,ntp,ntp) etc/ntp/keys
```

ISM Control Package

The ISM control package contains the scripts that provide operational control for an application. The default install locations of ISM control packages are as follows.

UNIX:

```
/var/opt/OPSWism/<name>-<version>/
```

Windows:

```
%ProgramFiles%\OPSWism\<name>-<version>\
```

The `<name>` and `<version>` indicate the name and version of the installed ISM. The path to the control package can be changed using the `--ctlprefix` command. This location is critical since an application package, once built, will reference files inside this path to execute packaging scripts such as `ism_post_install`. Note that the ISM control path contains the name and version of the ISM but not the release number. The release numbers are managed by the ISMtool to control rebuilding of either the application or control packages of an ISM. The application package would refer to the generic path `/var/opt/OPSWism/<name>-<version>/` (UNIX) or `%ProgramFiles%\OPSWism\` (Windows). For example, a control package with the name `ntp`, version of `4.1.2`, and release number `7` would have an RPM package name of `ntp-ism-4.1.2-7.i386.rpm` or an MSI name of `ntp-ism-4.1.2-7.msi`. However, the files would install into `/var/opt/OPSWism/ntp-4.1.2/` or `%ProgramFiles%\OPSWism\`, respectively. This separation allows the control package to be patched without requiring a reinstall of the application package.

An ISM exposes two types of developer extensions: packaging hooks and control hooks. The key to centralized control involves fetching control variables (ISM parameters) from a central store (Opsware) when a control action is invoked via Opsware or the command line. The developer interface to fetching control variables is the ISM `parameters` command, discussed further in the section “ISM Parameter Interface” on page 20.

ISM Packaging Scripts

When an ISM application package is built, the ISMTool automatically generates the following packaging scripts. These scripts can be used as is or edited by the developer.

UNIX:

```
check_install
pre_install
post_install
pre_uninstall
post_uninstall
```

Windows:

```
pre_install.cmd
post_install.cmd
pre_uninstall.cmd
post_uninstall.cmd
```

For listings of the default packaging scripts, see these sections:

- Default Packaging Scripts for UNIX
- Default Packaging Scripts for Windows

In our current NTP example, these scripts would be installed into the following directories:

UNIX:

```
/var/opt/OPSWism/ntp-4.1.2/pkg/
```

Windows:

```
%ProgramFiles%\OPSWism\ntp-4.1.2\pkg\
```

The `post_install` (`post_install.cmd` on Windows) script, for example, will run on the managed server immediately after the application in the ISM has been installed. The `post_install` script will call into the following control directory.

UNIX:

```
/var/opt/OPSWism/ntp-4.1.2/control/
```

Windows:

```
%ProgramFiles%\OPSWism\ntp-4.1.2\control\
```

In the default `post_install` script, if the control scripts called `ism_configure` and `ism_start` exists, they will be called. These control scripts are only operational suggestions. A developer could modify `post_install` to call other scripts, if required. Note that there are no default control scripts. Unlike the packaging scripts, the control scripts must be provided by the developer.

On UNIX, the environment variable `ISMDIR` is defined and written to the file `ism.sh` (Python, and Perl variants exists as well.) On Windows, the file is named `ism.cmd`. When the packaging scripts are called, the full path of `ism.sh` or `ism.cmd` is passed as the first argument to the packaging scripts. This allows the scripts to be written in a version-independent way. In the current example, the value of `ISMDIR` on UNIX would be `/var/opt/OPSWism/ntp-4.1.2`.

Some packaging engines support a `check_install` hook directly; others do so implicitly via the `preinstall` hook. The ISMtool will map the `check_install` feature onto the backend packaging engine. If the `check_install` script returns a non-zero return code, the install is halted.

Default Packaging Scripts for UNIX

The default `ism_pre_install` script:

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

The default `ism_post_install` script:

```
#!/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` script:

```
#!/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_uninstall.sh` script:

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

Default Packaging Scripts for Windows

The default `ism_pre_install.cmd` script:

```
@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` script:

```
@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` script:

```

@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_uninstall.cmd` script:

```

@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

```

ISM Control Scripts

The packaging scripts should be brief. Most of the work should be done by the control scripts. An ISM can have any number of control scripts. The control scripts should focus on operationally relevant events for the application such as start, stop, configure, reconfigure, hardstop, clearlogs, and dumpcache. The packaging scripts focus on which of these controls should be called during packaging events such as post-install and pre-uninstall.

During a build, the ISMtool searches the `ism/control` directory for files that are executable and begin with `ism_`. All such files are assumed to be for control and the file system location (inside the control package) will be available to Opsware via the ISM's XML descriptor. This allows Opsware to expose the relevant application controls for that ISM.

Many application controls are independent of centralized Opsware configuration modeling. For example a control such as `ism_start` or `ism_stop` would call the application start and stop mechanisms in an operationally safe way (that is, isolating unintentional file descriptor forking). Such controls could be run centrally by Opsware but the controls simply act on the local application; they do not talk back to Opsware.

Application configuration management is another story. To centrally model configuration and apply it to classes of similar servers, an application's configuration control needs access to the centrally managed configuration parameter model.

ISM Parameter Interface

Opsware contains a hierarchal modeling system for organizing operation information. Nodes in this model may represent hardware, software, or other operational information. For example, the location of the software model of NTP 4.1.2 compiled for Redhat Linux 7.2 might be found at this Opsware node:

```
/System Utilities/NTP/4.1.2/Redhat Linux 7.2
```

The operational aspects of servers are controlled through their association with nodes within the Opsware model. Such associations can specify which organization owns the server, the facility where the server resides, the hardware type, the installed OS, the installed applications, and other characteristics. An important Opsware feature of this hierarchal model is inheritance. One type of information that can be inherited is a data type called a custom attribute. A custom attribute is a name-value pair that holds meta information for a server. A common use of custom attributes is to hold configuration parameters for an application. For example, a custom attribute could designate the port number of a particular service.

By using the `--addParam` command of the `he ISMtool`, the developer can add to an ISM a list of configuration parameters. Each parameter is a tuple of four fields: name, type, default value, and description. After the parameters have been added, the developer can write an ISM control script that accesses the parameters. The supported scripting languages are Bourne Shell, Korn Shell, Windows command shell, Python, and Perl. The shell scripts access the parameters through environment variables. Python scripts and Perl scripts use hash tables (dictionaries and maps, respectively).

For example, the developer could write a script named `ism_configure` (UNIX) or `ism_configure.sh` (Windows) that retrieves from Opsware the value for the custom attribute whose name matches the parameter. If the parameter does not have a value in Opsware for that server, then the default value, as defined by the developer, is retrieved locally from the `ism.conf` file of the control package.

The ISMtool formalizes the mechanics and precedence of custom attribute retrieval. The interface to parameter fetching is a command-line program named `parameters`, which resides in the binary directory of an ISM's control package. It is important to note that ONLY parameters defined using the `--addParam` command are made available via the `parameters` interface. The `parameters` program has the following command-line help:

```
% ./parameters --help
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 next example is a configuration control script written in Bourne Shell for the NTP time service on UNIX:

```
#!/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()
```

In the preceding two examples, the parameters `NTP_CONF_TEMPLATE` and `NTP_SERVER`, have been defined for the NTP ISM. The `$ISMDIR/bin/parameters` interface retrieves the two parameters, either from Opsware or from the local `ism.conf` file. The output of `$ISMDIR/bin/parameters` is a text string which can be evaluated to give the caller access to the parameter names and values. Depending on the language, the result of this evaluation either inserts the parameters into the environment or instantiates them as a hash table.

If the `$ISMDIR/bin/parameters` interface encounters an error while retrieving the parameters, it returns a special parameter named `_OPSW_ISMERR`. The `_OPSW_ISMERR` parameter contains a brief description of the error encountered.

The following example shows a configuration control script for Windows. In this example, `%ISMDIR%binparameters.cmd` retrieves the parameters from Opsware or from the local `ism.conf` file. Each parameter is output as a name-value pair in the form of `name=value` (one per line). The script then uses the Windows `FOR` command to set 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
```

In summary, the Opsware system contains hierarchies of software nodes. Most nodes can have custom attributes attached. When an ISM is uploaded to the Opsware core with the `--upload` command of the `ISMTool`, a node is created with the characteristics defined by the ISM. The path to the node is specified via the `--opswpath` command. For

example, the path could be set to `/System Utilities/ntp/4.1.2/Redhat Linux 7.2`. The ISMtool would attach to this node the application package, the control package, the default control parameters, and the ISM XML descriptor.

Chapter 3: ISMTool Commands

IN THIS CHAPTER

This chapter contains the following topics:

- ISMtool Argument Types
- Informational Commands
- Creation Commands
- Build Commands
- Opware Interface Commands
- Environment Variables

ISMtool Argument Types

Table 3-2 defines the argument types that are used in the ISMTool commands defined in the rest of this chapter. The `ISMNAME` argument type, for example, is specified by the syntax of the ISMTool `--new` command.

Table 3-2: ISMTool Argument Types

ARGUMENT TYPE	DESCRIPTION	EXAMPLE
PATH	Absolute file system path.	<code>/foo/bar</code>
STRING	Text string with no spaces.	<code>foobar</code>
TEXT	Arbitrary quoted text.	<code>'This is some text'</code>
BOOL	Boolean.	<code>true</code> or <code>false</code> <code>0</code> or <code>1</code>
ISMFILE	Path to a valid <code>.ism</code> file in the file system. This file would unpack into an <code>ISMDIR</code> .	<code>/foo/bar/name.ism</code>

Table 3-2: ISMTool Argument Types

ARGUMENT TYPE	DESCRIPTION	EXAMPLE
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 back-end 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'.	'String' 'Template'

Informational Commands

--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

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

Creation Commands**--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`. For example.

UNIX:

```
% ismtool --new tick
% ismtool --version 3.14 tick
18
% ls
tick/
% mv tick spoon
% ls
spoon/
% ismtool --pack spoon
% ls
spoon/ 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
spoon/ tick-3.14.ism
% rm -rf spoon
% ls
tick-3.14.ism
% ismtool --unpack tick-3.14.ism
% ls
spoon/ 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

All of the build commands apply to an extracted ISMDIR:

```
% ismtool <build commands> ISMDIR
```

Most of the build commands make persistent configuration changes to an ISM, which are recorded in the ISM's `ism.conf` file. However, a few commands change only the current command execution (for example, `--verbose`, `--quiet`).

Multiple command actions can be specified on the same command-line. In the following UNIX example, the command changes the 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 ISMDIR named `apache`.

```
% ismtool --pkgeengine 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
```

--quiet

Suppress the display of the ascii throbber eye-candy.

--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

Start a build of the ISM. Building an ISM is a multi-step process:

- 1** Check for platform compatibility.
- 2** Perform 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.
- 3** Run user-extensible cleanhook `<name>/ism/build/ism_clean`.
- 4** Verify the ISM runtime version. Upgrade or downgrade the runtime as needed.
- 5** Run a checksum on the application sources and increment the application release number if the current checksum does not match the previous checksum.

- 6** Run a checksum on the control sources (the contents of the `ism` directory) and increment the control release number if the current checksum does not match the previous checksum.
- 7** Run the user-extensible pre-build hook `<name>/ism/build/ism_pre`.
- 8** Run the source build (UNIX only). Recursively search for `.spec` files in the `<name>/src` directory, compiling and executing each.
- 9** Generate a script to transform the binary archives into the application package.
- 10** Generate a script to transform the `ism` directory into the control package.
- 11** Generate the control package.
- 12** Generate the application package.
- 13** Run a checksum on the generated packages.
- 14** Save the build meta data to the `ism.conf` file.
- 15** Generate the automatic HTML document `<ISMDIR>/doc/index/index.html`.

--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 3-3 lists the developer actions if the currently installed and previous versions of ISMtool are not the same.

Table 3-3: 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 <code>--upgrade</code> 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 <code>--upgrade</code> 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.

--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 back-end 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 -q lpv 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/
ntpdcc
-rwxr-xr-x 1 root root 66892 Sep 5 2001 /usr/sbin/ntpqq
-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/
ntptimeaset
-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/ntpdcc

```

```

-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/ntpdate
-rwxr-xr-x  1 root  root   40908 Sep  5  2001 /usr/local/
usr/sbin/ntpdate
-rwxr-xr-x  1 root  root   19244 Sep  5  2001 /usr/local/
usr/sbin/ntpdate

```

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 UNIX, the default is `/var/opt/OPSwism`. Windows the prefix must be in this form: `driveletter:\directoryname` (for example, `D:\mydir`).

--user STRING (UNIX only)

When creating the application package, change the user owner to `STRING`.

--group STRING (UNIX only)

When creating the application package, change the group owner to `STRING`.

--ctluser STRING (UNIX only)

When creating the control package, change the user owner to `STRING`. The default value is `root`.

--ctlgroup STRING (UNIX only)

When creating the control package, change the group owner to `STRING`. The default value is `bin`.

--pkgengine STRING (UNIX only)

Change the backend 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.

--ignoreAbsolutePath 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 `--ignoreAbsolutePath` 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 49.

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.

Opware Interface Commands

--upload

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

The `--upload` command prompts for an Opware user name and password. Generally, the user name and password should be those of the default Opware administrator. To use another Opware user for the upload, first run ISMUSERTOOL to register the user. For more information, see “ISMUSERTOOL” on page 47.

--opswpath STRING

Specify the path of the Opware node associated with the uploaded ISM. Note that the Opware 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 3-4 lists the variables recognized by the ISMtool.

Table 3-4: 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"
```

--dataAccessEngine 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 Opware 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 Opware 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 last package in the list is tagged.

--registerAppScripts BOOL (Windows only)

Register the ISM packaging scripts 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 Opware package control scripts during the upload. The package control scripts are registered in the Model Repository and are viewable from the Opware 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 Opware 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.

Environment Variables

The ISMtool references the following shell environment variables:

- CRYPTO_PATH
- ISMTOOLBINPATH
- ISMTOOLCUSTOMER
- ISMTOOLDA
- ISMTOOLPASSWORD
- ISMTOOLSITEPATH
- ISMTOOLSRL
- ISMTOOLUSERNAME

CRYPTO_PATH

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

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

Once you have obtained the `token.srv` and are ready to work with the ISMtool, please keep in mind that using this certificate with the ISMtool invokes a different security mechanism than the one used by the Opware Command Center. As a result, you may

have increased or reduced privileges and you may be able to perform operations on servers belonging to customers that you may not normally have access to, and you may be able to perform operations that you may not normally be able to perform from within the Opware Command Center. Consequently, use caution with the ISMtool to avoid any unintended consequences that could arise with this increased level of permissions.

After you get the `token.srv` file, create a directory on the development machine. The last two components of the directory must be `crypto/ismtool`. Here's a UNIX example:

```
% mkdir -p /some/random/absolute/path/crypto/ismtool
```

Copy the `token.srv` file into this directory:

```
% cp token.srv /some/random/absolute/path/crypto/ismtool
```

Finally, set the `CRYPTO_PATH` environment variable as follows:

```
% setenv CRYPTO_PATH /some/random/absolute/path/crypto
```

Note: Although the `token.srv` file is in the `ismtool` subdirectory, the value of `CRYPTO_PATH` does not include `ismtool`.

On Windows, the equivalent commands are:

```
mkdir %SomeRandomAbsolutePath%\crypto\ismtool\  
copy token.srv %SomeRandomAbsolutePath%\crypto\ismtool\  
set CRYPTO_PATH=%SomeRandomAbsolutePath%\crypto\  

```

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 compiled-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 Opware 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.

ISMTOOLSUR

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 Opware user name. By default, this user name is the Opware Administrator that is specified when the Opware System is installed. To enable other Opware users to perform an upload, run the ISMUsertool.

To list the users that have upload privileges:

```
% ismusertool --showUsers
```

To grant a user or multiple users upload privileges:

```
% ismusertool --addUser johndoe
% ismusertool --addUser janedoe --addUser jackdoe --
showUsers
```

ISMUsertool allows you to add or remove multiple users on the same command line, as well as show the list of users. To revoke upload privileges:

```
% ismusertool --removeUser johndoe --showUsers
% ismusertool --removeUser janedoe --removeUser jackdoe --
showUsers
```

You cannot use ISMUsertool to revoke the upload privileges of the default Opware Administrator user.

The `--upload` command of ISMtool also prompts for the Opware customer. For users that were granted upload privileges using ISMUsertool, the only Opware customer allowed is Customer Independent. The default Administrator user can specify any customer defined in the Opware 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  
SPROcpl  
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 `f00bar`, the package names would be the following:

```
ISMf00ba0  
ISMf00bac
```

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.

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.