Tivoli® IBM Tivoli Configuration Manager

Version 4.3.1

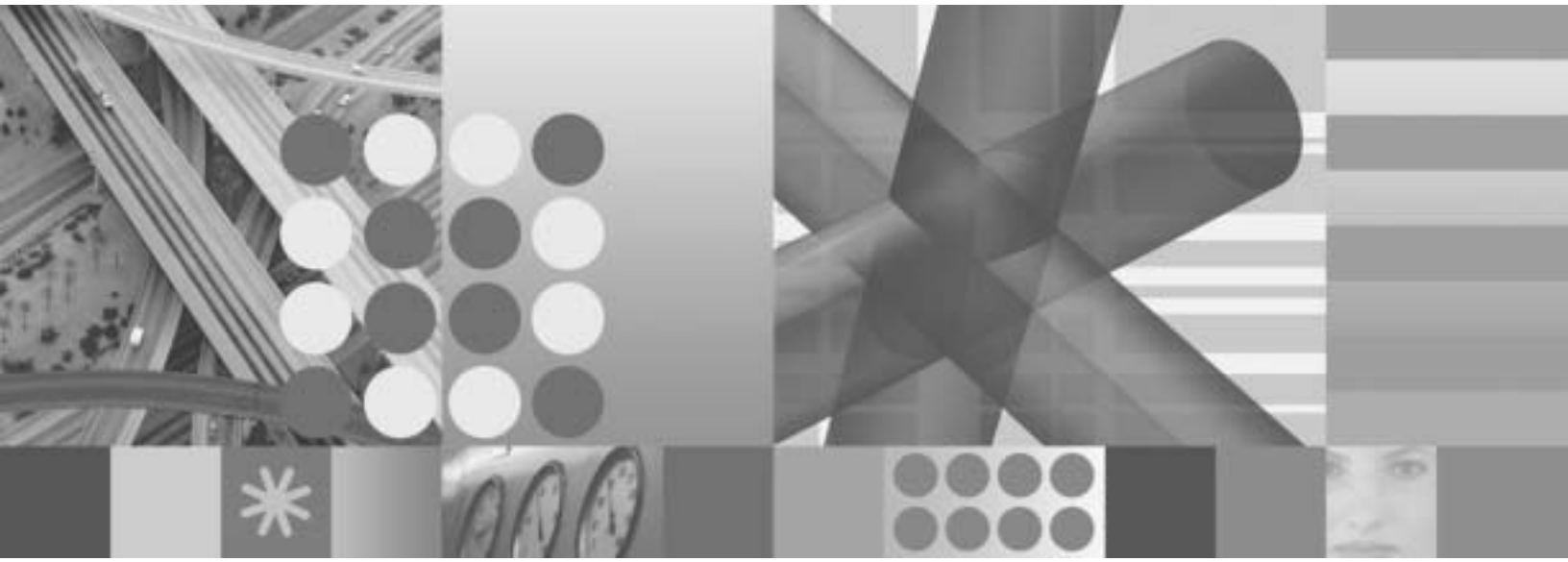
Reference Manual for Software Distribution

SC23-4712-05

Tivoli® IBM Tivoli Configuration Manager

IBM

Version 4.3.1



Reference Manual for Software Distribution

SC23-4712-05

Note

Before using this information and the product it supports, read the information in “Notices” on page 363.

This edition applies to version 4, release 3, modification level 1 of IBM Tivoli Configuration Manager (program number 5724-C06) and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC23-4712-04.

Contents

Figures

Tables

About This Guide

Software Distribution provides a means of managing and distributing software across a multi-platform network. For distributions that encompass wide area networks (WANs), Software Distribution has a built-in, WAN-smart capability that reduces inter-network traffic and ensures an efficient distribution.

This guide explains advanced features and concepts necessary for you to effectively use and tailor Software Distribution to fully meet your distribution needs

Who Should Read This Guide

The target audience for this guide is senior system administrators who intend to improve or customize Software Distribution functionality. You should have knowledge of the UNIX® operating system; concepts such as directories, files, and symbolic links; and the PC operating systems running on the systems to which you will distribute software. In addition, you should be familiar with Software Distribution and have used its advanced features.

What This Guide Contains

The *Reference Manual for Software Distribution* contains the following sections:

- Chapter 1, “Editing the Software Package Definition File,” on page 1

  Details how to edit and use the software package definition format and keywords.
- Chapter 2, “Performing Change Management Operations,” on page 143

  Explains concepts of change management operations and modes.
- Chapter 3, “Using Commands,” on page 157

  Provides the syntax statements, descriptions, and examples of the Software Distribution commands.
- Chapter 4, “Managing Policy,” on page 305

  Describes default and validation policy and how to define policy.
- Chapter 5, “Checking Object Consistency,” on page 333

  Describes how Software Distribution maintains database consistency if managed nodes or profiles are deleted or renamed.
- Chapter 6, “Migrating File Packages to Software Packages,” on page 335

  Describes the correlation between Software Distribution, Version 3.6 file packages and Software Distribution, Version 4.0 software packages.
- Appendix, Appendix A, “Built-in Variables,” on page 353

  Provides a list of the variables that are provided by Software Distribution.
- Appendix, Appendix B, “Support information,” on page 359

  Describes options for obtaining support for IBM® products.

Note: Information about troubleshooting for Software Distribution is provided in the *User’s Guide for Software Distribution*.

Publications

This section lists publications in the IBM Tivoli Configuration Manager library and related documents. It also describes how to access Tivoli® publications online and how to order Tivoli publications.

IBM Tivoli Configuration Manager Library

The following documents are available in the IBM Tivoli Configuration Manager library:

- *IBM Tivoli Configuration Manager: Introducing IBM Tivoli Configuration Manager*, GC23-4703

  Provides an overview of IBM Tivoli Configuration Manager and its components, as well as providing user scenarios to highlight various processes.
- *IBM Tivoli Configuration Manager: Planning and Installation Guide*, GC23-4702

  Explains how to install, upgrade, and uninstall the product and its components in a Tivoli environment.
- *IBM Tivoli Configuration Manager: User's Guide for Software Distribution*, SC23-4711

  Explains the concepts and procedures necessary for you to effectively use the Software Distribution component to distribute software over local area networks (LANs) and wide area networks (WANs).
- *IBM Tivoli Configuration Manager: Reference Manual for Software Distribution*, SC23-4712

  Explains advanced features and concepts needed to use and tailor the Software Distribution component.
- *IBM Tivoli Configuration Manager: User's Guide for Deployment Services*, SC32-0831

  Provides information about the Deployment Services of the product.
- *IBM Tivoli Configuration Manager: User's Guide for Inventory*, SC23-4713

  Describes the Inventory component and the management tasks that you can perform.
- *IBM Tivoli Configuration Manager: Database Schema Reference*, SC23-4783

  Provides information about the IBM Tivoli Configuration Manager repository.
- *IBM Tivoli Configuration Manager: Messages and Codes*, SC23-4706

  Details all the error, warning messages and error codes issued by all the components and services of the product.
- *IBM Tivoli Configuration Manager: Release Notes*, GI11-0926

  Contains late-breaking information about the product.
- *IBM Tivoli Configuration Manager: Patch Management Guide*, SC23-5263

  Describes a solution that covers the distribution and management of security patches and software updates in a Tivoli environment.
- *IBM Tivoli Configuration Manager: Guide for Active Directory Integration*, SC32-2285

  Describes the integration of Microsoft Active Directory with your Tivoli environment.
- *IBM Tivoli Configuration Manager: License Management Extension*, SC32-2260

  Describes the license management facilities provided in your Configuration Manager environment.
- *IBM Tivoli Configuration Manager: User's Guide for Operating System Deployment Solution*, SC32-2578

  Describes how you can implement an operating system deployment solution delivered with Configuration Manager.

Related Publications

The following documents also provide useful information:

- *IBM Tivoli Enterprise: Installation Guide*, GC32-0804

  Explains how to install and upgrade Tivoli Enterprise software within your Tivoli region using the available installation mechanisms provided by Tivoli Software Installation Service and Tivoli Management Framework.
- *Tivoli Management Framework: Planning for Deployment Guide*, GC32-0803

  Explains how to plan for deploying your Tivoli environment. It also describes Tivoli Management Framework and its services.
- *Tivoli Management Framework: Maintenance and Troubleshooting Guide*, GC32-0807

  Explains how to maintain a Tivoli environment and troubleshoot problems that can arise during normal operations.
- *Tivoli Management Framework: Reference Manual*, GC32-0806

  Provides in-depth information about Tivoli Management Framework commands. This guide is helpful when writing scripts that are later run as Tivoli tasks. This guide also documents default and validation policy scripts used by Tivoli Management Framework.
- *Tivoli Management Framework: User's Guide*, GC32-0805

  Describes the concepts and procedures for using Tivoli Management Framework services. It provides instructions for performing tasks from the Tivoli desktop and from the command line.
- *IBM Tivoli Configuration Manager: Warehouse Enablement Pack: Implementation Guide*

  Describes how to install and configure the warehouse enablement pack for the IBM Tivoli Configuration Manager product and describes the data flow and structures that are used by the warehouse pack.

The *Tivoli Software Glossary* includes definitions for many of the technical terms related to Tivoli software. The *Tivoli Software Glossary* is available at the following Tivoli software library Web site:

http://publib.boulder.ibm.com/tividd/glossary/tivoliglossarymst.htm

Accessing Publications Online

The documentation CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. Refer to the readme file on the CD for instructions on how to access the documentation.

The product CD contains the publications that are in the product library. The format of the publications is PDF, HTML, or both. To access the publications using a Web browser, open the infocenter.html file. The file is in the appropriate publications directory on the product CD.

IBM posts publications for this and all other Tivoli products, as they become available and whenever they are updated, to the Tivoli software information center Web site. Access the Tivoli software information center by first going to the Tivoli software library at the following Web address:

http://www.ibm.com/software/tivoli/library/

Scroll down and click the **Product manuals** link. In the Tivoli Technical Product Documents Alphabetical Listing window, click the **IBM Tivoli Configuration Manager** link to access the product library at the Tivoli software information center.

Note: If you print PDF documents on other than letter-sized paper, set the option in the **File → Print** window that allows Adobe Reader to print letter-sized pages on your local paper.

Ordering Publications

You can order many Tivoli publications online at the following Web site:

http://www.elink.ibmlink.ibm.com/public/applications/ publications/cgibin/pbi.cgi

You can also order by telephone by calling one of these numbers:
- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, see the following Web site for a list of telephone numbers:

http://www.ibm.com/software/tivoli/order-lit/

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For additional information, see the Accessibility Appendix in *IBM Tivoli Configuration Manager: User's Guide for Software Distribution*.

Tivoli Technical Training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site:

http://www.ibm.com/software/tivoli/education

Support Information

If you have a problem with your IBM software, you want to resolve it quickly. IBM provides the following ways for you to obtain the support you need:
- Searching knowledge bases: You can search across a large collection of known problems and workarounds, Technotes, and other information.
- Obtaining fixes: You can locate the latest fixes that are already available for your product.
- Contacting IBM Software Support: If you still cannot solve your problem, and you need to work with someone from IBM, you can use a variety of ways to contact IBM Software Support.

For more information about these three ways of resolving problems, see Appendix B, "Support information," on page 359.

Conventions Used In This Guide

This guide uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface Conventions

This guide uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations**:)
- Keywords and parameters in text

Italic

- Words defined in text
- Emphasis of words (words as words)
- New terms in text (except in a definition list)
- Variables and values you must provide

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating System-dependent Variables and Paths

This guide uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows® command line, replace **$***variable* with **%** *variable***%** for environment variables and replace each forward slash (*/*) with a backslash (\) in directory paths. The names of environment variables are not always the same in Windows and UNIX. For example, %TEMP% in Windows is equivalent to $tmp in UNIX.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. Editing the Software Package Definition File

Software Distribution enables you to create a software package in *software package definition (SPD) file* format by using the Software Package Editor graphical user interface (GUI). You can do this manually by using a text editor, or by exporting an existing software package and modifying it. An SPD file is a text file in ASCII format. This file consists of a signature and a sequence of stanzas, each of which describes objects, such as files, directories, and registry keys, and actions to be performed on these objects. You can edit the SPD file to change the characteristics of the software package, such as:

- Objects to be included in the software package, like files and directories.
- Actions to be performed on the target system, like adding Windows registry entries or removing OS/2 profile items. You can also define conditions under which an action should be performed.
- Processing and logging attributes, like replace_if_existing and stop_on_failure.
- Platform-specific attributes, like netware_attributes and unix_group.

Figure 1 illustrates the inheritance and containment relationships of the software package object model discussed in this chapter.

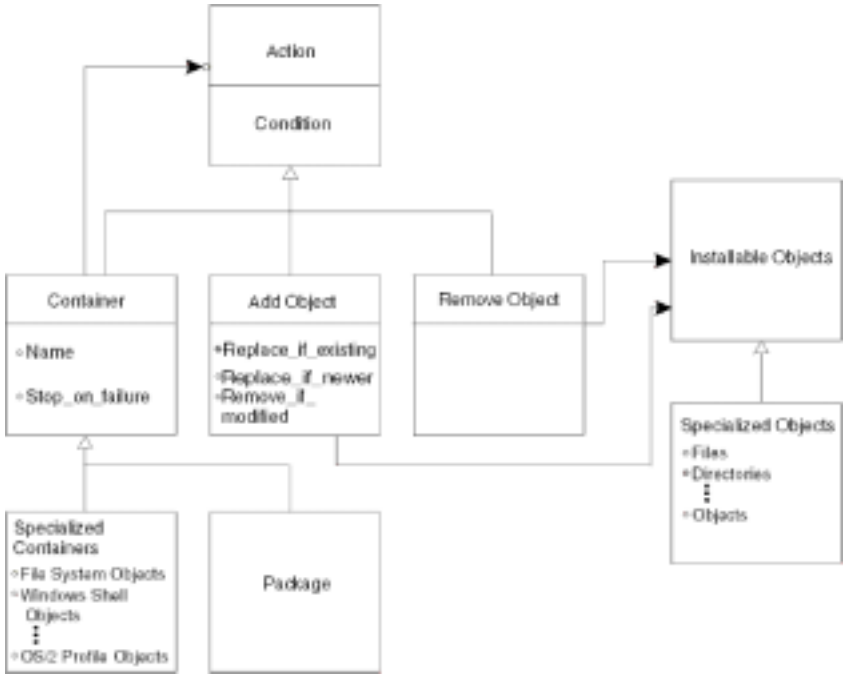


Figure 1. Software package object model

The following symbols are used in Figure 1:

- Boxes, which represent classes. Where applicable, a box is divided into two sections, where:
  - The top section is the class name
  - The bottom section represents meaningful attributes of the class
- Arrows, which, together with lines show the relationships between classes:
  - Empty arrows indicate an inheritance relationship between classes
  - Filled-in arrows indicate a containment relationship between classes

The classes illustrated here include the following:

Action
Performed on objects. Using conditions, you define the circumstances under which an action is performed.

Container
Inherits properties from the action class, but can also contain a list of actions to be performed. The example attributes shown include:

Name The identifier of the package, which is unique.

Stop_on_failure
Stops the execution of a package if an action fails.

Specialized container
A class of containers that are built into Software Distribution, including file system objects, Windows shell objects, and OS/2 profile objects.

Package
Contains references to the files and directories to be distributed, and instructions on how to distribute them. A package consists of one or more containers, each of which contains a series of actions.

Add object
An action that adds a specified object to the system. This class contains installable objects. The example attributes shown include:

Replace_if_existing
Specifies the replacement of an object that already exists on the target.

Replace_if_newer
Specifies the replacement of a target object (file or directory only) even if the target object is newer than the source object.

Remove_if_modified
Specifies the removal of an object even if the target object has been modified.

Remove object
An action that removes a specified object from the system. This class contains installable objects.

Installable object
The class of objects that can be installed by Software Distribution.

Specialized object
A class of objects that are built into Software Distribution, including objects such as files and directories.

These concepts, and how they are related to creating and maintaining software packages, are described in the following sections.

Software Package Name and Version

In the name and the version attribute in the SPD file, or in commands, you must separate the name from the version by inserting either of the following characters:
- Caret (^)
- Dot (.)

The length of the string that defines the name and version of a software package can vary depending on how you distribute it:

- If you use Activity Planner, the maximum length of the string must be 128 characters. It includes name, delimiter, version (64 characters), and `#region name`.
- If you do not use Activity Planner, the maximum length of the string must be 230 characters. It includes name, delimiter, and version (64 characters).

The string that defines the name and version can include more than one dot or caret or a combination of dots and carets. The following rules are used in interpreting the string:

- The string must include at least one caret or dot, otherwise the version is interpreted as a null string and this is invalid.
- Carets take precedence over dots as the name-version separator. Therefore, if a dot precedes a caret in the string, the dot is considered a part of the name.
- Moving from the left, the first caret or, if there is no caret, the first dot separates the name from the version.
- The version part of the string can be divided by dots into substrings.
- Consecutive dots are not allowed.
- All ASCII characters can be used except:
  - All characters not supported by Tivoli Management Framework. For more information, refer to *Tivoli Management Framework: Reference Manual*.
  - The wildcard characters ″*″ and ″?″.
  - Punctuation marks other than the dot used to separate tokens
  - Speech marks (″ ’)
  - Spaces or carets (^), other than a caret used to separate name and version.
- Both upper and lower case characters can be used.

Notes:

1. On the UNIX and NetWare platforms, a single caret or dot symbol between the name and version is required.
2. On OS/2 platforms, the format must be *spname^^version*, *“spname^version”* or *spname.version*.
3. In the Solaris Operating Environment (*sh* shell environment), specify *“spname^version”* or *spname.version*.

Table 1 shows examples of name-version strings, how they are interpreted, and whether they are valid.

Table 1. Interpretation of name-version strings

String	Name	Version	Valid
mypkg^b1.d	mypkg	b1.d	Yes
a.pkg^1c.2d	a.pkg	1c.2d	Yes
test.1.2.3b	test	1.2.3b	Yes
appl^1.1B.0.2	appl	1.1B.0.2	Yes
example^1..2	example	1..2	No
string	string	Null	No

Defining a Version String

The version string of a software package name is a maximum of 64 single-byte characters long and can be divided into tokens, using the dot character as the divider. Each token identifies a version or sub-version of the software package. For example, the software package name and version could be constructed as follows:

```
package_name.majorversion.minorversion.patchnumber
```

The three strings following the first dot provide versioning information about the package. They represent the following sequence in the development of the software package:

1. New features are added to the product. This is indicated by incrementing the first token.
2. The software is adapted to meet changes in hardware configurations, operating system versions, data formats, and so on. This is indicated by incrementing the second token.
3. Faults in the software are corrected. This is indicated by incrementing the third token.

Software Package Version Checking

Version checking introduces safeguards that prevent an out-of-sequence installation of a product or a patch.

Defining Version Checking

Version checking uses the following two attributes:

- Versioning type

  There are two valid values: SWD, which enables version checking, and None, which disables version checking. SWD is the default value.
- Package type

  There are two valid values: REFRESH and PATCH. REFRESH is the default value.

You define the versioning type and package type attributes for a package in the package stanza of the SPD file. For discovered packages, recorded using the disconnected target command **wdsetsps**, you can define the versioning attributes in the CLI. See “wdsetsps” on page 289.

You can obtain information about the current version, versioning type, and package type of a software package using the **wsdvers** command. See “wsdvers” on page 212.

Applying Versioning Checks

If the versioning type of the package is set to SWD, checks are made for other versions of the same package. Software Distribution does a first check on the server before distribution. The version checks are made on the target catalog, before a change management operation is performed.

The information in the catalog on the target cannot be assumed to be consistent with the information in the Inventory and historical database. For example, inconsistencies can occur when a package has been removed using the disconnected target command **wdrmvsp** or when a refresh package and a patch are included as nested packages in the same distribution. You can synchronize the information on the server with the information on the target using the **wsyncsp** command. See “wsyncsp” on page 261.

Note: If you specify the **–f** argument with the **winstsp** command for a versionable package, the version checks are not made on the server before distribution. Version checks are always made on the target and if these fail, the installation is cancelled.

Installation of the package is rejected if any of the following conditions are true:

- A later version of the product is present on the target in a software package state other than RC (removed, not undoable).

  For example, the software package pkg.2.1 should be installed on endpoints ep1, ep2, and ep3. The package type is defined as REFRESH. When the inventory is checked, ep1 has pkg.2.0 installed, ep2 currently does not have a version of the package installed, and ep3 has pkg 2.2 installed. The new software package, pkg.2.1 can be installed on ep1 where there is an earlier package and on ep2 where there is no package. It cannot be installed on ep3 because ep3 has a later version of the product already installed.
- There are versions of the package present on the target that are in uncommitted software package states.
- The package type is PATCH and there is no refresh package currently installed on the endpoint.

  Note: When a patch is applied to a product that is installed in undoable mode, the patch must also be applied in undoable mode.

Installing Versionable Packages

When a refresh package is installed in undoable mode, any package that is superseded by the installation is set to a hidden state. Packages can be retrieved from the hidden state by undoing the installation.

When a refresh package is installed in non-undoable mode, the cm status in the history of the superseded package is deleted and cannot be recovered.

Note: Any components of the old version that are also present in the new version are replaced when a versionable package is installed. For example, if the old and new versions both contain the file myfile, the old version of myfile is replaced by the new one. However, if the old version includes components that are not present in the new version they remain on the target system unless you include actions in the software package to remove them.

Using Variables

In a software package definition (SPD) file, you can define variables, which associate a symbolic name with a value. Using a variable instead of its value makes a software package more generic for use on different target systems. Each variable is replaced with its value during the execution of a Software Distribution operation. For example, to install the file app_config.txt under the Windows system32 directory, you create an SPD file stanza named File System objects:

```
add_directory
  location          = c:\winnt
  name              = system32
  destination       = c:\winnt\system32
  add_file
     name           = app_config.txt
     destination    = target_app_config.txt
  end
end
```

However, if the target system has the Windows XP operating system installed on d:\windows rather than d:\winnt, the software package just created cannot be used. To solve this problem, you can use a variable for the destination path, as follows:

```
add_directory
     location       = c:\winnt
     name           = system32
```

```
     destination    = "$(system_dir)"
     add_file
        name        = app_config.txt
        destination = target_app_config.txt
     end
end
```

The *system_dir* variable is resolved by the Software Distribution engine at install time, so the software package can be used on any Windows workstation, regardless of where the Windows operating system is installed.

Variables can also be nested. For example, *$(target_dir.$(os_name))* allows you to assign different values to *target_dir*, based on the platform. In the following example of a single file system section, a variable is used, and the default value depends on the operating system platform specified:

```
default_variables
  target_dir.Windows_NT = c:\target
  target_dir.AIX        = /target
end
...

add_directory
  location             = c:\newsd40\prod\...\src\winobjects
  name                 = *.*
  destination          == $(target_dir.$(os_name))
end
```

In addition, a variable can contain references to other variables, for example, the variable *target_dir* can be referenced as:

```
target_dir = $(program_files)\myappl
```

With some exceptions, the value for any attribute in the SPD file that is of the type ″string″ can contain variables specified in the format *$(variable_name)*. Several types of variables are supported in the SPD file. The program looks for each of these variables, in turn, in the order in which they are listed, and uses the information from the first one that it finds. That is, it ignores the other variables. The following list details the supported variables:

Command line variables

Override all other specified variables. Command line variables are defined using the **–D** attribute during a change management operation, for example, install (the **winstsp** command) or remove (the **wremovsp** command). These variables are common to all targets that undergo one or more operations on a particular software package and are saved on the target systems. Note that these variables can be resolved only on the endpoint.

The following example shows the usage of the **–D** attribute to substitute or define the target directory variable *target_dir* when the software package is installed:

```
winstsp –D target_dir=c:\myappl \
@mypackage^1.2 @target1 @target2
```

User-file variables

User-file variables are stored in a text file called swdis.var in the product directory of an endpoint. This file contains lines in the format:

```
variable-name=variable-value
```

The swdis.var file is unique to each target system and can be managed with text file commands or edited with a system editor. You can customize the location of swdis.var by changing the user_file_variables path in the swdis.ini file.

For example, when installing a base product, if you specify the attribute **save_default_variables=y**, the value of the target directory variable *target_dir* is saved in the swdis.var file. When installing an upgrade of the same product, the previously–assigned value for *target_dir* is used, so the upgrade is installed in the same directory as the base product.

Built-in variables

Evaluated automatically at run time without user intervention. See Appendix A, "Built-in Variables," on page 353 for a list of the variables that are provided with Software Distribution.

LDAP Variables

On Windows platforms, you can specify Lightweight Directory Access Protocol (LDAP) variables.

These variables are resolved by obtaining the value of an entry attribute, defined on an LDAP server. The value is obtained by executing a query on the LDAP server, according to the LDAP protocol, version 3.

Note: The user name and password of the user currently logged on to the system where the operation is performed is used to access the LDAP server. You must ensure that this user has access to the LDAP server.

The syntax for specifying a LDAP variable is as follows:

```
$(LDAP:<server name>:<distinguished name>:<attribute>)
```

Registry Variables

On Windows platforms, you can specify registry variables. These are variables whose values are taken directly from the Windows registry repository.

Note: This can only be used if the registry entry value is in string format. Otherwise, the value cannot be resolved.

You use the REG keyword to specify a registry variable. For example, to specify a registry variable for the S24TIST.DLL library, use the following syntax, where the last token in the path represents the registry variable:

```
$(REG:HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\
ComputerName/ComputerName)
```

If the path to the variable contains backslashes, it must be included between single or double quotes:

```
$(REG:HKEY_LOCAL_MACHINE\SOFTWARE\CurrentVersion\
SharedDLLs\'C:\Program Files\Test\S24TIST.DLL')
```

or

```
$(REG:HKEY_LOCAL_MACHINE\SOFTWARE\CurrentVersion\
SharedDLLs\"C:\Program Files\Test\S24TIST.DLL")
```

$(installed_software) variable

This variable is used in dependency expressions to specify a software requirement that must be met before an install, commit, remove, or undo operation can be executed on a software package. See "Dependency" on page 9.

Environment variables
Inherited from the lcfd.exe process or by the command line on a disconnected target, for example, *userdomain*.

Hardware-discovered variables
Discovered by the Inventory scanner in the form:

```
Table-name.Field-name = <value>
```

For example, the following are hardware-discovered variables:

```
$(COMPUTER_SYSTEM.COMPUTER_ARCHITECTURE)==Intel
$(PROCESSOR.PROCESSOR_SPEED) = 400
```

Hardware-discovered variables are stored in the following file on the endpoint: \Tivoli\lcf\inv\scanner\sd_scan.nfo. This file location is referenced as the inventory_scan file in the swdis.ini file.

The sd_scan.nfo file is created when an inventory hardware scan is performed on an endpoint and populates the configuration repository with current information about the target hardware. If this scan has not been performed, the variables cannot be used. Refer to the *User's Guide for Inventory* for more information.

Default variables
Defined directly in the SPD file using the default_variables stanza. This type of variable, for example, can specify the target directory where the files comprising an application are to be installed. The following is an example of the default_variables stanza:

```
default_variables
     target_directory = c:\MyApps
     temp_install_directory=
       "c:\MyApps\Temporary Files"
end
```

The format of *variable-name* is the same as *command-attribute*:

```
<Variable-name> [=] "<default value>"
```

The equal sign (=) is optional and *default value* must be enclosed between quotes only if it contains blanks.

You can make default variables persistent by using the software package attribute save_default_variables. Using this attribute, all the variables defined in the default_variables stanza become permanent user-file variables and are stored in the swdis.var file on the target system. For more information, see User-file variables on page 6.

Defining Dependencies and Conditions

Dependencies and conditions allow you to control the circumstances under which a package is installed or an action is executed. The difference between uses of dependency and of conditions is clarified later in this section.

Both dependencies and conditions are Boolean expressions that define a relationship between operands. For example:

```
"$(os_name) == AIX"
```

This expression defines a relationship in which the operand `$(os_name)` is equal to the operand `AIX`.

Table 2 summarizes the operators that can be used to define these relationships.

Table 2. Operators

Operator	Type	Left Operand	Right Operand	Result
NOT	Unary	N/A	Boolean expression	Boolean
> >= < >=	Binary	Numeric expression	Numeric expression	Boolean
== !=	Binary	Any expression	Any expression	Boolean
AND OR	Binary	Boolean expression	Boolean expression	Boolean
LIKE	Binary	String	String with wildcard	Boolean
CONTAINS	Binary	String	String	Boolean

Simple operands can be either constants or variables. Variables must be specified in the format *$(variable-name)*. Table 3 details the format for valid constants.

Table 3. Constants

Constant Type	Description
Boolean	The strings **true** and **false**.
Numeric	A number in the format *xxx.yyy*, where *x* and *y* are decimal digits.
String	Any string enclosed in quotes. The quotes are optional if the string contains neither spaces, parentheses nor any special characters used to specify operators. If the strings **true** and **false** are not enclosed between quotes, they are interpreted as Boolean constants.

Notes:

1. Note that when specifying the right operand of the LIKE operator, you can use the following wildcard characters: asterisk (*) to match any string, or question mark (?) to match any character, for example, `$(os_name) LIKE 'Win*'`. If the right operand does not contain wildcards, the LIKE operator functions identically to the == operator.
2. The CONTAINS operator returns a value of **true** if the first operand contains the second one. The first operand must be a multi-value string in which the values are separated by a semicolon (;), for example:

   ```
   "value1;value2" CONTAINS value54
   ```

   returns a value of **false**.
3. Parentheses can be used to prioritize expressions.

Dependency

Dependency is an attribute that you can define in the Package stanza of the SPD file. It allows you to define hardware and software prerequisites and restrictions to be evaluated when executing the install, commit, remove, and undo commands on the package.

Dependency expressions are evaluated when a package is installed or committed, when an installation is undone, and when a package is removed. The dependency check on installation evaluates the dependency expression defined in the dependency attribute for the package being installed. On remove and undo installation, the dependency expressions for other software packages that are installed on the endpoint are evaluated to ensure that the removal of the software package will not violate the dependency conditions of any of the packages.

The checks are made first on the server and then on the endpoints. If the check fails on either the server or the endpoint, the operation is aborted.

By default, dependency checking is turned on. It can be turned off using the **–R** argument when defining the appropriate command, see “Server Commands” on page 159.

The dependency condition is also evaluated during the verify phase of the software package distribution. If it fails at this point, the package is set to an error status.

A dependency expression will include one or more of the following variables:
- Hardware-discovered variables.
- The `$(installed_software)` variable.

Table 4 shows the use of the `$(installed software)` variable in dependency expressions.

Table 4. Expressions using the $installed_software variable

Expression	Indicates
″$(installed_software)== mypkg.1.0″	Version 1.0 of mypkg is a prerequisite for operations involving the package.
″$(installed_software) != mypkg.1.0″	No operations can be carried out on the package if version 1.0 of mypkg is installed.
″$(installed_software) > mypkg.1.0″	A version of mypkg that is later than 1.0 must be installed for the operations defined in the package to be allowed.
″$(installed_software) < mypkg.2.0″	The operations defined in the package to be allowed can only be performed if either an earlier version of mypkg is installed or if there is no version of mypkg installed.
″NOT($(installed_software) LIKE ‘mypkg*.*’)″	None of the sub-versions of mypkg.1, for example, mypkg1.1, mpkg.1.2, should be installed.
″$(installed_software) CONTAINS mypkg.2.3.″	Version 2.3 of mypkg is a prerequisite for the operations involving the package.

Note: Where the expression indicates that a software package must be present, for example, `"$(installed_software)== mypkg.1.0"`, the specified package must be in the IC, ICU, or IC-D state.

Where the expression indicates that the software package must not be present, for example, `"$(installed_software) != mypkg.1.0"`, the specified package must be in any state other than IC, ICU, or IC-D.

You define hardware prerequisites and restrictions using the hardware-discovered variables. See “Using Variables” on page 5.

Hardware-discovered variables are included in a file, on the endpoint, called sd_scan.nfo which is created by Inventory. If this file is not present on an endpoint, hardware-discovered variables must not be used for distributions to that endpoint.

You can build a complex expression that includes any number of hardware-discovered variables and multiple occurrences of the `$(installed_software)` variable.

For example:

```
dependency="$(installed_software)==mypkg.2.1b" AND
$(COMPUTER_SYSTEM.COMPUTER_ARCHITECTURE)==Intel OR
"$(installed_software)<mypkg.2.1b"
```

This expression indicates that the software package `mypkg` is a prerequisite for installation of the package. Additionally, if the version of `mypkg` that is installed is `2.1b`, the computer architecture must be Intel.

Note: For hardware dependencies, only variables from tables that contain the column HARDWARE_SYSTEM_ID can be evaluated on both the server and the endpoint. If you use a variable from a table that does not include this column, a warning is reported in the server log, and the dependency condition is evaluated only using the information included in the sd_scan.nfo file on the endpoint.

Conditions

Any software package action can contain conditions, which define circumstances under which the action should be executed. For example, to execute a package on an AIX system only, specify the following attribute in the package stanza:

```
condition = "$(os_name) == AIX"
```

The package is executed only if the actual value of the *$(os_name)* variable matches the string ″AIX″.

Any conditions set at the package level override conditions that are set within the package and must be met for the package to install. If a package-level condition is not met, a message indicates that the condition is false and the package will not install. However, if a package-level condition is met, but a lower-level condition is not met, the package will install successfully except for the section with the false condition, which will not install.

You can specify only one condition in the package-level stanza and only one condition within each stanza of the package. See “The generic_container Stanza” on page 18 for a technique to specify multiple conditions within a software package. Valid conditions are Boolean expressions that use the rules detailed in Table 2..

Following is an example of a complex condition that uses the expression AND:

```
$(os_name) == Windows_NT AND $(os_release) == '5.0'
```

Note: In a condition, if an operand is a string represented by numbers, it is automatically converted into a numeric value. If you compare it with a string, you receive an error reporting that the condition cannot be evaluated.

To solve this problem add an alphabetic character before the numeric value to be sure it is always considered a string.

For example, if you have the 0000040C string and you compare it with the variable langue_winnt = $(REG:HKEY_USERS\.DEFAULT\...\Locale), define the variable as follows:

```
langue_winnt = "X$(REG:HKEY_USERS\.DEFAULT\...\Locale"
```

and the condition as follows:

```
$(langue_winnt) =='X0000040C'
```

Structure of the SPD File

The first line of the SPD file is the *signature*, which uniquely identifies the object. This signature is used by Software Distribution for version checking and to determine which kind of importer should be used to read the file. The following is an example of the signature format:

```
'TIVOLI Software Package v4.3.1 – SPDF'
```

If the signature is not in this format or if the SPD file does not contain a signature, an error will result when the SPD file is imported.

The following lines are ignored in SPD files:
- Blank lines
- Comment lines, which are lines preceded by the number sign (#)

Note: Any comment lines that were included in the original SPD file are not preserved after running the **wexpspo** (export) or **wimpspo** (import) commands. See Chapter 3, “Using Commands,” on page 157 for more information about these commands.

Each software package definition file has a root stanza named package; all other stanzas are nested in this root stanza.

Each stanza in an SPD file describes an action to be performed. Actions, which can be issued from any point in the SPD file, are performed in the order in which they are found. There are three types of actions supported:
- System actions and checks, such as restart and check disk. For more information, see System actions.
- Program actions, such as execute program, install CID product, and install InstallShield product.
- Object-related actions, such as add object and remove object.

For description purposes, the stanzas in the SPD file are divided into the following categories:
- The package stanza and other stanzas that define general package information. See “General Stanzas” on page 13.
- Stanzas defining actions for adding and removing software package objects. See “Object-related Actions” on page 26.
- Stanzas defining program actions, such as execute program, install CID product, and install InstallShield product. See “Program Actions” on page 94.
- System actions and checks, such as restart and check disk. For more information, see System Actions.

The sections where these stanzas are described show the structure in which you should define each stanza and provide examples.

The following example shows the format of a generic action stanza:

```
<action> :=
     <action-name>
          # Sequence of <action-attribute>
          # Sequence of <action>
     end
```

The following example shows the format of *action-attribute*.

```
<attribute-name> = "<attribute-value>"
```

Following are the format rules for *action-attribute*:
- The equals sign is optional.
- The *attribute-value* must be enclosed in double quotes if it contains spaces.
- The *attribute-value* can span multiple lines only if enclosed in quotes. Its end is determined by the placement of the quotation mark.

All objects in stanzas must be enclosed in *containers*. For example, the stanza of the add object and remove object container has the following format:

```
<container>:=
     [add | remove] <container name>
          # <container-attribute>
          # Sequence of <object>
     end
```

The *container-attribute* value includes the following:
- Attributes inherited from the action class.
- Attributes inherited from the container class.
- Default attributes for all the objects in the container. These attributes depend on the specific container while the previous ones are common to all the containers.

The object stanza is as follows:

```
<object> :=
     <object name>
          # Sequence of <object-attribute>
     end
```

General Stanzas

This section describes the following stanzas:

package
: This defines the general information relating to the package, the name and version of the package, the name of the source host for the distribution, and information about logging.

nested_software_package
: This defines a package that is nested within the main package.

log_object_list
: This allows you to create an SPD log on the target system(s).

generic_container
: This groups commands that must satisfy the same condition.

The Package Stanza

This is the top-level stanza, in which all other stanzas that describe components of a software package are nested.

The format of the package stanza is shown in the following example.

```
package
     dependency              = <string constrain>
     # Inherited from action
     condition               = <string constrain>

     # Inherited from container
     stop_on_failure         = <Boolean>

     copyright               = <string>
     description             = <string>
     name                    = <string>
     title                   = <string>
     version                 = <string expression>
     versioning_type         = <string>
     package_type            = <string>
     sharing_control         = <string>
     lcf_before_program_path = <string>
     lcf_before_program_arguments = <string>
     lcf_before_program_timeout = <integer>
     lcf_after_program_path = <string>
     lcf_after_program_arguments = <string>
     lcf_after_program_timeout = <integer>
     undoable                = <string expression>
     committable             = <string expression>
     history_reset           = <Boolean>
     after_as_uid            = <integer>
     after_input_path        = <pathname>
     after_prog_env          = <string>
     after_program_path      = <pathname>
     before_as_uid           = <integer>
     before_input_path       = <pathname>
     before_prog_env         = <string>
     before_program_path     = <pathname>
     default_operation       = <string expression>
     lenient_distribution    = <Boolean>
     log_gid                 = <group ID>
     log_mode                = <octal>
     log_path                = <string>
     log_host_name           = <string>
     log_user_id             = <integer>
     mail_id                 = <string>
     move_removing_host      = <Boolean>
     no_check_source_host    = <Boolean>
     no_chk_on_rm            = <Boolean>
     operation_mode          = <string expression>
     post_notice             = <Boolean>
     save_default_variables = <Boolean>
     server_mode             = <Boolean
     skip_non_zero           = <Boolean>
     source_host_name        = <string>
     spb_path                = <string>
     stage_area              = <string>
     web_view_mode           = <string expression>
     creation_time           = <string>
     last_modification_time = <string>

     nested_software_package
        # Sequence of <package>
     end
```

```
     default_variables
        # Sequence of <name-value>
     end

        # Sequence of <command>
end
```

For information about the attributes that can be defined in this stanza, see Table 5 on page 18..

Setting Up Before and After Programs on the Source Host

You can set up programs to run on the source host before and after a software package is built. The following attributes of the package stanza control such programs:

- before_prog_path
- before_input_path
- before_as_uid
- before_prog_env
- after_prog_path
- after_input_path
- after_as_uid
- after prog_env

Refer to the description of the **wsetspgs** command (“wsetspgs” on page 219) for more information about these attributes.

Setting Up Before and After Programs on the Endpoint

You can set up programs to run on the endpoint before and after the change management operation is performed on the endpoint. For example you can use these programs to modify the variables in the package when the package is installed. The following attributes of the package stanza control such programs:

- lcf_before_program_path
- lcf_before_program_arguments
- lcf_before_program_timeout
- lcf_after_program_path
- lcf_after_program_arguments
- lcf_after_program_timeout

Refer to the description of the “Attributes in the General Stanzas” on page 18 for more information about these attributes.

You can define these attributes using the **wsetspgs** command, and view the values defined for the attributes using the **wgetspgs** command. For more information on these commands, see “wgetspgs” on page 180 and “wgetspgs” on page 180.

You can also use the **wdswdvar** command in the program to edit the variables in the software package. For more information on this command, see “wdswdvar” on page 291.

Notes:

1. When defining a before or after program on OS/2 machines, use .cmd or .exe files, because .bat files are not supported.
2. On Netware endpoints you cannot use disconnected target commands when using before and after programs.
3. The before program is not run on the endpoint, if you are installing the software package from a disconnected Command Line Interface.

Some arguments are passed to the program by default. The following is a list of the default arguments passed to the program:

operation_type
Supported values are either install or remove. The remove operation is performed only on endpoints where the software package was not previously installed.

endpoint_label
The label for the endpoint.

machine_id
The machine identifier.

endpoint_guid
The hardware system identifier of the machine stored in the Inventory database.

region_number
The region number.

distribution_id
The distribution identifier.

operation_result
For the after program only. The result of the change management operation. If 0 is returned, the operation completed successfully; if 1 is returned, the operation failed.

Nesting Software Packages

You can manage several software packages with a single action. The list of packages to be managed (called nested software packages) is specified inside a package called the primary software package. The order of processing is determined by the position of the nested software packages specified inside the primary software package. If the primary package is not present in the list, it is installed before all nested software packages. Nested software packages are specified as a stanza inside the package stanza in the following way:

```
nested_software_package
  # Sequence of <package>
end
```

where the sequence of nested packages is specified as follows:

```
package
    name = <string>
    version = <version>
end
```

The following list details the behavior of Software Distribution when change management operations are performed on software packages that contain nested software packages.

Install The primary software package and its nested software packages are installed in the order specified at preparation time.

Remove
Performed on the primary software package and its nested software packages, in the reverse of the order specified at creation time.

Undo Performed on the primary software package and its nested software packages, in the reverse of the order specified at creation time.

Accept
Performed on the primary software package and its nested software packages, in the order specified at creation time.

Commit
Performed on the primary software package and its nested software packages, in the order specified at creation time.

Verify Performed on the primary software package and its nested software packages, in the order specified at creation time.

Load Performed on the primary software package and its nested software packages, in the order specified at preparation time.

Unload
Performed on the primary software package and its nested software packages, in the reverse of the order specified at creation time.

The following limitations exist when nesting software packages:
- All software packages (both primary and nested) must have the same source host.
- Nested software packages are independent from the primary software package. They must be built and imported in the object database independently from the primary software package.
- Nested software packages can reside in different policy regions, but you must have the necessary authorizations to perform operations on the primary software package and on all nested software packages.
- Repair, install repair, install source and preview operations run only on the primary software package.
- Operations on nested software packages are effective only for server commands, not in a disconnected target environment.
- When setting up before and after programs on the endpoint for nested software packages, only the before and after programs specified for the primary package are considered, so that only one program is run at the beginning of the change management operation, and only after all of the nested software packages are executed is the after program run.

For information about the attributes that can be defined in this stanza, see Table 5 on page 18.

For information about the Software Distribution commands that retrieve information on existing nested software packages and that set up new ones, see “wgetsnsp” on page 176 and “wsetsnsp” on page 213.

The log_object_list Stanza

This stanza enables you to create an SPD log file on the target system. The log file will contain the actions and results of the change management operations on the package.

```
log_object_list
          location        = <pathname>
          unix_group_id   = <integer>
          unix_user_id    = <integer>
          unix_attributes = <string expression>
     end
```

For information about the attributes that can be defined in this stanza, see Table 5 on page 18.

The generic_container Stanza

You can use the generic_container stanza to group commands together that must satisfy the same condition to be affected by change management operations. You can specify a single condition within the package stanza and a single condition in each of the stanzas it contains. If you want to define multiple conditions, you must use the generic_container stanza.

This stanza can be specified in the package stanza, or inside any other container.

```
generic_container
  ##
  # All the attributes are inherited from composite_command
  ##
  caption            = <string>
  condition          = <string>
  stop_on_failure    = <Boolean>
  ## Sequence of commands
end
```

See “Defining Dependencies and Conditions ” on page 8 for information about defining the conditions.

This stanza can also be nested in other generic_container stanzas, for example:

```
generic_container
  condition = "$(os_name) LIKE 'Win*'"
  ## Command common to all supported Windows operating systems
  ...
  generic_container
    condition = "$(os_name) == Windows_NT"
    ## Command specific to all supported Windows operating systems
    end
  end
```

For information about the attributes that can be defined in this stanza, see Table 5.

Attributes in the General Stanzas

Table 5 shows a list of all the attributes that can be defined in the package, nested package, log_object_list, and generic_container stanzas.

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
after_as_uid	Specifies the UNIX user ID under which to run after_program_path.			
	Integer	No	None (**0**)	package
after_input_path	Used in conjunction with after_program_path, a file from which the program requires input during its execution.			
	String	No	None	package
after_prog_env	Sets the list of environment variables for the after program.			
	String of *name=value* pairs	No	None	package
after_program_path	Specifies the path of the program to be run on the source host after the build is completed.			
	String	No	None	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
before_as_uid	Specifies the UNIX user ID under which to run before_program_path.			
	Integer	No	None (**0**)	package
before_input_path	Used in conjunction with before_program_path, a file from which the program requires input during its execution.			
	String	No	None	package
before_prog_env	Sets the list of environment variables for the before program.			
	String of *name=value* pairs	No	None	package
before_program_path	Specifies the path of the program to be run on the source host before the build is begun.			
	String	No	None	package
caption	A unique identifier to define any component in the package.			
	String	Yes	None	generic_container
committable	If **y**, you must install the package in transactional mode. If this attribute is specified in a nested software package, it will override the value of the attribute specified in the primary software package.			
	y: yes **n**: no **o**: optional	No	**o**	package
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	generic_container package
copyright	Manufacturer of the package.			
	String	No	None	package
creation_time	The date and time when the package was created.			
	String	No	None	package
default_operation	Specifies the default change management operation to be performed on the object. If this attribute is specified inside a nested software package, it will be overridden by the value specified in the primary software package (or ignored if not specified in the primary software package).			
	install, **remove**, **undo**, **accept**, **commit**	No	None	package
dependency	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
description	A more complete description of the package, as in a readme file. If you plan to make this package available for download from the Software Distribution Web Interface, enter the text string with HTML tagging. This text displays in the Web Interface when you click the software package name.			
	String	No	None	package
history_reset	If set to **y**, indicates that the software history of all software packages that are already installed on the target is to be erased, if the package is installed successfully.			
	y: yes **n**: no	No	**n**	package
last_modification_time	The date and time when the package was last changed.			
	String	No	None	package
lcf_before_program_ path	The pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run. on the target system on which it is to run.			
	String	No	None	package
lcf_before_program_ arguments	Optional arguments passed to the program in addition to the default ones. Separate arguments with a blank space. You can also use variables to specify arguments.			
	String	No	None	package
lcf_before_program_ timeout	The time, expressed in seconds or the value -1, to wait for the completion of the before program. See “Format of the execute_user_program Stanza” on page 118 for detailed information.			
	Integer	No	None	package
lcf_after_program_ path	The pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run.			
	String	No	None	package
lcf_after_program_ arguments	Optional arguments passed to the program in addition to the default ones. Separate arguments with a blank space. You can also use variables to specify arguments.			
	String	No	None	package
lcf_after_program_ timeout	The time, expressed in seconds or the value -1,in seconds or the value -1, to wait for the completion of the after program. See “Format of the execute_user_program Stanza” on page 118 for detailed information.			
	Integer	No	None	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
lenient_distribution	If set to **n**, allows distributions and removals only to those managed nodes, endpoints, or profile managers that are current subscribers of the profile manager to which the software package belongs. If this attribute is specified inside a nested software package, it will be overridden by the value specified in the primary software package (or ignored if not specified in the primary software package).			
	y: yes **n**: no	No	**n**	package
location	The fully qualified path of the directory on the target where the log file is written. By default the log file is overwritten for each distribution of the software package. If the directory does not exist, it is created. The name of the log file is in the format package-name.version.log.			
	String	Yes	None	log_object_list
log_gid	Specifies the UNIX group ID for the log file.			
	Group ID (long)	No	**-1**	package
log_host_name	Sets the log_host attribute, which specifies the label of the managed node where the log file is generated. Specify the host name.			
	String	No	None	package
log_mode	Specifies the UNIX file mode of the generated log file.			
	Octal	No	None	package
log_path	Specifies the absolute path and file name of the generated log file.			
	String	No	None	package
log_user_id	Specifies the UNIX user ID of the log file.			
	Integer	No	None (**0**)	package
mail_id	Specifies the user e-mail address to notify when an operation is performed. To specify multiple e-mail IDs, separate each ID with a comma (requires the sendmail program).			
	String	No	None	package
move_removing_ host	Specifies whether to move the software package to the lost-n-found collection if the log host or source host of the software package has been removed. If this attribute is specified in a nested software package, the specified value is ignored.			
	y: yes **n**: no	No	**y**	package
name	A unique identifier. Any package must be uniquely identified by its name and version.			
	String	Yes	None	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
nested_software_ package	A sequence of stanzas listing the name and version of each software package that is nested in the primary software package. See “Nesting Software Packages” on page 16 for more information.			
	Sequence of stanzas: package name=*string* version=*string* end	No	None	package
no_check_source_ host	If set to **n**, as the object is imported as a software package, Software Distribution checks if the managed node and the file contents exist.			
	y: yes **n**: no	No	**y**	package
no_chk_on_rm	For nested software packages, the primary software package contains a list of all nested software packages. If this attribute is set to **n** when a software package is removed, all software packages in the Tivoli Name Registry are checked to eliminate any reference to the deleted software package.			
	y: yes **n**: no	No	**y**	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
operation_mode	Specifies the default mode for change management operations. Multiple values can be assigned to this attribute, for example: `operation_mode = "transactional|undoable|autocommit"` If this attribute is specified inside a nested software package, it will be overridden by the value specified in the primary software package (or ignored if not specified in the primary software package). This attribute is only used for change management operations performed using the GUI. It is ignored for change management operations performed via the command line. **Note:** The prefer_not_transactional value is not supported for OS/400 endpoints.			
	not_ transactional, **prefer_not_ transactional**, **auto_commit**, **transactional**, **undoable**, **prefer_undo**, **auto_accept**, **undoable_in transactional, during_reboot**, **auto_reboot**, **force**, **prefer_not_ reboot**	No	**not_ transac tional**	package
package_type	Specifies the type of package to be distributed. This is only relevant if version checking is used for the package. See “Software Package Version Checking” on page 4.			
	PATCH, **REFRESH**	No	**REFRESH**	package
post_notice	Specifies whether to send a notice. When specifying post_notice=y, also run the following CLI command: `wsetspop -P true @spobj_name` The notice is posted to the Software Distribution 4 notice group. If this attribute is specified inside a nested software package, it will be overridden by the value specified in the primary software package (or ignored if not specified in the primary software package).			
	y: yes **n**: no	No	**n**	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
save_default_ variables	Specifies whether to make the default variables persistent for future use. If **y**, the default variables are saved as user-file variables and added to the swdis.var file during the software package installation, so that the same variables can be used for installing upgrades or additional software package features.			
	y: yes **n**: no	No	**n**	package
server_mode	Specifies the default server mode. If this attribute is specified inside a nested software package, it will be overridden by the value specified in the primary package (or ignored if not specified in the primary software package). This attribute is only used for change management operations performed using the GUI. It is ignored for change management operations performed via the command line.			
	all, **src**, **repair**, **check**, **force**, **ignore**, **preview**	No	None	package
sharing_control	Specifies a check on the presence of files and registry values. This prevents their deletion when the package is removed, if these objects were already present on the target prior to the distribution of the software package. The values have the following meanings: **none** No check is performed on the files and registry values already present on the target and distributed with the software package. When the software package is removed, all distributed objects are removed. **only_shared** A check is performed only on the files and registry values that have the **is_shared** attribute set to **y**. When the software package is removed, these files and registry values are not removed if they were already present on the target prior to distributing the software package. **auto** A check is performed on all the files and registry values distributed with the software package. When the software package is removed, all the files and registry values distributed with the software package are not removed if they were already present on the target prior to distributing the software package. If either **auto** or **only_shared** is specified, the version of the objects distributed with the software package is left on the target when a remove operation is performed.			
	none, only_shared, auto	No	**none**	package

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
skip_non_zero	If set to **y**, indicates that distribution is skipped if before_program_path exits with a non-zero exit code.			
	y: yes **n**: no	No	**n**	package
source_host_name	Specifies the source host where the files in the software package block are obtained and where the software package block is stored. The source host can be any of the available managed nodes where the Software Distribution Source Host component is installed.			
	String	No	None	package
spb_path	Specifies the object path or registered name of the software package block whose attributes are set by this operation.			
	String	No	None	package
stage_area	Specifies the full path to a directory on the source host where the software package block of the software package object will reside.			
	String	No	None	package
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met. If the stop_on_failure attribute is set to y in a stanza contained in a package, and one of the actions in the stanza fails, the remaining actions in the stanza are not performed, but the execution of the remaining stanzas continues, provided that the stop_on_failure attribute in the package stanza is set to n. In this case, the final state of the package is IC. If the stop_on_failure attribute is set to y in the package stanza and an error occurs, the remaining actions in the package are not performed, and the execution of the package is not completed. In this case, the final state of the package is IC - -E.			
	y: yes **n**: no	No	**y**	generic_container package
title	Specifies a descriptive string used for display. The Software Distribution Web Interface displays this string in the Description column to identify the software package.			
	String	No	None	package
undoable	If set to **y**, you must install the package in undoable mode. If this attribute is specified in a nested software package, it will override the value of the attribute specified in the primary software package.			
	y: yes **n**: no **o**: optional	No	**o**	package
unix_attributes	A string formatted as: owner, group, others, where each set of attributes is a combination of the characters **r** and **w**.			
	r: read **w**:write	No	None	log_object_list

Table 5. SPD file attributes in the package, log_object_list and generic_container stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
unix_group_id	The UNIX group ID.			
	Integer	No	None	log_object_list
unix_user_id	The UNIX user ID.			
	Integer	No	None	log_object_list
version	One to three numeric strings separated by a period.			
	String expression	Yes	None	package
versioning_type	Indicates whether the package is versionable. Version checking is enabled if this attribute is set to SWD. See "Software Package Version Checking" on page 4.			
	NONE, **SWD**	No	**SWD**	package
web_view_mode	Specifies access permission to the package when using the Web Interface feature.			
	hidden, **subscriber**, **public**	No	**hidden**	package

Object-related Actions

The add object and remove object actions drive the engine to add the specified object to the system or to remove it from the system. The types of specialized, built-in objects that are supported include the following:

- "File System Objects" on page 27
  - Files
  - Directories
  - File system links
- "Windows Profile Objects" on page 43
  - Sections
  - Items
- "Windows Shell Objects" on page 46
  - Folders
  - Links
- "Windows Registry Objects" on page 51
  - Keys
  - Values
- "Windows Services Objects" on page 56
  - Service
- "OS/2 Profile Objects" on page 58
  - Profiles
  - Items
- "OS/2 Desktop Objects" on page 62
  - Generic objects
  - Folders
  - Programs
  - Shadows
- "Text File Objects" on page 69

  - Lines
  - Command lines
  - Tokens
- "OS/400 Native Objects" on page 76
  - Libraries
  - Objects
  - Licensed programs
  - System values
- "The contained_signature Stanza" on page 81.

You can also perform add actions on target devices. Actions include adding the following objects:
- Directory
- Execute program
- File
- Device Customization

See "Device-related Actions" on page 83 for details about creating an SPD file for devices.

Note: Throughout this book, variables are shown in italic font, except inside code examples, where they are shown within angle brackets. For example, the *object-attribute* variable is represented as follows inside a code example:

```
# Sequence of <object-attribute>
```

Management of Shared Objects

If any object is shared between multiple software packages, use the is_shared attribute, in the appropriate stanza, to prevent the inadvertent deletion of an object when one of the software packages in which it is contained is removed.

If you install several packages with shared objects and one of these objects already exists on the target (but has not been identified as shared), when you remove the same packages from the target, this object is removed as well. To restore the original object to the target when all packages specifying the object as a shared resource have been removed, perform an undo operation on the last package removed.

In addition to the is_shared attribute, file system objects have a related attribute, shared_counter. Use the shared_counter attribute in conjunction with is_shared to indicate how many times the reference counter associated with the shared file should be incremented.

File System Objects

File system objects include files and directories for OS/2, UNIX, NetWare, and all Windows operating systems, as well as file system links for UNIX operating systems. In this section, the format of each file system objects stanza is provided.

Replacing Target Objects

File system objects can be added, removed, or replaced using definitions provided in these stanzas. When removing a target object or replacing a target object with a source object, the operation performed depends on the settings of the following attributes:
- replace_if_existing
- replace_if_newer (applicable to files only)
- remove_if_modified

- verify_crc

On Windows platforms, the Windows file version, if specified, is always compared first. A check of the file version is performed only if the source host is a Windows machine, or if the software package containing the object in question was built on a Windows machine and imported in software package block (built) format.

If replace_if_newer=y, even if the file version of the target object is newer than the source object, the target object is replaced. If the file version is not set (for example, on non-Windows platforms), the modification time is compared. Even if the modification time of the target object is more recent than the source object, the target object is replaced.

Figure 2 outlines the logic for replacing target objects with source objects.

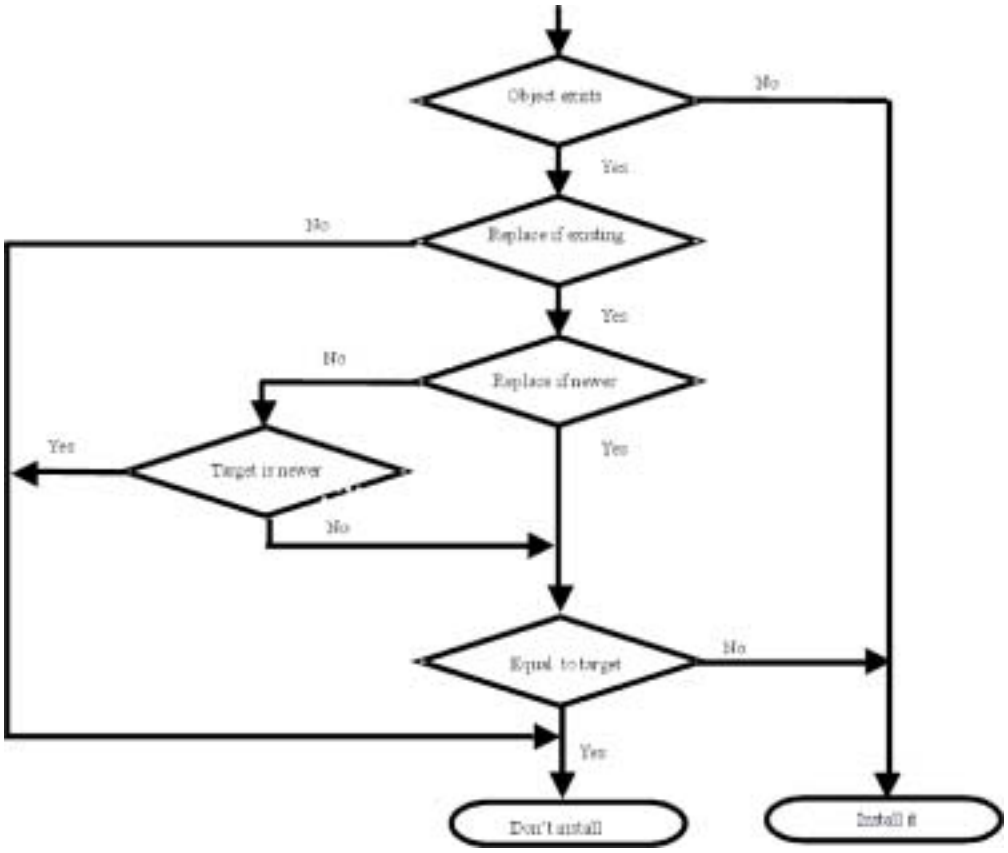


Figure 2. Replacing target objects

Figure 3 on page 29 details the logic for comparing source and target files. This comparison may be useful to verify the integrity of a file if a network connection temporarily fails while the file is being installed.

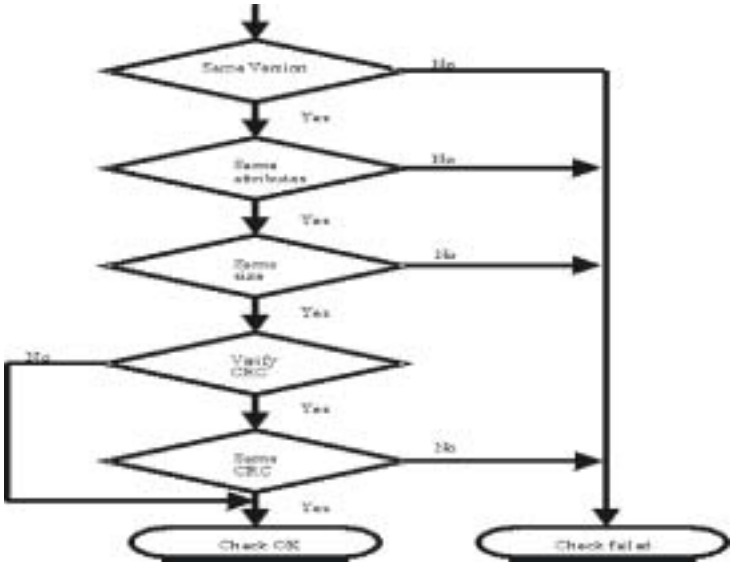


Figure 3. Checking the integrity of files

Figure 4 outlines the logic for removing target objects.

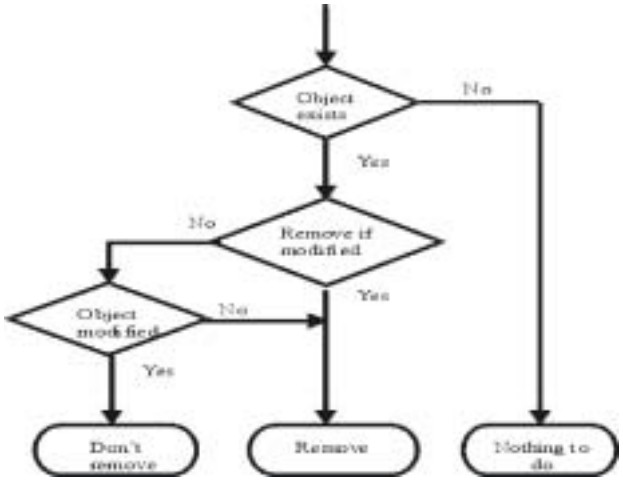


Figure 4. Removing target objects

Directories and Files

The definitions in this section shows the attributes that can be defined in the **add_directory** stanza. For information about these attributes, see Table 11 on page 36.

```
add_directory
     #Specific attributes
     add                  = <Boolean>
     location             = <pathname>
     name                 = <pathname>
     destination          = <pathname>
     descend_dirs         = <Boolean>
     translate            = <Boolean>

     # Inherited from Action
     condition            = <string constrain>

     # Inherited from Container
```

```
     stop_on_failure      = <Boolean>

     # Inherited from AddObject
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
     is_shared            = <Boolean>
     shared_counter       = <integer>

     # Defaults for all the contained objects
     substitute_variables = <Boolean>
     remove_extraneous    = <Boolean>
     compute_crc          = <Boolean>
     verify_crc           = <Boolean>
     fat_attributes       = <string expression>
     netware_attributes   = <string expression>
     ntfs_attributes      = <string expression>
     unix_attributes      = <string expression>
     unix_owner           = <string>
     unix_group           = <string>
     unix_group_id        = <integer>
     unix_user_id         = <integer>
     create_dirs          = <Boolean>
     remove_empty_dirs    = <Boolean>
     remote               = <Boolean>
     compression_method   = <deflated, stored>
     temporary            = <Boolean>
     rename_if_locked     = <Boolean>
     delta_compressible   = <string>
     is_signature         = <string>
     inventory_description= <string>
     inventory_version    = <string>

     # Sequence of <directory, file, link>
end
```

The following definition shows the attributes that can be defined in a **remove_directory** stanza. For information about these attributes, see Table 11 on page 36.

```
remove_directory
      # Specific attributes
      remove              = <Boolean>
      descend_dirs        = <Boolean>

     # Inherited from action
     condition            = <string constrain>

     # Inherited from container
      stop_on_failure     = <Boolean>
      destination         = <pathname>

     # Defaults for all the contained objects
      remove_empty_dirs   = <Boolean>

     # Sequence of <directory, file, link stanzas>
end
```

The following definition shows the attributes that can be defined in an **add_file** stanza. For information about these attributes, see Table 11 on page 36.

```
add_file
     # Specific attributes
     name                 = <pathname>
     destination          = <pathname>
     translate            = <Boolean>
```

```
     # Inherited from action
     condition            = <string constrain>

     # Inherited from add object
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
     is_shared            = <Boolean>
     shared_counter       = <integer>


     # Defaults for all the contained objects
     substitute_variables = <Boolean>
     remove_extraneous    = <Boolean>
     compute_crc          = <Boolean>
     verify_crc           = <Boolean>
     fat_attributes       = <string expression>
     netware_attributes   = <string expression>
     ntfs_attributes      = <string expression>
     unix_attributes      = <string expression>
     unix_owner           = <string>
     unix_group           = <string>
     unix_group_id        = <integer>
     unix_user_id         = <integer>
     create_dirs          = <Boolean>
     remove_empty_dirs    = <Boolean>
     remote               = <Boolean>
     compression_method   = <deflated, stored>
     is_shared            = <Boolean>
     temporary            = <Boolean>
     rename_if_locked     = <Boolean>
     delta_compressible   = <string constrain>
     is_signature         = <string>
     inventory_description= <string>
     inventory_version    = <string>

end
```

The following definition shows the attributes that can be defined in a **remove_file** stanza. For information about these attributes, see Table 11 on page 36.

```
remove_file
     # Inherited from action
     condition          = <string constrain>
     destination        = <pathname>
     remove_empty_dirs  = <Boolean>

end
```

Managing Default Attributes for Files and Directories: Default attributes for files and directories are set at build time only if you did not explicitly specify them in a software package (using either the graphical user interface (GUI) or an SPD file). For example, if the source host is running Windows XP and you specify ntfs_attributes for a particular file, no default attributes are set for other platforms (fat_attributes, unix_attributes, or netware_attributes).

File system attributes related to the build platform are loaded directly from the source host, while those related to the other supported platforms use defaults according to the following rules. In the following tables, “n/a” signifies that the default attribute of the build platform applies.

If the build platform is file allocation table (FAT), the following default attributes apply:

Table 6. Default attributes for the FAT build platform

Build Platform FAT	NTFS	UNIX	NetWare
R	**R**	**r, r, r**	**R**
A	**A**	n/a	**A**
H	**H**	n/a	**H**
S	**S**	n/a	**S**

If the build platform is NT file system (NTFS), the following default attributes apply:

Table 7. Default attributes for the NTFS build platform

Build Platform NTFS	FAT	UNIX	NetWare
R	**R**	**r, r, r**	**R**
A	**A**	n/a	**A**
H	**H**	n/a	**H**
S	**S**	n/a	**S**
C	n/a	n/a	n/a

If the build platform is UNIX, the following default attributes apply:

Table 8. Default attributes for the UNIX build platform

Build Platform UNIX	FAT	NTFS	NetWare
r-wx	**R**	**R**	**R**
r-x.	n/a	n/a	n/a

If the build platform is NetWare, the following default attributes apply:

Table 9. Default attributes for the NetWare build platform

Build Platform NetWare	FAT	NTFS	UNIX
X	n/a	n/a	n/a
B (shareable)	n/a	n/a	n/a
T (transaction)	n/a	n/a	n/a
P (purge)	n/a	n/a	n/a
N (rename inhibit)	n/a	n/a	**r, r, r,**
D (delete inhibit)	**R**	**R**	**r, r, r**
C (copy inhibit)	n/a	n/a	n/a
R	**R**	**R**	**r, r, r**
A	**A**	**A**	n/a
H	**H**	**H**	n/a
S	**S**	**S**	n/a

For example, if fat_attributes is set to **R**, the following defaults apply for other platforms:
- ntfs_attributes = **R**
- netware_attributes = **R**
- unix_attributes = **r, r, r**

If netware_attributes is set to **A D X**, the following defaults apply:
- ntfs_attributes = **A R**
- fat_attributes = **A R**
- unix_attributes = **r - x, r - x, r - x**

If unix_attributes is set to **r - x...**, the following defaults apply:
- ntfs_attributes = **R**
- fat_attributes = **R**
- netware_attributes = **R X**

Managing Locked Files: By default, files that are in use by target applications at the time of installation are not replaced on the target. Instead, such locked files are replaced during the next reboot of the target. This is achieved by installing a software package in transactional mode, then performing a commit operation with reboot.

To avoid the necessity to reboot, you can use the rename_if_locked attribute for install operations. If rename_if_locked is set to **y**, during the installation, Software Distribution attempts to overwrite or rename the locked file so that the file can be replaced and the installation can continue. The new copy of the file takes effect the next time the application is started. If it cannot either overwrite or rename the file, the installation fails. Therefore, though setting this attribute can avoid a reboot it also carries the risk of causing the installation to fail.

During a remove operation it is not possible to avoid the reboot by using the rename_if_locked attribute. If a file to be removed is in use and rename_if_locked is set to **y**, the removal of the file is delayed until the next reboot of the target system.

You can use the rename_if_locked attribute on all supported Windows platforms (for all file types) and on OS/2 (for .exe and .dll files only).

If you are not sure if the target files are in use, perform the distribution specifying the preferably-not-transactional option (**–to**), as well as the commit with the reboot-only-if-necessary option (**–co**), which performs a transactional install and an automatic reboot if the files are in use (locked). If the files are not locked, an installation is performed to the active area.

Note that if target files are read- and write-locked when you perform a distribution in undoable mode, the backup operation required to potentially undo the distribution will fail. If the target files are only write-locked, the backup operation will succeed. Using Undo-transactional mode avoids this problem.

For more information about the modes and commands described in this section, see Chapter 2, “Performing Change Management Operations,” on page 143 and Chapter 3, “Using Commands,” on page 157.

Specifying File and Directory Names with Wildcard Characters: When specifying the path of a source or target file, file and directory names may contain wildcard (pattern-matching) characters. The supported wildcard characters include:

Asterisk (*)
Matches any string, including the null string.

Question mark (?)
Matches any single character.

The following example shows how to use wildcards to add directories and files.

```
add_directory
  descend_dirs       = y
  location           = c:\MyApp
  name               = dir*
  destination        = c:\target
end
```

In this example, directories and files that match the pattern in the name are added to the package. The files and directories are installed with their original name into the target directory at installation time. If the same directories are specified with the **descend_dirs = n** attribute, only files and directories at the top level of the specified directory structure are added.

On the other hand, if you specify the following files:

```
add_directory
  descend_dirs       = n
  location           = c:\MyApp
  name               = *.txt
  destination        = c:\target\*.new
```

only the files that match the pattern are added to the package. The files are installed with their original name into the target directory at installation time. However, if you had specified **descend_dirs=y**, and the following files were in the source directory structure:

```
c:\MyApp\MyFile.txt
c:\MyApp\bin1\File.txt\
c:\MyApp\bin2\bin3\web.txt
```

the following files would be installed on the target system:

```
c:\target\MyFile.new
c:\target\bin1\File.new
c:\target\bin2\bin3\web.new
```

The following example shows how to use wildcards to remove directories and files:

```
remove directory
               destination =" c:\target\new*"
               descend_dirs = y
               name = "new*"
               remove= y
               remove_empty_dirs = y
               rename_if_locked = n
               stop_on_failure = y
   end
```

In this example, directories and files that match the pattern in the name are removed from the target. For example, if the following directory and files are present on the target they are removed:

```
c:\target\newfile.txt
c:\target\newdoc.doc
c:\target\newweb.txt
c:\target\newdir\*.*
```

If the **descend_dirs** = **n** attribute is specified, only files at the top level of the specified directory structure are removed. For the example above, the following files are removed from the target:

```
c:\target\newfile.txt
c:\target\newdoc.doc
c:\target\newweb.txt
```

Note: In the file or link stanzas, the name attribute cannot include wildcards.

Links

The following definition shows the attributes that can be defined in an **add_link** stanza. For information about these attributes, see Table 11 on page 36.

```
add_link

  # Inherited from action
  condition               = <string constrain>

  # Add object default attributes
  replace_if_existing     = <Boolean>
  replace_if_newer        = <Boolean>
  remove_if_modified      = <Boolean>

      # Specific attributes
  source                  = <string>
  destination             = <string>
  follow_links            = <Boolean>
  hard_link               = <Boolean>

end
```

The following definition shows the attributes that can be defined in a **remove_link** stanza. For information about these attributes, see Table 11 on page 36.

```
remove_link

  # Inherited from action
    condition               = <string constrain>

  # Folder specific attributes
    destination             = <string>
    remove_empty_dirs       = <Boolean>

end
```

Table 10 details the results of the possible interactions of the follow_links and hard_link values.

Table 10. Combining follow_links and hard_link values

Value of follow_links	Value of hard_link	Result
n	**n**	Creates a symbolic link file (called *destination*) to *source*. Note that *source* may or may not exist.
n	**y**	Creates a hard link to *source*. Note that *source* must exist, otherwise an error message is displayed and the install operation fails.
y	**n**	Creates a new *destination* link file by copying the file to which the *source_file* link is pointing on the source host. The *source* and the file to which it is pointing must exist.
y	**y**	Invalid combination.

Attributes in the File System Stanzas

Table 11 shows a list of all the attributes that can be defined in the directory, file, and link stanzas.

Table 11. SPD File attributes in file system stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
add	Specifies that an object must be created at install time **y**, or that it already exists **n**.			
	y: yes **n**: no	No	None	add_directory
compression_method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	add_directory, add_file
compute_crc	After installation of a file, specifies whether to compare the cyclic redundancy check (CRC) of the file installed on the target system with the corresponding file in the package (to check whether the file was corrupted during distribution or installation).			
	y: yes **n**: no	No	**n**	add_directory, add_file
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	add_directory, add_file, add_link
create_dirs	If set to **y**, create directories if they do not already exist on the target system.			
	y: yes **n**: no	No	**y**	add_directory, add_file

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
delta_compressible	Specifies whether the file must be delta compressed or not. You can specify this attribute at the directory level also. If set to **y**, indicates that the file must be delta compressed. If set to **n**, indicates that the file must not be delta compressed. If set to **d**, indicates that the file must be delta compressed only if it satisfies the following rules: • It is not located in one of the following folders: – common_files – system_ root – system_ dir – system 16_dir – windir • The following attributes have not been set: – remote – temporary – is_shared – substitute_variables • In a Windows, OS/2, or NetWare environment, it is a .exe, or a .dll, or a .nlm file. • In a UNIX environment, at least one execute mode has been set. **Note:** Files in the base package that have changed since installation cannot be reconstructed using this algorithm. If you select a file for delta compression and its base file has changed since installation, the installation fails. To avoid this situation, use the **n** option if there is a possibility that you are upgrading changed files. You can also use the **d** option to compress the files that satisfy the rules listed above. These are files, such as executables, which will not have been changed.			
	y: yes **n**: no **d**: default	No	**d**	add_directory, add_file
descend_dirs	Determines whether the add or remove command relates to the entire directory tree **y** or only to the top-level directory **n**. **Notes:** 1. If this attribute is set to y and the **is_signature** attribute is set to **d**, in case the files selected as signatures are not valid, you will get a warning message for all such files present in the nested directories. 2. If you set this attribute to **y** when creating packages containing symbolic links, the symbolic links are resolved, and the data they point to is inserted in the package in place of the links. To transfer a directory containing symbolic links to directories and files without resolving the links, you must respect the following rules: • Add directories to the package by setting the descend_dirs attribute to n. • Add links to the package by setting the follow_links and hard_link attributes to n.			
	y: yes **n**: no	No	**n**	add_directory

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
destination	The fully qualified path of the destination directory of the file or link to be created or removed. If specified in a nested stanza, it must be the destination path relative to the parent directory stanza. If not specified in a nested stanza, its default name is the name of the source object. It can include variables as well as wildcards, in the event that you want to rename the target object.			
	String	No	None	add_directory, add_file, add_link
fat_attributes	Specifies the attributes of the file or directory to be added. Any combination of the characters **r**, **a**, **h**, or **s** can be specified.			
	r: read only **a**: archive **h**: hidden **s**: system	No	Default value is read from file when it is built.	add_directory, add_file
follow_links	If set to **y**, the existing symbolic source link is followed and its source file is copied.			
	y: yes **n**: no	No	**y**	add_link
hard_link	If set to **y**, a hard link to the file specified in source_file is created. In the case of a hard link, the source file must exist. Otherwise, an error message is displayed and the install operation fails. Although not required, if a value for hard_link is not specified in the remove link stanza, the default value **n** is used. Setting the appropriate value enables the system to determine when creating a backup package if a hard or symbolic link is to be removed, for example, during undoable operations on the package containing the remove link stanza.			
	y: yes **n**: no	No	**n**	add_link
inventory_ description	Description of the software to be installed, for example, Lotus Notes			
	String	No	None	add_directory, add_file
inventory_ version	Version number of the software to be installed.			
	String expression	No	None	add_directory, add_file
is_shared	Set to **y** if the object is shared between software packages. See “Management of Shared Objects” on page 27.			
	y: yes **n**: no	No	**n**	add_directory, add_file

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
is_signature	Specifies whether the file can be used as an Inventory signature. Set to **y** if you want to consider the file as a signature and create it if not already present in the Inventory database. Set to **n** if the file system object is not an Inventory signature, set to **d** if you want to consider the file as a signature but only if the signature has been already created in the Inventory database. Some files cannot be specified as signatures. For example, a file shared among different applications (is_shared='y') cannot be a signature. For this reason, a consistency check is performed on each file marked as signature (is_signature='y') to establish whether the file can be accepted as a signature. If the file is invalid, a warning is issued and the signature attribute rejected. A file can be accepted as a signature in the following cases: • It is an executable file or a .DLL • It is not a temporary file • It is not shared • It does not belong to a system directory or a common files directory For more information, refer to *IBM Tivoli Configuration Manager: User's Guide for Software Distribution*.			
	n: no, **d**: check, **y**: yes (only for the add_file stanza)	No	**n**	add_directory, add_file
location	The fully qualified path of the source parent directory. This attribute can be specified in a top-level (unnested) stanza only. It cannot contain wildcards. **Note:** You should be aware that the root directory of a drive, for example g:\, has the attribute ″system hidden″. If you specify a root directory as the source location for adding a directory, the added directory is created as ″hidden″.			
	String	Yes	None	add_directory
name	The name (relative to the current location) of the source file or directory to be installed.			
	String	No	None	add_directory, add_file, add_link
netware_attributes	Specifies the netware attributes of the file or directory to be added. Any combination of the valid values can be specified			
	x: execute **b**: shareable **t**: transaction **p**: purge **n**: inhibit **d**: delete inhibit **c**: copy inhibit **h**: hidden **s**: system **r**: read-only **a**: archive	No	Default value is read from file when it is built.	add_directory, add_file

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
ntfs_attributes	Specifies the ntfs attributes of the file or directory to be added. Any combination of the valid values can be specified			
	r: read only **a**: archive **h**: hidden **s**: system **c**: compressed	No	Default value is read from file when it is built.	add_directory, add_file
remote	When set to **y**, the specified source files are not added to the software package. To help reduce network congestion, they are added at installation time from a network drive that is accessible from the target system. This attribute is not valid for the remove directory action.			
	y: yes **n**: no	No	**n**	add_directory, add_file
remove	In a remove stanza, specifies whether the directory must be deleted at installation time.			
	y: yes **n**: no	No	None	remove_directory
remove_empty_dirs	Specify **y** to remove empty directories when installing or removing a software package. Empty directories are removed upwards starting from the installation directory of the software package.			
	y: yes **n**: no	No	**y**	add_directory, add_file, remove_file
remove_extraneous	Specifies whether to remove all files and directories in an existing destination directory prior to installing a software package.			
	y: yes **n**: no	No	**n**	add_directory, add_file
remove_if_modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_directory, add_file, add_link
rename_if_locked	Specifies to rename files that are in use, so that the new copies will be effective the next time the application starts. Setting this attribute to **y** may avoid the need to reboot a workstation immediately after a distribution.			
	y: yes **n**: no	No	**n**	add_directory
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_directory, add_file, add_link

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
replace_if_newer	If set to **y**, specifies to replace a target object even if the target object is newer than the source object. For the implications of using this attribute on Windows platforms, see "Replacing Target Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_directory, add_file, add_link
shared_counter	Used in conjunction with **is_shared=y**, indicates how many times the reference counter associated with the shared file should be incremented. Any positive integer can be specified.			
	Integer	No	**1**	add_directory, add_file
source_file	If **follow_links=y**, the full pathname of a source file (link) on the source host. During the build, the link is followed to the file that is packed in the software package. At install time, this existing file is installed. If **follow_links=n**, the full pathname of the file on the target system that must be linked. For remove_link stanzas, specify the source file attribute only if the link to be removed is a hard link.			
	String	Yes	None	add_link remove_link
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_directory
substitute_variables	If set to **y**, opens the file and customizes it, substituting variables inside the file being installed on the target system.			
	y: yes **n**: no	No	**n**	directory, file
temporary	Set to **y** if the object must be removed during cleanup.			
	y: yes **n**: no	No	**n**	add_directory, add_file
translate	If set to **y**, the file is translated. Translation can be from EBCDIC to ASCII and vice versa or code page translation.			
	y: yes **n**: no	No	**n**	add_directory, add_file
unix_attributes	A string formatted as: owner, group, others, where each set of attributes is a combination of the characters: **r**, **w**, **x**, **s**, or **t**.			
	r: read **w**:write **x**: execute **s**: set user/group ID on execution **t**: save text images after execution	No	rwx, rx	add_directory, add_file

Table 11. SPD File attributes in file system stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
unix_group	The UNIX group of the object.			
	String	No	Default value is read from the SPD file.	add_directory, add_file
unix_group_id	The UNIX group ID.			
	Integer	No	None	add_directory, add_file
unix_owner	The owner of the object.			
	String	No	Default value is read from the SPD file.	add_directory, add_file
unix_user_id	The UNIX user ID.			
	Integer	No	None	add_directory, add_file
verify_crc	When checking the identity of the source and target files, specifies whether to compare their cyclic redundancy check (CRC) values as well. See “Replacing Target Objects” on page 27 for more information.			
	y: yes **n**: no	No	**n**	add_directory, add_file

SPD File Example: Adding File System Directory Objects

In this example, two system .dll files, one of which is shared, are added to the target system directory.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name = "Appsample"
  title = "Sample Software Package"
  version = "1.0"

  add_directory
    location = "C:\WINNT"
    name = "system32"
    destination = "$(system_dir)"
    add = y
    descend_dirs = n
    create_dirs = y

    add_file
      name = "App32.dll"
      destination = "App32.dll"
      is_shared = y
      shared_counter = 1
      create_dirs = y
    end


    add_file
      name = "Apprun32.dll"
```

```
      destination = "Apprun32.dll"
      is_shared = n
      create_dirs = y
    end
  end
end
```

Windows Profile Objects

Windows profile objects include sections and key value pairs of .ini files for all Windows operating systems. This section includes the format of each stanza where an add or remove operation for a Windows profile object can be defined.

win_profile_objects

The following definition shows the attributes that can be defined in an **add_win_profile_objects** stanza. For information about these attributes, see Table 12 on page 45.

```
add_win_profile_objects
     # Windows Profile Object defaults
     file                 = <pathname>

    # Inherited from action
     condition            = <string constrain>

     # Inherited from container
          stop_on_failure      = <Boolean>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     # Sequence of <section>
end
```

The following definition shows the attributes that can be defined in a **remove_win_profile_objects** stanza. For information about these attributes, see Table 12 on page 45.

```
remove_win_profile_objects
     # Inherited from action
      condition             = <string constrain>

     # Inherited from container
      stop_on_failure         = <Boolean>

     # Folder-specific attributes
      file                       = <pathname>

       # Sequence of <section>
end
```

sections

The following definition shows the attributes that can be defined in an **add_section** stanza. For information about these attributes, see Table 12 on page 45.

```
add_section
     # Specific attributes
           name                  = <string>

     # Inherited from action
     condition             = <string constrain>
```

```
     # Inherited from container
     stop_on_failure       = <Boolean>

     # Add object default attributes
     replace_if_existing   = <Boolean>
     remove_if_modified    = <Boolean>
     is_shared             = <Boolean>
     # Sequence of <item>
end
```

The following definition shows the attributes that can be defined in a **remove_section** stanza. For information about these attributes, see Table 12 on page 45.

```
remove_section

     # Inherited from action
     condition             = <string constrain>

     # Inherited from container
     stop_on_failure       = <Boolean>

     # Specific attributes
     name                  = <string>

     # Sequence of <item>
end
```

items

The following definition shows the attributes that can be defined in an **add_item** stanza. For information about these attributes, see Table 12 on page 45.

```
add_item
     condition             = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     # Specific attributes
     key                 = <string>
     value               = <string>
     duplicate           = <Boolean>
end
```

The following definition shows the attributes that can be defined in a **remove_item** stanza. For information about these attributes, see Table 12 on page 45.

```
remove_item
     # Remove object default attributes
     condition           = <string constrain>

     # Specific attributes
     key                 = <string>
     value               = <string, buffer>

end
```

Attributes in Windows Profile Object Stanzas

Table 12 on page 45 shows a list of all the attributes that can be defined in the win_profile_objects, section, and item stanzas.

Table 12. SPD file attributes in Windows profile object stanzas

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
add	Specifies that a profile section must be created at install time (**y**), or that it already exists (**n**).			
	y: yes **n**: no	No	None	add_section
condition	A valid expression (see "Defining Dependencies and Conditions " on page 8 for more information).			
	String constrain	No	None	add_section add_win_profile_objects
duplicate	If set to **n**, and the key already exists in the .ini file, its value is updated. If set to **y**, the same key with the current value specified in the SPD file is added, even if it is already present in the .ini file with a different value.			
	y: yes **n**: no	Yes	**y**	add_item
file	Full path of the file to update. This is the Windows.ini file.			
	String	Yes	None	add_win_profile_objects
is_shared	Set to **y** if the object is shared between software packages. See "Management of Shared Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_item, add_section, add_win_profile_objects
key	The key to be added or removed.			
	String	No	None	add_item remove_item
name	A unique identifier. The section object must be uniquely identified by its name. The package is identified uniquely by name and version.			
	String	Yes	None	add_section
remove	In a remove stanza, specifies whether the section must be deleted at installation time.			
	y: yes **n**: no	No	None	remove_section
remove_if_ modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_item, add_section, add_win_profile_ objects
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_item, add_section, add_win_profile_ objects

Table 12. SPD file attributes in Windows profile object stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
replace_if_newer	If set to **y**, specifies to replace a target object even if the target object is newer than the source object. For the implications of using this attribute on Windows platforms, see "Replacing Target Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_win_profile_ objects
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_section, add_win_profile_objects
value	Value of the profile item being added, deleted, or updated. Either this attribute or the key attribute (or both) are required.			
	String	No	None	add_item remove_item

SPD File Example: Adding Windows Profile Objects

In this example, changes are made to a Windows profile by modifying a .ini file.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name = "Appsample"
  title = "Sample Software Package"
  version = "1.0"

  default_variables
    target_dir = "$(system_drive)\Appsample"
  end

  add_win_profile_objects
    file = "$(system_root)\Applesample.ini"
    stop_on_failure = y
    condition = "$(os_name)  == Windows_NT"

    add_section
      add = y
      name = "1.0"

      add_item
        key = "InstallationDirectory"
        value = "$(target_dir)"
        duplicate = y
     end
    end
  end
end
```

Windows Shell Objects

Windows shell objects include folders and links (shortcuts) for all supported Windows operating systems. This section includes the formats of stanzas where

win_shell_folder

The following definition shows the attributes that can be defined in an **add_win_shell_folder** stanza. For information about these attributes, see Table 13 on page 48.

```
add_win_shell_folder
     # Inherited from action
     condition            = <string constrain>
     is_per_user          = <Boolean>

     # Inherited from container
     stop_on_failure      = <Boolean>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     # Windows Folder attributes
     location             = <pathname>
     display_name         = <string>
     add                  = <Boolean>

     # Sequence of <win_shell_folder, link>
end
```

The following definition shows the attributes that can be defined in a **remove_win_shell_folder** stanza. For information about these attributes, see Table 13 on page 48.

```
remove_win_shell_folder
     # Inherited from action
     condition       = <string constrain>
     is_per_user          = <Boolean>

     # Inherited from container
     stop_on_failure = <Boolean>

     # Windows Folder attributes
     location        = <pathname>
     display_name    = <string>
     remove          = <Boolean>

     # Sequence of <win_shell_folder, link>
end
```

link

The following definition shows the attributes that can be defined in an **add_link** stanza. For information about these attributes, see Table 13 on page 48.

```
add_link
     display_name         = <string>
     command              = <pathname>
     arguments            = <string>
     working_dir          = <pathname>
     hotkey               = <integer>
     show                 = <normal, minimized, maximized>
     icon_location        = <pathname>
     icon_index           = <integer>

     # Inherited from action
     condition            = <string constrain>


     # Add object default attributes
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

end
```

The following definition shows the attributes that can be defined in a **remove_link** stanza. For information about these attributes, see Table 13.

```
remove_link
        display_name         = <string>

        # Remove object default attributes
        condition            = <string>

end
```

Attributes in Windows Shell Object Stanzas

Table 13 shows a list of all the attributes that can be defined in the win_shell_folder and link stanzas.

Table 13. SPD file attributes in Windows shell object stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
add	Specifies that a Windows folder or link, must be created at install time **y**, or that it already exists **n**.			
	y: yes **n**: no	No	None	add_win_shell_folder
arguments	Arguments for the command.			
	String	No	None	add_link
command	Full path for the program.			
	String	Yes	None	add_link
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	add_win_shell_folder, remove_win_shell_ folder, add_link, remove_link
display_name	For Windows shell folders and links, the name of this folder or link on the target system.			
	String	No	None	add_win_shell_folder, remove_win_shell_ folder, add_link remove_link
hotkey	Integer that maps to a key combination for starting the icon for the link, as follows: **112 to 123** Maps keys from F1 to F12 **1584 to 1593** Maps Ctrl+Alt+*n*, where *n* is a digit from 0 to 9. **1601 to 1626** Maps Ctrl+Alt+*x*, where *x* is a letter from A to Z.			
	Integer	No	None	add_link
icon_index	The index of the icon in the resource file. If this attribute is not specified, icon_location is treated as an icon file.			
	Integer	No	0	add_link

Table 13. SPD file attributes in Windows shell object stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
icon_location	Specifies the path to the file containing the icon.			
	String	No	None	add_link
is_per_user	Set to **n** if the folder and all its contained commands are common to all users. Set to **y** if the folder and its contents apply to the logged-on user only and are installed at the next user logon by the user profile update program (**wdusrprf.exe**). (This attribute is valid for the Windows 2000 and Windows XP platforms only.) **Note:** In Windows 2000 the user must be a member of the Power Users group.			
	y: yes **n**: no	No	**n**	add_win_shell_folder, remove_win_shell_ folder
is_shared	Set to **y** if the object is shared between software packages. See “Management of Shared Objects” on page 27.			
	y: yes **n**: no	No	**n**	add_link add_win_shell_folder
location	The folder where the folder or link is to be added or removed.			
	String	Yes	None	add_win_shell_folder remove_win_shell_ folder
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_link, add_win_shell_folder
remove_if_ modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_link, add_win_shell_folder
replace_if_newer	If set to **y**, specifies to replace a target object even if the target object is newer than the source object. For the implications of using this attribute on Windows platforms, see “Replacing Target Objects” on page 27.			
	y: yes **n**: no	No	**n**	add_win_shell_folder
show	Display mode of the command.			
	normal, minimized	No	**normal**	add_link
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_win_shell_folder
working_dir	Working directory for the command.			
	String	No	Command directory	add_link

SPD File Example: Adding Windows Shell Folder Objects

In this example, a new Windows shell folder containing a shortcut is added to the Windows desktop.

```
"TIVOLI Software Package v4.3.1 - SPDF"package
  name = "Appsample"
  title = "Sample Software Package"
  version = "1.0"

  default_variables
    target_dir = "$(system_drive)\Appsample"
  end

  add_win_shell_folder
    condition = "$(os_name)  == Windows_NT AND $(os_release) >  3"

    ## Creates folder 'Tivoli Appsample' to the desktop
    display_name = "Tivoli Appsample"
    location = "$(all_users_shell_desktop)"
    add = y
    is_per_user = n
    stop_on_failure = y

    ## Add a link to 'Tivoli Appsample'
    add_link
      display_name = "Appsample 1.0"
      command = "$(target_dir)\bin\Appsample.exe"
      working_dir = "$(target_dir)\bin"
      hotkey = 0
      show = "normal"
      icon_index = 0
    end

    ## Add a nested folder to 'Tivoli Appsample'
    add_win_shell_folder
      display_name = "Appsample Folder"
      location = "$(all_users_shell_desktop)\Tivoli Appsample"
      add = y
      is_per_user = n
    end
  end
end
```

The following figure shows the result after this SPD file example is run on a sample machine.

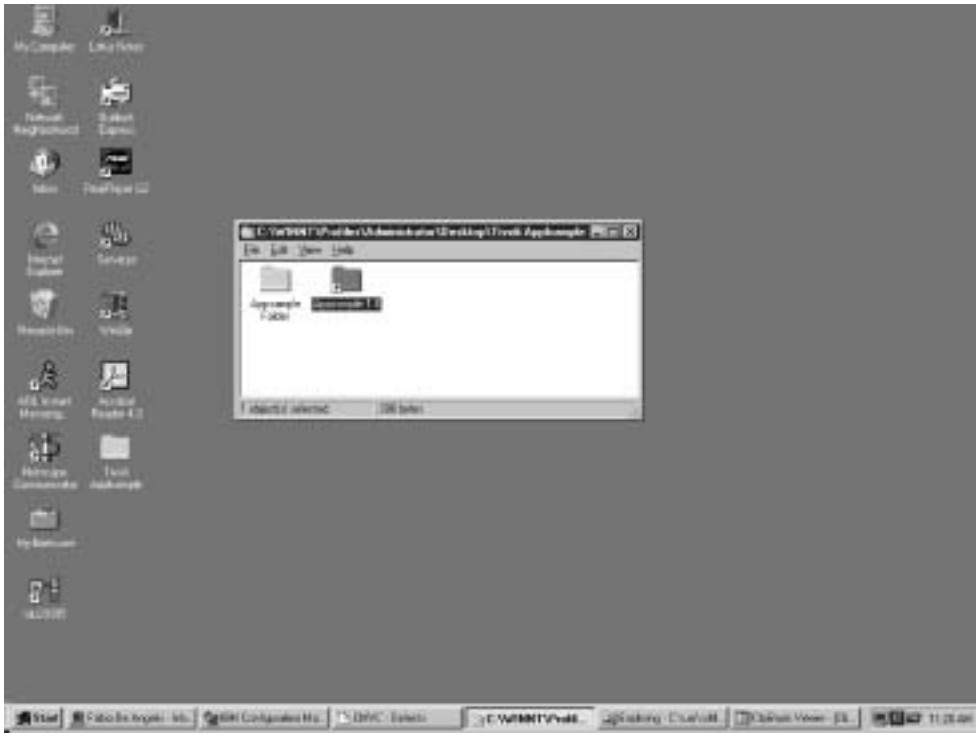


Figure 5. Output of the SPD file example for adding Windows shell folders

Windows Registry Objects

Windows registry objects include keys and values for the all supported Windows operating systems. These objects specify configuration information regarding the user, hardware, and the programs and applications installed on a system. This section includes the format of each Windows registry objects stanza.

win_registry_key

The following definition shows the attributes that can be defined in an **add_win_registry_key** stanza. For information about these attributes, see Table 14 on page 53.

```
add_win_registry_key
     # Inherited from action
     condition            = <string constrain>

     # Inherited from container
      stop_on_failure      = <Boolean>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     # Windows Registry Key attributes
     parent_key           = <string>
     key                  = <string>
     class                = <string>
     add                  = <Boolean>
     override_permissions = <Boolean>

     # Sequence of <win_registry_key, value>
end
```

The following definition shows the attributes that can be defined in a **remove_win_registry_key** stanza. For information about these attributes, see Table 14 on page 53.

```
remove_win_registry_key
     # Inherited from action
     condition            = <string constrain>

     # Inherited from container
     stop_on_failure      = <Boolean>

     # Windows Registry Key attributes
     parent_key           = <string>
     key                  = <string>
     override_permissions = <Boolean>

     # Sequence of <win_registry_key, value>
end
```

value

The following definition shows the attributes that can be defined for an **add_value** stanza, nested within a win_key_registry stanza. For information about these attributes, see Table 14 on page 53.

```
add_value
     name                 = <string>
     type                 = <binary, dword, expand_string,
       multi_string, string>
     position             = <begin, end, replace>
     data                 = <string, buffer>
     replace_if_existing = <Boolean>
     remove_if_modified  = <Boolean>
     is_shared            = <Boolean>
     override_permissions = <Boolean>

end
```

The following definition shows the attributes that can be defined for a **remove_value** stanza, nested within a win_key_registry stanza. For information about these attributes, see Table 14 on page 53.

```
remove_value
     name                 = <string>

   # Remove object default attributes
     condition            = <string>
     override_permissions = <Boolean>

end
```

Attributes in Windows Registry Object Stanzas

Table 14 on page 53 shows a list of all the attributes that can be defined in the win_registry_key and value stanzas.

Table 14. SPD file attributes for Windows registry objects

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
add	Specifies that a directory, profile section, Windows folder or link, or Windows registry key must be created at install time **y**, or that it already exists **n**. **Note:** To override this setting, add the _always_add_winreg_keys_ variable to the `swdis.var` file and set it to YES or NO. If you set it to YES, parent registry keys are always created, irrespective of the setting specified for the **add** attribute.			
	y: yes **n**: no	No	None	add_win_registry_key
class	Specifies the class of the key.			
	String	No	None	add_win_registry_key
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	add_win_registry_key, remove_win_registry_key, add_value, remove_value
data	Interpreted as a binary buffer if the type attribute is specified as binary, dword, or multi_string data; otherwise, it will be interpreted as a string.			
	String, binary buffer	No	None	add_value
is_shared	Set to **y** if the object is shared between software packages. See “Management of Shared Objects” on page 27.			
	y: yes **n**: no	No	**n**	add-value, add_win_registry_key
key	The key to be added or removed.			
	String	No	None	add_win_registry_key, remove_win_registry_key
name	This attribute is optional in the value stanzas and automatically defaults to the value of the parent key.			
	String	See comments	Value of the parent key in the parent win_registry _key stanza	remove_value, add_value
override_ permissions	If set to **y**, access permissions for adding or removing protected registry keys and values are temporarily overridden, then restored after the operation is completed. To modify these protected entries, the process running the Software Distribution method must have administrator privileges. On a disconnected system, the logged-on user must be the administrator of the system.			
	y: yes **n**: no	No	**n**	add_win_registry_key remove_win_registry_key add_value remove_value

Table 14. SPD file attributes for Windows registry objects (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
parent_key	Specifies the parent key of a Windows registry key that is being added or removed.			
	String	Yes	None	add_win_registry_key add_win_registry_key
position	Specify **replace** if the value or item must be replaced on the target. The **begin** and **end** values are generally used for expand_string type values.			
	replace, **begin**, **end**	No	**replace**	add_value
remove_if_modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_value add_win_registry_key
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_value add_win_registry_key
replace_if_newer	If set to **y**, specifies to replace a target object even if the target object is newer than the source object. For the implications of using this attribute on Windows platforms, see "Replacing Target Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_value
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_win_registry_key, remove_win_registry_key
type	Specifies the configuration data type, either as a string (text, displayed within quotes), as a binary value (of unlimited size, in hexadecimal format) or as a dword value (4-byte binary value, in decimal format).			
	Binary, **dword**, **expand_string**, **multi_string**, **string**	No	String	add_value

SPD File Example: Windows Registry Objects

In this example, a new Windows registry key and value for Windows 2000 target systems are added to the container.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                    = "Appsample"
  title                   = "Sample Software Package"
  version                 = "1.0"
  undoable                = "o"
  committable             = "o"
  history_reset           = n
  save_default_variables  = n
  creation_time           = "2002-01-17 11:06:54"
```

```
  last_modification_time  = "2002-01-17 11:36:31"

  default_variables
     target_dir           = "$(system_drive)\Appsample"
  end

  add_win_registry_key
    condition             = "$(os_name)  == Windows_NT
    parent_key            = "HKEY_LOCAL_MACHINE\SOFTWARE\Tivoli"
    key                   = "Appsample"
    add = y
    override_permissions  = n
    stop_on_failure       = y
    name                  = "Appsample"

    add_value
      name                = "String"
      type                = "string"
      position = "replace"
      data = "Appsample String"
    end

    add_value
      name = "Installation Directory"
      type = "expand_string"
      position = "replace"
      data = "$(target_dir)"
    end

    add_value
      name = "Appsample binary"
      type = "binary"
      position = "replace"
      data = "ff1ff0"
    end

    add_value
      name = "Appsample DWORD"
      type = "dword"
      position = "replace"
      data = 64497
    end

    add_value
      name = "Appsample Multistring"
      type = "multi_string"
      position = "replace"
      data = "32333333  34352036  36363737  38203938"
    end
  end
end
```

The following figure shows the result after this SPD file example is run on a sample machine.

Figure 6. Output of the SPD file example for Windows registry objects

Windows Services Objects

Windows services objects allow you to install and remove services on a Windows NT, Windows 2000, Windows XP and Windows 2003 operating system. To use these operations, the Software Distribution engine must run with administrator authority. This section includes the formats of the stanzas that add and remove Windows Services objects.

win_nt_service

The following definition shows the attributes that can be defined for an **add_win_nt_service** stanza. For information about these attributes, see Table 15 on page 57.

```
add_win_nt_service
     # Inherited from action
     condition            = <string constrain>


     # Windows NT Service attributes
     add                   = <Boolean>
     name                  = <string>
     display_name          = <string>
     path                  = <pathname>
     type                  = <win32_own_process,
       <win32_share_process, kernel_driver,
       file_system_driver>
     interact_with_desktop = <Boolean>
     start_type            = <boot, system, auto, demand,
       disabled>
     error_control         = <ignore, normal, severe,
       critical>
     load_ordering_group   = <string>
     dependency_groups     = <string expression>
     dependency_services   = <string expression>

    end
```

The following definition shows the attributes that can be defined for a **remove_win_nt_service** stanza. For information about these attributes, see Table 15.

```
remove_win_nt_service
     # Inherited from action
     condition            = <string constrain>

     # Windows NT Service attributes
     name                 = <string>

 end
```

Attributes in Windows Services Stanzas

Table 15 shows a list of all the attributes that can be defined in win_nt_services stanzas.

Table 15. SPD file attributes in Windows services stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
add	Specifies whether a Windows service must be created at install time or whether it already exists.			
	y: yes **n**: no	No	None	add_win_nt_service
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	add_win_nt_service
dependency_groups	The list of dependency load-ordering groups for the service. The list is in the format: *group1*, *group2*, *group3*, and so on.			
	String expression	No	None	win_nt_service
dependency_services	The list of services that must be started before the current service. The list is in the format: *service1*, *service2*, *service3*, and so on.			
	String expression	No	None	win_nt_service
display_name	For Windows shell folders and links, the name of this folder or link on the target system. For Windows NT service objects, the name of the service as shown when you click **Control Panel**, and then click **Services**.			
	String	No	None	win_nt_service
error_control	Specifies the severity of the error if this service fails to start during startup, and determines the action taken by the startup program if failure occurs.			
	ignore, **normal**, **severe**, **critical**	No	**normal**	win_nt_service
interact_with_ desktop	Specifies whether to allow the service to interact with the desktop.			
	y: yes **n**: no	No	**n**	win_nt_service
load_ordering_group	Specifies the service as a member of a load-ordering group from the registry.			
	String	No	None	win_nt_service
name	The name of the service.			
	String	Yes	None	win_nt_service

Table 15. SPD file attributes in Windows services stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
path	The fully qualified pathname of the binary file associated with the service.			
	String	Yes	None	add_win_nt_service
start_type	Indicates the circumstances for starting the service.			
	boot, **system**, **auto**, **demand**, **disabled**	No	**demand**	add_win_nt_service
type	The service type.			
	win32_own_process, **win32_share_process**, **kernel_driver**, **file_system_driver**	No	**win32_own_process**	win_nt_service

SPD File Example: Adding Windows Services Objects

In this example, a Windows NT services action is added to a container.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                        = "Appsample"
  title                       = "Sample Software Package"
  version                     = "1.0"

  default_variables
    target_dir                = "$(system_drive)\Appsample"
  end

  add_win_nt_service
    name                      = "TivoliAppsample"
    display_name              = "Tivoli Appsample Service"
    path                      = "$(target_dir)\bin\AppSrv.exe"
    type                      = "win32_own_process"
    interact_with_desktop     = y
    start_type                = "demand"
    error_control             = "normal"
  end
end
```

OS/2 Profile Objects

With OS/2 profile objects, you can add or remove keys on binary OS/2 profile files, such as os2.ini or os2sys.ini. In this section, the format of each OS/2 profile objects stanza, as well as the details on each attribute of OS/2 profile objects, are provided.

Note: Some OS/2 applications use text (non-binary) profile files instead of OS/2 binary profiles. To handle such text files, you should use the actions provided by the text file objects stanza (see “Text File Objects” on page 69 for details).

os2_profile_objects

The following definition shows the attributes that can be defined for an **add_os2_profile_objects** stanza. For information about these attributes, see Table 16 on page 60.

```
add_os2_profile_objects
     # Inherited from action
     condition            = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     # Inherited from container
     stop_on_failure      = <Boolean>

     # OS/2 Profile Object defaults
     file                 = <pathname>

     # Sequence of <item>
end
```

The following definition shows the attributes that can be defined for a **remove_os2_profile_objects** stanza. For information about these attributes, see Table 16 on page 60.

```
remove_os2_profile_objects
     # Inherited from action
     condition            = <string constrain>

     # Inherited from container
     stop_on_failure     = <Boolean>

     # OS/2 Profile Object defaults
     file                = <pathname>

     # Sequence of <item>
end
```

item

The following definition shows the attributes that can be defined for an **add_item** stanza, nested within an **os2_profile_objects** stanza. For information about these attributes, see Table 16 on page 60.

```
add_item

     # Inherited from action
     condition            = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     section             = <string>
     key                 = <string>
     type                = <string, binary>
     position            = <replace, begin, end>
     value               = <string, buffer>

end
```

The following definition shows the attributes that can be defined for a **remove_item** stanza, nested within an **os2_profile_objects** stanza. For information about these attributes, see Table 16 on page 60.

```
remove_item
     # Remove object default attributes
     condition           = <string constrain>
```

```
    # Specific attributes
    section             = <string>
    key                 = <string>

end
```

Attributes in OS/2 Profile Stanzas

Table 16 shows a list of all the attributes that can be defined in the os2_profile_objects and item stanzas.

Table 16. SPD file attributes in OS/2 profile stanzas

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
condition	A valid expression (see "Defining Dependencies and Conditions " on page 8 for more information).			
	String constrain	No	None	add_os2_profile_objects remove_os2_profile_ objects add_item remove_item
file	Full path of the OS/2 profile to update.			
	String	Yes	None	add_os2_profile_objects remove_os2_profile_ objects
is_shared	Set to **y** if the object is shared between software packages. See "Management of Shared Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_item, add_os2_profile_objects
key	Name of the key (case-sensitive).			
	String	Yes	None	add_item remove_item
position	Specify **replace** if the value or item must be replaced on the target. The **begin** and **end** values are generally used for expand_string type values.			
	replace, **begin**, **end**	No	**replace**	add_item
remove_if_ modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_item, add_os2_profile_objects
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_item, add_os2_profile_objects
section	Name of the section.			
	String	Yes	None	add_item remove_item

Table 16. SPD file attributes in OS/2 profile stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_os2_profile_objects remove_os2_profile_objects
type	Type of the key value.			
	string, **binary**	No	**string**	add_item
value	Specifies the key value.			
	string, **buffer**	No	**string**	add_item

SPD File Example: Adding OS/2 Profile Objects

In this example, an OS/2 profile object is created that adds an item containing a new section and key by modifying a .ini file.

```
"TIVOLI software package v4.2  SPDF"
package
  name                    = "Appsample"
  title                   = "Sample Software Package"
  version                 = "1.0"

  default_variables
    target_dir            = "$(system_drive)\Appsample"
  end

  add_os2_profile_objects

    condition             = "$(os_name) == 'OS/2'"

      file                = "$(system_root)\appsample.ini"

      add_item
        section           = "1.0"
        key               = "InstallationDirectory"
        type              = "string"
        position          = "replace"
        value             = "$(target_dir)"
      end
    end
  end
end
```

The following figure shows the result when this SPD file example is run on a sample machine.

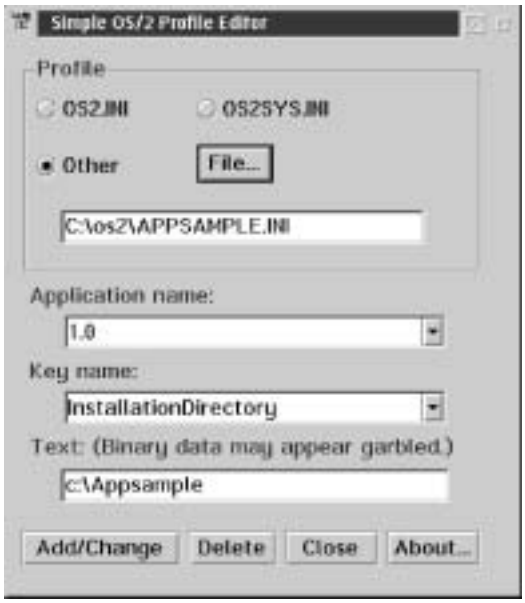


Figure 7. Output of the SPD file example for adding OS/2 profile objects

OS/2 Desktop Objects

OS/2 desktop objects include folders, programs, shadows, and generic objects for the OS/2 3.0, and later, operating systems. In this section, the format of each OS/2 desktop objects stanza, as well as the details on each attribute of OS/2 desktop objects, are provided.

os2_desktop_folder

The following definition shows the attributes that can be defined for an **add_os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
add_os2_desktop_folder
     # Inherited from action
     condition            = <string constrain>

      # Inherited from container
      stop_on_failure         = <Boolean>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

      # Folder-specific attributes
      add                     = <Boolean>
      location                = <string expression>
      title                   = <string>
      object_id               = <string expression>
      icon_location           = <pathname>
      icon_index              = <integer>
      template                = <Boolean>
      background_image_file   = <pathname>
      background_image_mode   = <normal>
      background_color        = <string expression>
      scaling_factor          = <integer>
      animation_icon_location = <pathname>
      animation_icon_index    = <integer>
      default_view            = <icon, tree, details>
```

```
      # Sequence of <os2_desktop_folder, object, program, shadow>

end
```

The following definition shows the attributes that can be defined for a **remove_os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
remove_os2_desktop_folder
     # Inherited from action
     condition            = <string constrain>

     # Inherited from container
     stop_on_failure      = <Boolean>

     # Folder-specific attributes
     location             = <string expression>
     title                = <string>
     object_id            = <string expression>
     remove               = <Boolean>
     # Sequence of <os2_desktop_folder, object, program, shadow>

end
```

object

The following definition shows the attributes that can be defined for an **add_object** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
add_object
     # Inherited from action
     condition            = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     class                = <string>
     title                = <string>
     object_id            = <string expression>
     icon_location        = <pathname>
     icon_index           = <integer>
     template             = <Boolean>
     setup_string         = <string expression>

end
```

The following definition shows the attributes that can be defined for a **remove_object** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
remove_object
     title                = <string>
     object_id            = <string expression>

     # Remove object default attributes
     condition            = <string>

end
```

program

The following definition shows the attributes that can be defined for an **add_program** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
add_program# Inherited from action
     condition            = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>
     title                    = <string>
     object_id                = <string expression>
     icon_location            = <pathname>
     icon_index               = <integer>
     template                 = <Boolean>

     command                  = <pathname>
     arguments                = <string>
     working_dir              = <pathname>
     type                     = <see table>

end
```

The following definition shows the attributes that can be defined for a **remove_program** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
remove_program
     title                    = <string>
     object_id                = <string expression>

     # Remove object default attributes
     condition            = <string>

end
```

shadow

The following definition shows the attributes that can be defined for a **add_shadow** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
add_shadow
     # Inherited from action
     condition            = <string constrain>

     # Add object default attributes
     replace_if_existing  = <Boolean>
     replace_if_newer     = <Boolean>
     remove_if_modified   = <Boolean>
         is_shared            = <Boolean>

     object_id                = <string expression>

     shadowed_object_id       = <string expression>
     shadowed_object_title    = <string>
     shadowed_object_location  = <string expression>

     end
```

The following definition shows the attributes that can be defined for a **remove_shadow** stanza, nested within an **os2_desktop_folder** stanza. For information about these attributes, see Table 17 on page 65.

```
remove_shadow

     object_id                = <string expression>

     shadowed_object_id       = <string expression>
     shadowed_object_title    = <string>
     shadowed_object_location  = <string expression>

     # Remove object default attributes
     condition            = <string>

end
```

Attributes in OS/2 Desktop Stanzas

Table 17 shows a list of all the attributes that can be defined in the os2_desktop_folder, object, program, and shadow stanzas.

Table 17. SPD file attributes in OS/2 desktop stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
add	Specifies that an OS/2 desktop folder must be created at install time **y**, or that it already exists **n**.			
	y: yes **n**: no	No	None	add_os2_desktop_folder
animation_icon_index	If this attribute is not specified, animation_icon_location is treated as an icon file instead of a resource file.			
	Integer	No	**0**	os2_desktop_folder
animation_icon_location	Full path of the icon file to be used as an animation icon.			
	String	No	None	os2_desktop_folder
arguments	Arguments to the program			
	String	No	None	program
background_color	Specifies the percent of RGB colors in the form: **r**, **g**, **b**. This attribute cannot be specified if background_image_file has been specified.			
	r, **g**, **b**	No	None	os2_desktop_folder
background_image_file	File containing the background image for the folder.			
	String	No	None	os2_desktop_folder
background_image_mode	How the background is displayed. This attribute cannot be specified if background_color has been specified.			
	normal, **tiled**, **scaled**	No	**normal**	os2_desktop_folder
class	A valid WP SOM class			
	String	Yes	None	object
command	Full name of the program			
	String	Yes	None	program

Table 17. SPD file attributes in OS/2 desktop stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
condition	A valid expression (see "Defining Dependencies and Conditions " on page 8 for more information).			
	String constrain	No	None	add_os2_desktop_ folder add_object add_program add_shadow
default_view	How the folder is displayed.			
	icon, **tree**, **details**	No	**icon**	os2_desktop_folder
icon_index	The index of the icon in the resource file. If this attribute is not specified, icon_location is treated as an icon file.			
	Integer	No	**-1**	os2_desktop_folder, object, program
icon_location	Specifies the path to the file containing the icon.			
	String	No	None	os2_desktop_folder, object, program
is_shared	Set to **y** if the object is shared between software packages. See "Management of Shared Objects" on page 27.			
	y: yes **n**: no	No	**n**	add_os2_desktop_ folder add_object, add_program, add_shadow
location	The full path or the WP SOM object ID of the location folder.			
	String expression	Yes	None	add_os2_desktop_ folder
object_id	The WP SOM object ID of the object. It must be a string enclosed between brackets (<>).			
	String expression	No	None	add_os2_desktop_ folder add_object, add_program, add_shadow
remove	In a remove stanza, specifies whether the OS/2 desktop folder must be deleted at installation time.			
	y: yes **n**: no	No	None	remove_os2_ desktop_folder
remove_if_ modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	add_os2_desktop_ folder add_object, add_program, add_shadow

Table 17. SPD file attributes in OS/2 desktop stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	add_os2_desktop_folder add_object, add_program, add_shadow
scaling_factor	Meaningful only if background_image_mode is set to **scaled**.			
	Integer	No	None	add_os2_desktop_folder
setup_string	A valid setup string for the object			
	String expression	No	None	add_object
shadowed_object_id	The WP SOM object ID or the full path of the shadowed object (if it is of the file system type). Use shadowed_object_title and shadowed_object_location when the shadowed object does not have an associated object ID and it is not a file system object.			
	String expression	No	None	add_shadow
shadowed_object_location	The WP SOM object ID or the full path of the folder containing the shadowed object. This attribute can be used together with shadowed_object_title.			
	String expression	No	None	add_shadow
shadowed_object_title	Title of the shadowed object. This attribute can be used together with shadowed_object_location.			
	String	No	None	add_shadow
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	add_os2_desktop_folder, remove_os2_desktop_folder
template	Set to **y** if the folder is a template.			
	y: yes **n**: no	No	**n**	add_os2_desktop_folder add_object, add_program
title	The title of the folder.			
	String	No	None	add_os2_desktop_folder add_object, add_program

Table 17. SPD file attributes in OS/2 desktop stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
type	Type of program			
	fullscreen, **pm**, **standard_ compatibility_ fullscreen**, **enhanced_ compatibility_ fullscreen**, **winos2_ enhanced_ common_ session**, **winos2_ enhanced_ separate_ session**, **standard_ compatibility_ fullscreen31,** **winos2_ standard_ common_ session,** **winos2_ standard_ separate_ session**, **winos2_ separate _vdm**, **dos_fullscreen,** **winos2_ fullscreen,** **os2 _window**, **dos_window**, **winos2_ window**	No	**os2_window**	program
working_dir	Working directory of the program.			
	String	No	Command directory	program

SPD File Example: Adding OS/2 Desktop Folder Objects

In this example, a folder containing a program and a shadow are created on the desktop.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name = "Appsample"
  title = "Sample Software Package"
  version = "1.0"

  default_variables
    target_dir = "$(system_drive)\Appsample"
  end
```

```
  add_os2_desktop_folder

    condition = "$(os_name)  == 'OS/2'"

      location = "$(os2_desktop)"
      title = "Tivoli Appsample"
      object_id = "<TIVOLI_APPSAMPLE>"
      template = n
      add = y
      default_view = "icon"
      stop_on_failure = y
      name = "$(os2_desktop)\Tivoli Appsample"

      add_program
        template = n
        command = "$(target_dir)\bin\Appsample.exe"
        working_dir = "$(target_dir)\bin"
        type = "pm"
      end


      add_shadow
        shadowed_object_title = "README.TXT"
        shadowed_object_location = "$(target_dir)"
      end
    end
  end
end
```

The following figure shows the result after this SPD file example is run on a sample machine.

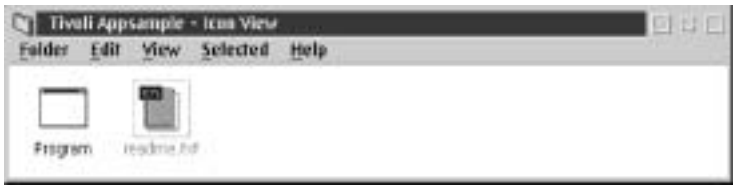


Figure 8. Output of the SPD file example for adding OS/2 desktop folder objects

Text File Objects

Text file objects include the following:

- Generic text lines in a file, such as the following statement in an autoexec.bat file:

  ```
  REM Application statement follows
  ```

- Command lines in a text file, such as the following statement in a config.sys file:

  ```
  device = c:\dos\smartdrv.sys 1024
  ```

- Tokens in a statement, like the c:\apps string in the following statement in an autoexec.bat file:

  ```
  set path = c:\dos;c:\word;c:\apps
  ```

In this section, the format of each text file objects stanza, as well as the details on each attribute of text file objects, are provided.

text_file_objects

The following definition shows the attributes that can be defined for an **add_text_file_objects** stanza. For information about these attributes, see Table 18 on page 72.

```
add_text_file_objects
     # Inherited from action
     condition              = <string constrain>

     # Inherited from container
     stop_on_failure        = <Boolean>

     # Add object default attributes
     replace_if_existing    = <Boolean>
     remove_if_modified     = <Boolean>
     is_shared              = <Boolean>

     # Text File Defaults
     file                   = <pathname>

     # Sequence of <line, command, token>
end
```

The following definition shows the attributes that can be defined for a **remove_text_file_objects** stanza. For information about these attributes, see Table 18 on page 72.

```
remove_text_file_objects
     # Inherited from action
     condition           = <string constrain>

     # Inherited from container
     stop_on_failure     = <Boolean>

     # Text File Defaults
     file                = <pathname>
     # Sequence of <line, command, token>
end
```

line

The following definition shows the attributes that can be defined for **add_line** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
add_line
     text                = <string>
     pattern             = <string expression>
     position            = <begin, end, after_first,
       before_first, after_last, before_last>
     is_shared           = <Boolean>

end
```

The following definition shows the attributes that can be defined for **remove_line** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
remove_line
     text                = <string>
     pattern             = <string expression>
     position            = <begin, end, after_first,
       before_first, after_last>

     # Remove object default attributes
     condition            = <string>

end
```

command_line

The following definition shows the attributes that can be defined for **add_command_line** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
add_command_line
     text                  = <string>
     pattern               = <string expression>
     position              = <begin, end, after_first,
       before_first, after_last>
     command               = <string>
     key                   = <string>
     replace_if_existing   = <Boolean>
     remove_if_modified    = <Boolean>
     is_shared             = <Boolean>
     condition             = <string>

end
```

The following definition shows the attributes that can be defined for **remove_command_line** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
remove_command_line
     text                  = <string>
     pattern               = <string expression>
     position              = <begin, end, after_first,
       before_first, after_last>
     command               = <string>
     key                   = <string>

     # Remove object default attributes
     condition            = <string>

end
```

token

The following definition shows the attributes that can be defined for **add_token** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
add_token
     text                 = <string>
     position             = <begin, end>
     key                  = <string>
                    token_separator      = <string>
     replace_if_existing  = <Boolean>
     remove_if_modified   = <Boolean>
     is_shared            = <Boolean>

end
```

The following definition shows the attributes that can be defined for **remove_token** stanza, nested within a text_file_objects stanza. For information about these attributes, see Table 18 on page 72.

```
remove_token
     text                 = <string>
     key                  = <string>
     token_separator      = <string>


     # Remove object default attributes
     condition            = <string>

end
```

Attributes in Text File Stanzas

Table 18 shows a list of all the attributes that can be defined in the text_file_objects, line, command_line, and token stanzas.

Table 18. SPD file attributes in text file stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
command	The name of the command. For example, for DEVICE = c:\os2\driver.sys, the command is DEVICE.			
	String	Yes	None	command_line
condition	A valid expression (see "Defining Dependencies and Conditions " on page 8 for more information).			
	String constrain	No	None	text_file_objects remove_line remove_command_ line remove_token
file	The text file to update.			
	String	Yes	None	text_file_objects
is_shared	Set to **y** if the object is shared between software packages. See "Management of Shared Objects" on page 27.			
	y: yes **n**: no	No	**n**	command_line, line, text_file_objects, token
key	In command_line stanzas, a key identifying the command. For example, for DEVICE = c:\os2\driver.sys, the key is driver.sys. Be sure to specify a key path that uniquely identifies the command, in case there are multiple devices with the same name. In token stanzas, the environment variable where the token must be added, for example, *path*.			
	String	Yes	None	command_line
name	The name of the text file.			
	String	Yes	None	text_file_objects
pattern	A valid search pattern, which is used to locate the position where the new line will be added. Wildcard characters allowed are asterisk (*) (for all characters) and question mark (?) (for one character). This attribute is not valid for remove text_file_objects			
	String expression	No	None	line, command_line

Table 18. SPD file attributes in text file stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
position	The position where the line, command_line, or token is added. For lines and command lines, it is used together with the pattern attribute to establish position. A pattern is needed for relative positions (*after_xxx*, *before_xxx*). If the pattern is not found for **after_first** and **after_last**, a line is added to the end of the file; for **before_first** and **before_last**, a line is added to the beginning of the file. This attribute is not valid for remove text_file_objects.			
	begin, **end**, **after_first**, **before_first**, **after_last**, **before_last** For tokens, only **begin** and **end** are valid.	No	**end**	line, command_line, token
remove_if_modified	Specifies whether to remove the object even if the target object has been modified.			
	y: yes **n**: no	No	**n**	text_file_objects, command_line, token
replace_if_existing	Specifies whether to replace an object that already exists on the target system.			
	y: yes **n**: no	No	**y**	text_file_objects, command_line, token
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	text_file_objects
text	The text or token to be added or removed.			
	String	Yes	None	line, command_line, token
token_separator	The character to be used as token separator.			
	String	No	semicolon (;)	token

SPD File Example: Adding Text File Objects

```
'TIVOLI Software Package v4.3.1 – SPDF'
package
  name                    = "Text_File2"
  version                 = 2.0
  stop_on_failure         = y

  default_variables
    prod_dir              = c:\prod
    sys_dir               = c:\prod\sys
  end

  add_text_file_objects
```

```
# Object characteristics
name                    = "Add items to d:\test.txt"
stop_on_failure         = n

# Text File defaults
file                    = "d:\test.txt"

  add_line
    text                = "REM –line added at the beginning
      of c:\test.txt file"
    position            = begin
  end

  add_line
    text                = "REM –line added at the end of
      c:\test.txt file"
    position            = end
  end

  add_line
    text                = "REM –line added after_first
      set of c:\test.txt file"
    position            = after_first
    pattern             = set*
  end

  add_line
    text                = "REM –line added before last
      path of c:\test.txt file"
    position            = before_first
    pattern             = path*
  end

  add_line
    text                = "REM –line added after last
      REM of c:\test.txt file"
    position            = after_last
    pattern             = REM*
  end

  add_command_line
    text                = "device=$(sys_dir)\begin.sys"
    position            = begin
    command             = "device"
    key                 = "begin.sys"
  end

  add_command_line
    text                = "device=$(sys_dir)\end.sys"
    position            = end
    command             = "device"
    key                 = "end.sys"
  end

  add_command_line
    text                = "device=$(sys_dir) \
      \after_first_pippo.sys"
    position            = after_first
    pattern             = pippo*
    command             = "device"
    key                 = "after_first_pippo.sys"
  end

  add_command_line
    text                ="device=$(sys_dir)
      \before_first_pippo.sys"
```

```
        position           = before_first
        pattern            = pippo*
        command            = "device"
        key                = "before_first_pippo.sys"
      end


      add_command_line
        text               = "device=$(sys_dir)
          \after_last_pluto.sys"

        position           = after_last
        pattern            = pluto*
        command            = "device"
        key                = "after_last_pluto.sys"
      end

      add_token
        text               = "$(prod_dir)\bin"
        position           = begin
        key                = path
      end

      add_token
        text               = "$(prod_dir)\dll"
        position           = end
        key                = path
      end
   end
end
```

The following figure shows the result after this SPD file example is run on a sample machine.

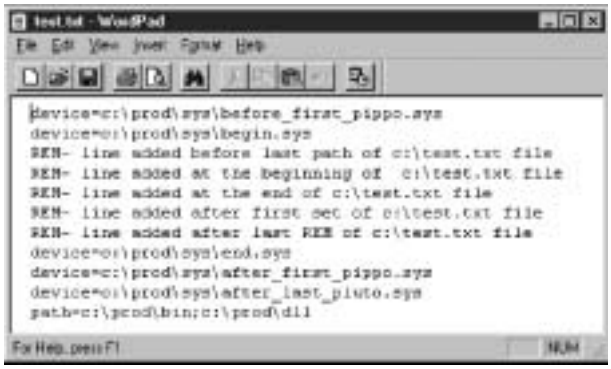


Figure 9. Output of the SPD file example for adding text file objects

SPD File Example: Removing Text File Objects

```
'TIVOLI Software Package v4.3.1 – SPDF'
package
  name                     = "REM_Text_File2"
  version                  = 2.0
  stop_on_failure          = y

  default_variables
    prod_dir               = c:\prod
    sys_dir                = c:\prod\sys
  end

  remove_text_file_objects
```

```
    # Object characteristics
    name                  = "remove items to d:\test.txt"
    stop_on_failure       = n

    # Text File defaults
    file                  = "d:\test.txt"

      add_line
        text              = "REM -line added at the
          beginning of c:\test.txt file"
      end

      add_command_line
        text              = "device=$(sys_dir)\begin.sys"
        command           = "device"
        key               = "begin.sys"
      end

      add_token
        text             = "$(prod_dir)\bin"
        key              = path
      end
   end
end
```

OS/400 Native Objects

Software packages can be built to perform actions on OS/400 systems. These actions include:

- Add or remove an OS/400 library.
- Add or remove individual objects for an OS/400 library.
- Install or remove a licensed program on an OS/400 system.
- Change an OS/400 system value

os400_lib and os400_obj

The following definition shows the attributes that can be defined for an **add_os400_lib** and **add_os400_obj** stanza. For information about these attributes, see Table 19 on page 78.

```
add_os400_lib
     # Inherited from action
      condition               = <string constrain<

      # lib-specific attributes
      add                     = <Boolean>
      location                            = <pathname>
      name                    = <pathname>
      destination             = <pathname>
      descend                     = <Boolean>
      changed_obj_only        = <Boolean>
      reference_date          = <string>
      reference_time          = <string>
      target_release          = <string>
      remote                  = <Boolean>
      replace_option          = <enumerated

     add_os400_obj
       name                    = <pathname>
       destination             = <pathname>
       target_release          = <string>
       remote                  = <Boolean>
       replace_option          = <enumerated
     end
end
```

The following definition shows the attributes that can be defined for **remove_os400_lib** and **remove_os400_obj** stanzas. For information about these attributes, see Table 19 on page 78.

```
remove_os400_lib
     # Inherited from action
      condition               = <string constrain>

      # lib-specific attributes
        remove              = <Boolean>
        descend             = <Boolean>

      # Inherited
        destination          = <pathname>

      # Remove object default attributes
        stop_on_failure      = <Boolean>
        caption              = <string>
        condition            = <string>
        is_per_user          = <Boolean>


     remove_os400_obj
      # Inherited
        destination          = <pathname>

      # Remove object default attributes
        caption              = <string>
        condition            = <string>
        is_per_user          = <Boolean>
        end
end
```

os400_licpgm

The following definition shows the attributes that can be defined for **add_os400_licpgm** and **remove_os400_licpgm** stanzas. For information about these attributes, see Table 19 on page 78.

```
add_os400_licpgm
     add                 = <Boolean>
     licpgm_ID           = <string>
     licpgm_option       = <string>
     language            = <string>
     release             = <string>
     restore_object      = <enumerated>
     remote              = <Boolean>
     device              = <pathname>
     source              = <pathname>
     target_release      = <string>

end
remove_os400_licpgm
     remove              = <Boolean>
     licpgm_ID           = <string>
     licpgm_option       = <string>
     language            = <string>
     release             = <string>

    # Inherited
     destination         = <pathname>

    # Remove object default attributes
     caption             = <string>
     condition           = <string>
     is_per_user         = <Boolean>

end
```

Note: To successfully install a software package that installs a licensed program, some products require that the user profile running the installation process must be included in the system distribution directory. In this case, add the QTIVROOT user profile to the system distribution directory. For more information, refer to the product installation manual.

os400_sysval

The following definition shows the attributes that can be defined in a **os400_sysval** stanza. For information about these attributes, see Table 19.

```
os400_sysval
     # Inherited from action
      condition               = <string constrain>

      sysval_name             = <string>
      value                   = <string>
end
```

Attributes in OS/400 Stanzas

Table 19 shows a list of all the attributes that can be defined in the **os400_lib**, **os400_obj**, **os400_licpgm**, and **os400_sysval** stanzas.

Table 19. SPD file attributes in OS/400 stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
changed_obj_only	If set to **n**, all objects in the specified library are to be included. If set to **y** only those objects that have changed since the reference date and time are to be included. This attribute is only valid if the **descend** attribute is set to **y**.			
	y: yes **n**: no	No	**n**: no	os400_lib
descend	If set to **n**, only the library is included in the package and you can specify individual objects by embedding os400_obj stanzas within the os400_lib stanza. If set to **y**, the library and its contents are included in the package. The contents of the library that are included when **descend** is set to **y** can be limited by setting the **changed_obj_only** attribute to **y**.			
	y: yes **n**: no	No	**n**: **no**	remove_os400_lib
destination	Destination path of the os400_lib or os400_obj object.			
	String	No	value of the **location** attribute	remove_os400_lib, remove_os400_obj remove_os400_lic pgm
device	If the **remote** attribute is set to **y**, the remote location of the program must be indicated. This can either be a device, such as a cdrom, which you specify using this attribute, or a save file, which you specify using the **source** attribute. You cannot specify both a **device** and a **source** attribute.			
	String	No	None	os400_licpgm
language	Identifies the language option to be used when installing the licensed program.			
	String	No	***PRIMARY**	remove_os400_lic pgm

Table 19. SPD file attributes in OS/400 stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
licpgm_ID	Alphanumeric code that identifies the product to be installed or removed.			
	String	Yes	None	remove_os400_licpgm
licpgm_option	Identifies the optional parts of the specified licensed program which are to be installed.			
	String	No	***BASE**	remove_os400_licpgm
location	Source path of the os400_lib or os400_obj object.			
	String	Yes	None	add_os400_lib
reference_date	If **changed_obj_only** is set to **y**, only objects that have changed since the date and time specified in this attribute and the **reference_time** attribute are included.			
	String	If **changed_objects** is **y**.	None	add_os400_lib
reference_time	If **changed_obj_only** is set to **y**, only objects that have changed since the date and time specified in this attribute and the **reference_date** attribute are included.			
	String	If **changed_objects** is **y**.	None	add_os400_lib
release	Specifies the release of the program that is to be installed or removed.			
	String	No	None	add_os400_licpgm remove_os400_licpgm
remote	In os400_lib and os400_obj stanzas, if this attribute is set to **FALSE**, the object is to be collected on the source system. If set to **TRUE** the object is to be collected on the target system. In os400_licpgm stanzas, if this attribute is set to **TRUE**, the program is not included in the package at build time. Instead it is retrieved at install time from a specified device or savf.			
	y: yes **n**: no	No	**n**: no	add_os400_lib, add_os400_obj, add_os400_licpgm
replace_option	This specifies which objects are to be written to the library on the target system, as follows: ***ALL** All files from the source library are written to the target library. ***NEW** Only files that exist in the source library but do not exist in the target library are written. ***OLD** Existing files in the target library are overwritten with versions from the source library.			
	***ALL** ***NEW** ***OLD**	No	***ALL**	add_os400_lib, add_os400_obj

Table 19. SPD file attributes in OS/400 stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
restore_object	Species the type of installation or upgrade to be made. Possible values are: ***ALL** Install the licensed program and the language component. ***PGM** Install the licensed program only. ***LNG** Install the language component only.			
	***ALL** ***PGM** ***LNG**	No	***ALL**	add_os400_licpgm
source	If the **remote** attribute is set to **y** the remote location of the program must be indicated. This can either be a device, such as a cdrom, which you specify using the **device** attribute, or a save file, which you specify using this attribute. You cannot specify both a **device** and a **source** attribute.			
	String	No	None	os400_licpgm
sysval_name	Identifies the system value that is to be changed.			
	String	Yes	None	os400_sysval
target_release	Specifying an operating system release here makes the addition of the object dependent on the operating system on the target being the same release.			
	String	No	None	add_os400_lib, add_os400_obj, add_os400_licpgm
value	Specifies the new value to be assigned to a system value.			
	String	Yes	None	os400_sysval

SPD File Examples: OS/400 Objects

The following example defines an action to add the library /QSYS.LIB/ABD.LIB on the target system. The library and all the files in it are collected from the preparation system, as defined by the settings of the **descend** and **changed_obj_only** attributes. All the files collected from the preparation system library are written to the target system, even if they already exist. This is controlled by the setting of the **replace_option** attribute.

```
add_os400_lib
   location = "\QSYS.LIB"
   name = "ABD.LIB"
   destination = "\QSYS.LIB\ABD.LIB"
   descend = y
   changed_obj_only = n
   target_release = "*CURRENT"
   remote = n
   replace_option = "*ALL"
 end
```

The following example defines an action the add a file from a library on the preparation system to a library on the target system. The file, MYOBJ.FILE, is retrieved from the library /QSYS.LIB/MYLIB.LIB and written to the library /QSYS.LIB/NEWLIB.LIB as NEWOBJ.FILE.

```
add_os400_lib
  stop_on_failure = y
  location = "\QSYS.LIB"
  name = "MYLIB.LIB"
  destination = "\QSYS.LIB\NEWLIB.LIB"
  descend = n
  remote = n
  add_os400_obj
    name = "MYOBJ.FILE"
    destination = "NEWOBJ.FILE"
    target_release = "*CURRENT"
    remote = n
    replace_option = "*ALL"
  end
end
```

The following example defines an action to install the BASE part of release V4R3M0 of the licensed program with the ID 5799XEH. The program is retrieved from the preparation system and added to a save file at build time. Only the primary language is installed.

```
add_os400_licpgm
  licpgm_ID = "5799XEH"
  licpgm_option = "*BASE"
  language = "*PRIMARY"
  release = "V4R3M0"
  restore_object = "*ALL"
  target_release = "*CURRENT"
  remote = n
end
```

The following example defines an action to install the same licensed program. In this case, the licensed program is retrieved from a save file that is already on the target system. For this action, there would be no *SAVF containing the licensed program included in the software package.

```
add_os400_licpgm
  licpgm_ID = "5799XEH"
  licpgm_option = "*BASE"
  language = "*PRIMARY"
  release = "V4R3M0"
  restore_object = "*ALL"
  target_release = "*CURRENT"
  remote = y
  source = "\QSYS.LIB\QGPL.LIB\SAVEFILE.FILE"
end
```

The following example defines an action to change the value of the OS/400 system value QUSRLIBL by adding the new libraries MYLIB and MYLIB2.

```
os400_sysval
  sysval_name = "QUSRLIBL"
  value = "MYLIB MYLIB2"
end
```

The contained_signature Stanza

During the preparation of a software package it is possible to specify a file object within the package to be used a signature by setting the is_signature attribute. The following definition shows the attributes that can be defined in a contained_signature stanza. For more information about this attribute, see Table 11 on page 36.

```
contained_signature
            file_name           = <string>
            file_size           = <integer>
```

```
            add_if_non_existing  = <Boolean>
            inventory_description = <string>
            inventory_version     = <Boolean>
 end
```

If the software package contains files to be installed with a local installer, such as MSI, InstallShield, and so on, it is not possible to assign a signature property. The contained_signature stanza allows you to define the signatures in the software package by name and size. Table 20 provides details of the attributes of the contained_signature stanza.

Table 20. SPD file attributes in contained_signature stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
file_name	File name of the product or software to be installed, for example, notes.exe. In a PC environment, the signature file name must be in upper case. In a UNIX environment, the signature file name must be in the same case as the file name.			
	String	Yes	None	contained_signature
file_size	Size of the file to be installed.			
	Integer	Yes	None	contained_signature
add_if_not_existing	If this signature file is not already recorded in the Inventory database, use this attribute to add the signature to appropriate table.			
	y: yes **n**: no	No	**n**	contained_signature
inventory_description	Description of the software to be installed, for example, Lotus Notes			
	String	Yes, if add_if_not _existing is set to **y**	None	contained_signature
inventory_version	Version number of the software to be installed.			
	Integer	Yes, if add_if_not _existing is set to **y**	None	contained_signature

SPD file example: contained_signature

```
contained_signature
               file_name            = mysign
               file_size            =  2000
               add_if_non_existing  = y
               inventory_description = "Test signature"
               inventory_version     = 1
 end
```

Device-related Actions

Device stanzas include sections and keyword-value pairs for devices. The SPD file can include one or more device stanzas for each device type. However, the same SPD file cannot contain actions for both devices and for endpoints.

Supported device types are as follows:
- PalmOS
- WinCE
- Nokia 9300
- Nokia 9500
- Nokia_s60

Depending on your target device, you can perform different actions. The following operations can be performed on PalmOS and WinCE devices:
- Add a directory
- Execute a program
- Configure device settings
- Check disk space
- Remove an installed software package

For more information on these actions, see “The device_objects stanza for WinCE devices” on page 87 and “The device_objects stanza for PalmOS devices” on page 91.

On Nokia devices, you can define the configuration parameters using the device_action stanza. For more information, see “The device_action stanza for Nokia devices.”

The device_action stanza uses device management technology, with which service providers and users can easily configure and maintain different applications on Nokia devices. This technology is based on a secure relationship between the device and the Tivoli Web Gateway (TWG).

Device provisioning or initialization provides the pervasive device with the following features:
- A secure relationship with the Tivoli Web Gateway
- Key parameters for the initial connection to the Tivoli Web Gateway
- Repair of the initial configuration if it becomes damaged or corrupted

If you plan to create reference models or activity plans based on the results of packages distributed to devices, these packages must contain only one action. If this is not the case, you can create packages containing as many actions as necessary.

The device_action stanza for Nokia devices

The following definition shows the attributes that can be defined in a device_action stanza. To perform several actions on a Nokia device, add a series of device_action stanzas. For information about these attributes, see Table 21 on page 84.

```
device_action
 caption = <string>
 device_type = <Nokia9300, Nokia9500, Nokia_s60>
 action_type = <device_configuration, application_distribution, device_provisioning,
 notification, TARM_config, reboot, process_listing,
 process_stop,  wipe, generic,>

 # Sequence of <action_parameter stanzas>
end
```

Note: Due to Nokia device limitations, the `process_listing` and `process_stop` actions do not work properly.

action_parameter

The following definition shows the attributes that can be defined in an action_parameter stanza. For information about these attributes, see Table 21.

```
action_parameter
     key   = <SourceFilePath, SendNotifcation, ProcessID>
     value = <string>
end
```

Attributes in the device_action Stanza for Nokia devices

Table 21 shows a list of the attributes that can be defined in the device_action and action_parameter stanzas.

Table 21. SPD file attributes in the device_action and action_parameter stanzas

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
caption	A unique identifier to define any component in the package.			
	String	No	None	device_action
device_type	Specifies the device type to which the action is targeted.			
	• Nokia9300 • Nokia9500 • Nokia_s60	Yes	None	device_action
action_type	Species the action to be performed on the specified device.			
	• device_ configuration • application_ distribution • device_ provisioning • notification • TARM_config • reboot • process_listing (*) • process_stop (*) • wipe • generic	Yes	None	device_action
key	Specifies the parameters for the device action to be performed.			
	• SourceFilePath • Send Notification • ProcessID	Yes	None	action_parameter
value	Specifies a value to be used for the **key** attribute. The value can be either the path to a file, when the **key** attribute is set to **SourceFilePath**, or can be either **true** or **false** when the **key** attribute is set to **SendNotification**.			
	String	Yes	None	action_parameter

(*): Due to Nokia device limitations, the `process_listing` and `process_stop` actions do not work properly.

The following table shows a list of the parameters that can be defined in the different device actions:

Table 22. SPD file parameters in the device actions

	Parameter		
Action	**SendNotification**	**SourceFilePath**	**ProcessID**
device_ configuration	X	X	
application_ distribution	X	X	
device_ provisioning		X	
notification			
TARM_config	X	X	
reboot	X		
process_listing	X		
process_stop	X		X
wipe	X		
generic	X	X	

The following table shows a definition of the parameters supported by the device actions:

Table 23.

Parameter	**Definition**
SendNotification	A boolean value.
SourceFilePath	The destination path of a source file.
ProcessID	A string of alpha-numeric characters.

SPD File Example: Performing Two Device Actions for Nokia 9500 and One Device Action for Nokia 9300

```
"TIVOLI Software Package v4.3.1 - SPDF"

package
      name = MultiActions
      title = "No title"
      version = 1
      web_view_mode = hidden
       undoable = o
      committable = o
      history_reset = n
      save_default_variables = n
      creation_time = "2004-11-05 14:23:15"
      last_modification_time = "2004-11-05 14:23:51"
      source_host_name = LAB132042
      move_removing_host = y
      no_check_source_host = y
       lenient_distribution = n
      default_operation = install
      server_mode = all
      operation_mode = not_transactional
      log_path = C:\PROGRA~1\Tivoli\bin\swdis\work\MultiActions.1.log
      post_notice = n
      before_as_uid = 0
      skip_non_zero = n
       after_as_uid = 0
```

```
        no_chk_on_rm = y
        log_host_name = LAB132042
       versioning_type = swd
        package_type = refresh
        sharing_control = none
        stop_on_failure = y

        device_action
          caption = SametimeInstNokia9500
           device_type = Nokia9500
          action_type = application_distribution

          action_parameter
                 key = SourceFilePath
               value = C:\Nokia\sis\sametime9500.sis
          end

    end
       device_action
         caption = SametimeConf
          device_type = Nokia9500
         action_type = device_configuration

         action_parameter
                key = SourceFilePath
              value = C:\PROGRA~1\Tivoli\bin\w32-ix86\speditor\
             classes\config\Sametime.properties
         end

    end

       device_action
         caption = SametimeInstNokia9300
          device_type = Nokia9300
         action_type = application_distribution

         action_parameter
                key = SourceFilePath
              value = C:\Nokia\sis\sametime9300.sis
         end

    end
end
```

SPD File Example: Performing One Device Action for Nokia s60

```
"TIVOLI Software Package v4.3.1 - SPDF"

package
      name = SingleAction
      title = "No title"
      version = 1
      web_view_mode = hidden
       undoable = o
      committable = o
      history_reset = n
      save_default_variables = n
      creation_time = "2004-11-05 14:23:15"
      last_modification_time = "2004-11-05 14:23:51"
      source_host_name = LAB132042
      move_removing_host = y
      no_check_source_host = y
       lenient_distribution = n
      default_operation = install
      server_mode = all
```

```
      operation_mode = not_transactional
      log_path = C:\PROGRA~1\Tivoli\bin\swdis\work\SingleAction.1.log
      post_notice = n
      before_as_uid = 0
      skip_non_zero = n
       after_as_uid = 0
       no_chk_on_rm = y
       log_host_name = LAB132042
      versioning_type = swd
       package_type = refresh
       sharing_control = none
       stop_on_failure = y

       device_action
         caption = policy_configuration
          device_type = Nokia_s60
         action_type = TARM_config

         action_parameter
                key = SourceFilePath
              value = C:\temp\policy.xml
         end

           end
```

The device_objects stanza for WinCE devices

The following definition shows the attributes that can be defined in a device_objects stanza. For information about these attributes, see Table 24 on page 89.

```
device_objects
       # Device Objects defaults
   # Action name
          name = <device_object_name>
   # The device subtype (WinCE)
          device_type = <device_subtype>

   # Inherited from container
          stop_on_failure = y

   # Sequence of <add_device_file>
   # Sequence of <add_device_directory>
   # Sequence of <device_execute_program>
   # Sequence of <device_configuration_settings>

end
```

add_device_file

The following definition shows the attributes that can be defined in an add_device_file stanza. For information about these attributes, see Table 24 on page 89.

```
add_device_file
   # Inherited from container
              stop_on_failure = <Boolean>
   # The source path and filename
              name = <path>\<filename>
   # The destination file name
              destination = <filename>
   # Space file occupies on device
       need_space = <disk_space>
end
```

add_device_directory

The following definition shows the attributes that can be defined in an add_device_directory stanza. For information about these attributes, see Table 24 on page 89.

```
add_device_directory
  # Inherited from container
              stop_on_failure = <Boolean>
 # The source drive
              location = <drive>

   # The source directory
              name = <directory_name>

   # The full destination path
              destination = <pathname>

   # Specific attributes
              descend_dirs = <Boolean>
              # Sequence of <add_device_file>
        end
```

device_execute_program

The following definition shows the attributes that can be defined in a device_execute_program stanza. The program to be run must already be stored on the device or must have been previously distributed with a software package. For information about these attributes, see Table 24 on page 89.

```
device_execute_program

   # Inherited from container
           device_execute_program
    caption = <string>
    path = <pathname>
    arguments = <string>
end
```

device_configuration_settings

The following definition shows the attributes that can be defined in a device_configuration_settings stanza. For information about these attributes, see Table 24 on page 89.

```
device_configuration_settings

   # Inherited from container
              stop_on_failure = <Boolean>
   # Inherited from container
             caption = <string>

   # Sequence of <device_item>
end
```

device_item: The following definition shows the attributes that can be defined in a device_item stanza. For information about these attributes, see Table 24 on page 89.

```
device_item
    # Keyword setting
                    key = <keyword>

    # The value
                    value = <setting>
                end
```

Attributes in the device_objects stanza for WinCE devices

Table 24 shows a list of the options that are valid and of all the attributes that can be defined in the device_objects stanza and the stanzas it includes.

Table 24. SPD file attributes in device objects stanzas

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
arguments	Specifies the arguments to be included to run the program.			
	String	No	None	device_execute_program
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	device_configuration_ settings
descend_dirs	Determines whether the add command relates to the entire directory tree, **y**, or only to the top-level directory, **n**. For example, if for the directory \Program Files\Tivoli\Desktop, you specify **n**, only the files that are in the folder Program Files are added. If you specify **y**, all the subfolders and their contents are added.			
	y: yes **n**: no	No	**n**	add_device_directory
destination	The path or filename of the device where the object is to be added. Do *not* specify a drive.			
	String	No	None	add_device_file, add_device_directory
device_type	Determines on which type of devices the actions in the device_objects stanza are performed.			
	WinCE	Yes	None	device_objects
key	The keyword for a device configuration setting during customization. Refer to the Resource Manager part of the *User's Guide for Deployment Services* for a list of keywords for each device type.			
	String	No	None	device_item
location	Drive of the directory to add to the device.			
	String	Yes	None	add_device_directory
name	A unique identifier.			
	String	Yes	None	device_objects, add_device_file, add_device_directory device_execute_program
need_space	Space file or program required on device.			
	Valid units can be: **b**: bytes **k**: kilobytes **m**: megabytes	No	None	add_device_file
path	Specifies the path on the device where the program to be executed is located. Do not specify a drive.			
	String	No	None	device_execute_program

Table 24. SPD file attributes in device objects stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	device_objects, add_device_file, add_device_directory, device_configuration_ settings, device_execute_program
value	Value for the keyword for a device configuration setting during customization. Refer to the Resource Manager part of the *User's Guide for Deployment Services* for a list of keywords for each device type.			
	String	No	None	device_item

SPD File Example: device_objects for WinCE devices

```
        device_objects
                name = WinCEObj
                device_type = WinCE
                stop_on_failure = y

                add_device_directory
                        location = C:\
                        name = bin
                        destination = C:\bin
                        descend_dirs = n

                        add_device_file
                                name = basename.exe
                                destination = basename.exe
                                need_space = 0
                        end
                 end

                device_execute_program
                       path = \snake.exe
                       arguments = -p
                       need_space = 33k
                end

                device_configuration_settings
                   stop_on_failure = y
                        device_item
                              key = net.dns1
                             value = 255.233.122.110
                        end
                end

        add_device_directory
                location = C:\
                name = CMVC
                destination = \CMVC
                descend_dirs = n
        end
end
```

The device_objects stanza for PalmOS devices

The following definition shows the attributes that can be defined in a device_objects stanza. For information about these attributes, see Table 24 on page 89.

```
device_objects
       # Device Objects defaults
   # Action name
           name = <device_object_name>
   # The device subtype (Palm)
           device_type = <device_subtype>

   # Inherited from container
           stop_on_failure = y

   # Sequence of <add_device_file>
   # Sequence of <device_execute_program>
   # Sequence of <device_configuration_settings>

end
```

add_device_file

The following definition shows the attributes that can be defined in an add_device_file stanza. For information about these attributes, see Table 24 on page 89.

```
add_device_file
   # Inherited from container
               stop_on_failure = <Boolean>
# The source path and filename
               name = <path>\<filename>
   # The destination file name
               destination = <filename>
   # Space file occupies on device
   need_space = <disk_space>
        end
```

device_execute_palm_program

The following definition shows the attributes that can be defined in a device_execute_palm_program stanza. The program to be run must be sent with the software package as a corequisite file. For information about these attributes, see Table 24 on page 89.

```
device_execute_palm_program

   # Inherited from container
        device_execute_program
    caption = <string>
    name = <string>
    type = <Database, Application>
    launch_code = <Integer>
    end
```

device_configuration_settings

The following definition shows the attributes that can be defined in a device_configuration_settings stanza. For information about these attributes, see Table 24 on page 89.

```
device_configuration_settings

   # Inherited from container
               stop_on_failure = <Boolean>
    # Inherited from container
```

```
            caption = <string>

   # Sequence of <device_item>
             end
```

device_item: The following definition shows the attributes that can be defined in a device_item stanza. For information about these attributes, see Table 24 on page 89.

```
device_item
    # Keyword setting
                    key = <keyword>

    # The value
                    value = <setting>
end
```

Attributes in the device_objects Stanza for PalmOS devices

Table 24 on page 89 shows a list of the options that are valid and of all the attributes that can be defined in the device_objects stanza and the stanzas it includes.

Table 25. SPD file attributes in device objects stanzas for PalmOS devices

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	device_configuration_ settings
destination	The path or filename of the device where the object is to be added.			
	String	No	None	add_device_file device_execute_ palm_program
device_type	Determines on which type of devices the actions in the device_objects stanza are performed.			
	Palm	Yes	None	device_objects
key	The keyword for a device configuration setting during customization. Refer to the Resource Manager part of the *User's Guide for Deployment Services* for a list of keywords for each device type.			
	String	No	None	device_item
launch_code	Specifies the launch code for the application or database specified in the **type** attribute.			
	Integer	No	0	device_execute_ palm_program
name	A unique identifier. For the add_device_file stanza, specify the full path and filename.			
	String	Yes	None	device_objects, add_device_file, device_execute_ palm_program

Table 25. SPD file attributes in device objects stanzas for PalmOS devices (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
need_space	Space file or program required on device.			
	Valid units can be: **b**: bytes **k**: kilobytes **m**: megabytes	No	None	add_device_file device_execute_ palm_program
stop_on_failure	Specifies whether to stop execution if the action fails or the condition is not met.			
	y: yes **n**: no	No	**y**	device_objects, add_device_file, device_configuration_ settings, device_execute_ palm_program
type	Specifies whether the program is to be run on an application or on a database.			
	Database **Application**	No	Application	device_execute_ palm_program
value	Value for the keyword for a device configuration setting during customization. Refer to the Resource Manager part of the *User's Guide for Deployment Services* for a list of keywords for each device type.			
	String	No	None	device_item

SPD File Example: device_objects for PalmOS devices

```
device_objects
        caption = "execute a Palm application"
        device_type = Palm
        stop_on_failure = y

                add_device_file
                        name = C:\bin\sel.prc
                        destination = sel.prc
                        need_space = 0
                end


                  device_configuration_settings
                        stop_on_failure = y
                             device_item
                                  key = Preset
                                  value = "3"
                             end
                  end


             device_execute_palm_program
                  caption = Backgammon
                 name = D:\src\backgammon.prc
                  destination = backgammon.prc
                  need_space = 45000b
                  type = Application
```

```
                launch_code = 0

            end
end
```

Program Actions

In addition to the actions of adding and removing various types of objects, other actions are available. These are decried in the following sections:

install_msi_product and install_msi_patch

The install_msi_product and install_msi_patch actions enable you to distribute and install software products and patches using packages containing native Microsoft Software Installer (MSI) actions.

The SPD file stanzas include the following types of information:

- Identification information required for the installation such as the name of the package file and the location of product image files to be used in the installation.
- Log options, including the level of logging, the location of the log file and whether the log must be reported to the Software Distribution server.
- Parameters that determine the method of installation. For example:
  - Whether the product images are included in the package (bundled installation) or located on a remote server (redirected installation).
  - Whether the installation is to be silent or have some level of interaction.
  - The conditions for reinstalling files, for example, if the file is missing.
- Options for reinstalling the product when the package is executed with the **repair** installation option.
- Command line options for the Windows installer. Each option consists of a Property = Value pair.
- Entries for program features (install_msi_product stanza only).

  Each entry defined in the file includes the name of the feature and an action that can be taken, for example, install it on the local machine.

  Note: The MSI package includes a default set of features to be installed. You only need to create entries for features if you do not want to use the defaults.

Table 26 on page 95 provides details of the attributes of the two stanzas.

The following is the format of the install_msi_product stanza:

```
install_msi_product
      image_dir            = <pathname>
      package_file         = <pathname>
      source_dir           = <pathname>
      is_image_remote      = <Boolean>
      keep_images          = <Boolean>
      compression_method   = <deflated, stored>
      all_users            = <Boolean>
      destination_folder   = <pathname>
```

```
      reinstall_mode       = <enumerated >
      log_mode             = <enumerated >
      log_path             = <pathname>
      report_log           = <Boolean>
      ui_level             = <enumerated>
      properties           = <key-value list>
      SequenceOf<feature>
end
```

The elements specified in the sequence must be features of the MSI product. Each feature stanza must be specified in the following way:

```
feature
      name                   =<string>
      action                =<enumerated>
end
```

The following is the format of the install_msi_patch stanza:

```
install_msi_patch
      image_dir            = <pathname>
      package_file         = <pathname>
      source_dir           = <pathname>
      is_image_remote      = <Boolean>
      compression_method   = <deflated, stored>
      reinstall_mode       = <enumerated >
      log_mode             = <enumerated >
      log_path             = <pathname>
      report_log           = <Boolean>
      ui_level             = <enumerated>
      properties           = <key-value list>
      end
```

Attributes in MSI File Stanzas

Table 26 shows all the attributes that can be defined in the install_msi_package and install_msi_patch stanzas.

Table 26. SPD file attributes in Install MSI stanzas

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
action	Specifies an action to be carried out on the MSI product feature with which it is defined.			
	InstallLocal InstallWhen Required Remove	No	InstallLocal	feature
all_users	Indicates whether the MSI product is to be available for all users of the target system or only for the user logged on at the time of installation.			
	Y N	No	**Y**	install_msi_product
compression_ method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	install_msi_product, install_msi_patch
destination_folder	Specifies the value of the INSTALLDIR property.			
	String	Conditional		install_msi_product

Table 26. SPD file attributes in Install MSI stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
features	A list of features that can be installed or removed. Each feature has the attribute name and action.			
	Sequence of <feature>	No	All defaults for the product	install_msi_product
image_dir	Full pathname of the directory on the target system that contains the product images. For bundled installations this is a local path; for redirected installations it is a network path. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	string	Yes		install_msi_product install_msi_patch
is_image_remote	If set to **y**, specifies a redirected installation, where image files are obtained from a directory on a remote server, at installation time.			
	y: yes **n**: no	No	**n**	install_msi_product install_msi_patch
keep_images	Specifies whether or not product images are to be stored on the target system following installation. This parameter is relevant only for bundled installations.			
	y: yes **n**: no	No	**n**	install_msi_product
log_mode	Specifies the level of logging to be used during the installation. More than one value can be specified.			
	Disabled, FatalExit, Error, Warning, User, Info, ResolveSource, OutOfDiskSpace, ActionStart, ActionData, CommonData, PropertyDump, Verbose	No	Error FatalExit	install_msi_product install_msi_patch
log_path	Pathname of the directory where the log is to be saved. This value is required unless the log_mode attribute is set to **Disabled**.			
	String	If log_mode is not **Disabled**		install_msi_product install_msi_patch
name	The name of a feature of the MSI product to be installed or removed.			
	String	Yes		feature

Table 26. SPD file attributes in Install MSI stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
package_file	Name of the MSI file that contains the package to be installed. This file is assumed to be in the directory identified in the image_dir attribute.			
	String	Yes		install_msi_product install_msi_patch
properties	Each entry defines a property type and its value. The entire definition of this attribute must be enclosed between double quotation marks. If the value includes any spaces, it must be enclosed between double quotation marks.			
	Vector<Pair>	No	Empty list	install_msi_product install_msi_patch
reinstall_mode	Specifies the types of components to be reinstalled when the package is executed in **repair** mode. More than one mode can be specified.			
	FileMissing, FileOlderVersion, FileEqualVersion, FileExact, FileVerify, FileReplace, UserData, MachineData, Shortcut Package	No	FileEqual Version UserData Machine Data Shortcut	install_msi_product install_msi_patch
report_log	Indicates whether the log is to be reported back to the Software Distribution server.			
	y: yes **n**: no	No	**n**	install_msi_product install_msi_patch
source_dir	Pathname for the directory on the source host machine that contains the product images. This path must only be specified for a bundled installation. In this case, the image files are distributed with the package and downloaded to the directory identified in the **image_dir** attribute. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	String	Conditional		install_msi_product install_msi_patch
ui_level	Specifies the level of interface to be used during disconnected installation processes. For server installations this attribute is ignored and the installation is silent.			
	Full, Reduced, Basic, UIDefault, None, EndDialog, ProgressOnly	No	None	install_msi_product install_msi_patch

Note: For more information about the values allowed for the log_mode, reinstall_mode, and ui_level attributes, see the Microsoft Software Installer documentation.

SPD File Example: install_msi_product

The following section shows an example of software package definition file containing an install_msi_product stanza:

```
install_msi_product
    condition = '$(operation) == INSTALL'
    image_dir        = C:\tmp\Repos\Editors
    package_file     = Editors.msi
    is_image_remote  = y
    compression_method   = deflated
    destination_folder = "$(program_files)/My Folder"
    log_mode         = "fatal_exit,error, warning
    log_path         = $(product_dir)/msi/logs/editors.log
    report_log       = y
    reinstall_mode   = file_missing,file_verify,file_replace
    ui_level         =none
       feature
          name = Wordpad
          action = install_local
       end
       feature
          name = Notepad
          action = install_local
    end
```

install_solaris_package and install_solaris_patch

The install_solaris_package and install_solaris_patch actions enable you to distribute and install software products and patches using packages containing native Solaris actions.

The SPD file stanzas include the following types of information:

- Identification information required for the installation such as the name of the package file and the location of product image files used in the installation.
- The standard output and the standard error of the operation you performed are optionally reported to the server.
- Parameters that determine the method of installation. For example:
  - Whether the product images are included in the package (bundled installation) or located on a remote server (redirected installation).
  - Whether the installation is to be silent or have some level of interaction.
- Entries for package instances.

  Each entry defined in the file includes the name of the package instance for which you want to take action, for example, to install it on the local machine.

Table 29 on page 100 provides details of the attributes of the two stanzas.

The following is the format of the install_solaris_package stanza:

```
install_solaris_package
  source_dir                    <pathname>
  image_dir                     <pathname>
  keep_images                   <bool>
  is_image_remote               <bool>
  administration_file           <pathname>
  package_file                  <string>
  spool_directory               <pathname>
  client_root_path              <pathname>
  use_root_path                 <bool>
  interactive                   <bool>
```

```
  report_output_to_server          <bool>
  response_file                    <pathname>
  remove_absolutely                <bool>
  fs_file                          <pathname>
  compression_method               <deflated, stored>
  package_instance
    name                <ustring>
    description         <ustring>
    platform            <ustring>
    version             <ustring>
    revision            <ustring>
  end
end
```

Some Software Distribution operations are limited or not available in the install_solaris_package stanza. Table 27 describes the mappings between Software Distribution and Solaris package operations:

Table 27. Supported Software Distribution Operations in install_solaris_package stanza

Software Distribution Operations	PKG Command Options
Install	pkgadd
Install transactional	pkgadd -s *spool_dir*
Commit	pkgadd -d *spool_dir*
Verify	pkgadd -v [other options] package instance
Remove	pkgrm -s package instance
Undo (rollback of an IP--- package)	pkgrm -s *spool_dir*

The following is the format of the **install_solaris_patch** stanza:

```
install_solaris_patch
                        source_dir               <pathname>
                        image_dir                <pathname>
                        keep_images              <bool>
                        is_image_remote          <bool>
                        install_unconditionally  <bool>
                        client_root_path         <pathname>
                        backout_dir              <pathname>
                        service                  <ustring>
                        report_output_to_server  <bool>
                        patch_id                 <ustring>
                        net_install_image        <pathname>
                        force                    <bool>
                        compression_method       <deflated, stored>
end
```

Some Software Distribution operations are limited or not available in the **install_solaris_patch** stanza. Table 28 describes the mappings between Software Distribution and Solaris patch operations:

Table 28. Supported Software Distribution Operations in install_solaris_patch stanza

Software Distribution Operations	PKG Command Options
Install	patchadd -d
Install undoable	patchadd
Undo	patchrm patchid

Attributes in Solaris File Stanzas

Table 29 shows a list of all the attributes that can be defined in the install_solaris_package and install_solaris_patch stanzas.

Table 29. SPD file attributes in Install Solaris stanzas

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
administration_file	Specifies the fully qualified path name for the installation administration file to be used in place of the default installation administration file. This corresponds to the -a option of the **pkgadd** and **pkgrm** Solaris commands.			
	String	No	None	install_solaris_ package
backout_dir	Specifies the name of the backout directory where you want to save the files related to the patch you are installing. If you specify the backout directory the patch files are not saved in the database. This corresponds to the -B option of the **patchadd** and **patchrm** Solaris commands.			
	String	No	None	install_solaris_ patch
client_root_path	Specifies the full path name of the directory to be used as the root path location All files, including package system information files, are relocated to a directory tree starting in the specified root path. The root path may be specified when installing to a client from a server (for example, /export/root/client1).			
	String	No	None	install_solaris_ package, install_solaris_ patch
compression_ method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	install_solaris_ package, install_solaris_ patch
description	Description of the package instance.			
	String	No	None	install_solaris_ package
force	Forces the patch removal regardless of whether the patch was superseded by another patch. This corresponds to the -f option of the **patchrm** Solaris command.			
	y: yes **n**: no	No	**n**	install_solaris_ patch
fs_file	Specifies an alternative FS file to map the file systems of the client. For example, you can use it in situations where the $root_path/etc/vfstab file is non-existent or unreliable. This corresponds to the -V option of the **pkgrm** Solaris command			
	String	No	None	install_solaris_ package

Table 29. SPD file attributes in Install Solaris stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
image_dir	Full pathname of the directory on the target system that contains the product images. For bundled installations this is a local path; for redirected installations it is a network path. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	string	Yes	None	install_solaris_ package, install_solaris_ patch
is_image_remote	If set to **y**, specifies a redirected installation, where image files are obtained from a directory on a remote server, at installation time.			
	y: yes **n**: no	No	**n**	install_solaris_ package, install_solaris_ patch
keep_images	Specifies whether or not product images are to be stored on the target system following installation. This parameter is only relevant for bundled installations.			
	y: yes **n**: no	No	**n**	install_solaris_ package, install_solaris_ patch
install_ unconditionally	Performs the installation even if some target files have changed.			
	y: yes **n**: no	No	**n**	install_solaris_ patch
interactive	Specifies whether the installation requires user intervention.			
	y: yes **n**: no	No	**n**	install_solaris_ package
name	Name of the package instance			
	String	Yes	None	install_solaris_ package, install_solaris_ patch
net_install_image	Absolute path name (net_install_image) to a boot directory where the patch files are located. This corresponds to the -c option of the **patchadd** and **patchrm** Solaris commands.			
	String	No	None	install_solaris_ patch
package_file	Filename of the Solaris package to be installed, if necessary.			
	String	No	None	install_solaris_ package
package_instance	Vector of package instances to be installed by the current package. The package instance is a specific class.			
	Vector	Yes	None	install_solaris_ package

Table 29. SPD file attributes in Install Solaris stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
patch_id	Identifier of the Solaris patch. This corresponds to the -M option of the **patchadd** Solaris command.			
	String	Yes	None	install_solaris_ patch
platform	Name of the platform for the workstation on which you have to install the package.			
	y: yes **n**: no	No	**n**	install_solaris_ package, install_solaris_ patch
remove_absolutely	Remove the package even if a file of the package is shared with another application. This corresponds to the -A option of the **pkgrm** Solaris command.			
	y: yes **n**: no	No	**n**	install_solaris_ package
response_file	Fully qualified path name of the file to be used to perform the installation. This file supplies the interaction responses that are requested by the package in interactive mode. This corresponds to the -r option of the **pkgadd** Solaris command			
	String	No	None	install_solaris_ package
report_output_to_ server	Reports to the server the standard output and the standard error of the operation you performed.			
	Boolean	No	None	install_solaris_ package, install_solaris_ patch
revision	Specifies the revision number of the instance			
	Integer	No	None	install_solaris_ package
service	Specifies an alternate service where you want to store the patch. This service is part of the server and client model, and can be used only from the server's console. Servers can contain shared /usr file systems that are created by a host manager. These service areas can then be made available to the clients they serve. This corresponds to the -S option of the **patchadd** and **patchrm** Solaris commands.			
	String	No	None	install_solaris_ patch
spool_directory	Name of the directory in which to store the package instead of installing it. This parameter is used when you perform an install transactional operation. This corresponds to the -s option of the **pkgadd** Solaris command			
	String	No	var/spool/ pkg	install_solaris_ package

Table 29. SPD file attributes in Install Solaris stanzas (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
source_dir	Pathname for the directory on the source host machine that contains the product images. This path must only be specified for a bundled installation. In this case, the image files are distributed with the package and downloaded to the directory identified in the image_dir attribute. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	String	Conditional		install_solaris_ package, install_solaris_ patch
use_root_path	Specifies whether the root file system of the client is used.			
	y: yes **n**: no	No	**n**	install_solaris_ package
version	Specifies the version of the instance.			
	Integer	No	**n**	install_solaris_ package

SPD File Example: install_solaris_package

The following section shows an example of software package definition file containing an install_solaris_package stanza:

```
install_solaris_package
                              image_dir = /space/tmp/pkgadd/netscape
                        source_dir = /tivoli/armando/solaris/netscape
                        is_image_remote = n
                        keep_images = y
                        spool_directory = /var/spool/pkg
                        use_root_path = n
                        interactive = n
                        administration_file = /var/sadm/install/admin/unattend
                remove_absolutely = n
                package_instance
                   name= NSCPcom
                   description="Install English version for Netscape 4.78"
                end
    end
```

SPD File Example: install_solaris_patch

The following section shows an example of software package definition file containing an install_solaris_patch stanza:

```
install_solaris_patch
                        image_dir = /data/armando/adtool
                        source_dir = /tivoli/armando/solaris/patch/adtool
                        is_image_remote = y
                        keep_images = n
                        compression_method = stored
                        install_unconditionally = n
                        patch_id = 108721-04
                        force = n
    end
```

install_aix_package

The install_aix_package action enables you to distribute and install software products and updates using packages containing native AIX actions.

The SPD file stanza includes the following types of information:

- Identification information required for the installation, such as the name of the package file and the location of product image files to be used in the installation.
- Log options, including the level of logging, the location of the log file and whether the log must be reported to the Software Distribution server.
- Parameters that determine the method of installation. For example, whether the product images are included in the package (bundled installation) or located on a remote server (redirected installation).
- Entries for filesets.

  Each entry defined in the file includes the name of the fileset, its level and description.

Table 32 on page 105 provides details of the attributes of the install_aix_package stanza.

The following is the format of the install_aix_package stanza:

```
install_aix_package
                source_dir                  <pathname>
                image_dir                   <pathname>
                keep_images                 <bool>
                is_image_remote             <bool>
                log_path                    <pathname>
                log_mode                    <int>
                report_log                  <bool>
                override_files              <bool>
                install_root                <bool>
                install_share               <bool>
                install_usr                 <bool>
                cdrom_volume                <bool>
                save_directory              <pathname>
                expand_fs                   <bool>
                block_size                  <int>
                package_file                <ustring>
                compression_method          <deflated, stored>
                    is_update               <bool>
                SequenceOf<fileset>
                             fileset
                          name                  <ustring>
                          level                 <ustring>
                          description           <ustring>
                end
end
```

Some Software Distribution operations are limited or not available in the install_aix_package stanza. Table 30 describes the mappings between Software Distribution and installp command operations:

Table 30. Supported Software Distribution Operations in install_aix_package stanza

Software Distribution Operations	Installp Command Options
Install	installp -ac
Install repair	Software Distribution performs a check of the installed filesets and reinstalls them if they are not found.
Remove	installp -u

If the is_update attribute is set to y, an update installation is performed. Software Distribution operations are limited or not available in the stanza. Table 31 on page 105 describes the mappings between Software Distribution and installp command

operations for AIX patch installations:

Table 31. Supported Software Distribution Operations for AIX update installation

Software Distribution Operations	Installp Command Options
Install undoable	installp -a
Undo	installp -r
Accept	installp -c

Notes:

1. Running an install undoable on a software package containing an AIX update native installation object, the path for backup or alternate save directory, if specified, is not created.
2. The Software Distribution force option cannot be used with software packages containing an AIX update native installation object.

Attributes in AIX Package Stanza

Table 32 shows a list of all the attributes that can be defined in the install_aix_package stanza.

Table 32. SPD file attributes in Install AIX stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
block_size	Block size of the installation media.			
	Integer	No	512	install_aix_package
cdrom_volume	Specifies if a CD ROM is used as the installation device and you want to suppress multiple volume processing.			
	y: yes **n**: no	No	**n**	install_aix_package
compression_method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	install_aix_package
expand_fs	Specifies whether to expand the file system, if necessary			
	y: yes **n**: no	No	**n**	install_aix_package
fileset	Specifies the vector of filesets to be installed by the current package. The fileset is a specific class.			
	Vector	No		install_aix_package
image_dir	Full pathname of the directory on the target system that contains the product images. For bundled installations this is a local path; for redirected installations it is a network path. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	string	Yes		install_aix_package
is_image_remote	If set to **y**, specifies a redirected installation, where image files are obtained from a directory on a remote server, at installation time.			
	y: yes **n**: no	No	**n**	install_aix_package

Table 32. SPD file attributes in Install AIX stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
is_update	If set to **y**, an update, or patch installation, is performed.			
	y: yes **n**: no	No	**n**	install_aix_package
keep_images	Specifies whether product images are to be stored on the target system following installation. This parameter is relevant only for bundled installations.			
	y: yes **n**: no	No	**n**	install_aix_package
install_root	For diskless systems, specifies whether to install the root part.			
	y: yes **n**: no	No	**n**	install_aix_package
install_share	For diskless systems, specifies whether to install the share part.			
	y: yes **n**: no	No	**n**	install_aix_package
install_usr	For diskless systems, specifies whether to install the user part.			
	y: yes **n**: no	No	**n**	install_aix_package
log_path	Specifies the fully qualified path to the native installation log. This attribute, although set, has no meaning unless the log_mode attribute is enabled.			
	String	No		install_aix_package
log_mode	Specifies the verbose option for the pre-installation output.			
	Disabled, Default, Low, Medium, High	No	Disabled	install_aix_package
override_files	Files are replaced and cannot be recovered.			
	Boolean	No	**n**	install_aix_package
package_file	Name of the file containing the installp images of the AIX package to be installed.			
	String	No		install_aix_package
report_log	Specifies whether to include log entries in the log on the server. This attribute, although set, has no meaning unless the log_mode attribute is enabled.			
	y: yes **n**: no	No	**n**	install_aix_package
save_directory	Specifies the alternative path for backup.			
	String	No		install_aix_package

Table 32. SPD file attributes in Install AIX stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
source_dir	Pathname for the directory on the source host machine that contains the product images. This path must only be specified for a bundled installation. In this case, the image files are distributed with the package and downloaded to the directory identified in the image_dir attribute. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	String	No		install_aix_package

SDP File Example: install_aix_package

The following section shows an example of software package definition file containing an install_aix_package stanza:

```
install_aix_package
                image_dir = "/tmp/installp"
                source_dir = "/installp"
                         is_image_remote = n
                         keep_images = y
                         package_file = Adobe
                         log_mode = high
                         log_path = /tmp/installp/log_Acrobat
                         report_log = y
                         block_size = 512
                         override_files = n
                         install_root = n
                         install_share = n
                         install_usr = n
                         cdrom_volume = n
                         install_corequisites = n
                         is_update = n
                         expand_fs = n
                         save_directory = /tmp/installp/salva_Acrobat3.01

                     fileset
                        name = Adobe.acrobat
                                    level = 3.0.1.0
                                    description = "Adobe Acrobat reader for AIX"
                     end

end
```

install_rpm_package

The install_rpm_package action enables you to distribute and install software products and patches using RPM (Red Hat Package Manager) packages on Linux platforms. The install_rpm_package stanza contains the rpm_file sub-stanza which allows you to define the rpm packages to be installed. You can add a sequence of rpm_file sub-stanzas; all sub-stanzas inherit the attributes defined in the instal_rpm_package stanza.

The SPD file stanza that defines these actions includes the following types of information:

- Identification information required for the installation such as the name of one or more RPM files and the location of product image files to be used in the installation.
- Parameters that determine the method of installation. For example:

- Whether the product images are included in the package (bundled installation) or located on a remote server (redirected installation).
- The conditions for reinstalling files, for example, if the file is missing.

Each entry defined in the file includes the name of the RPM file and other information related to the action that can be taken, for example, install it on the local machine.

Table 34 on page 109 provides details of the attributes in the install_rpm_package stanza, Table 35 on page 110 provides details of the attributes in the rpm_file sub-stanza.

The following is the format of the install_rpm_package stanza and of the rpm_file sub-stanza:

```
install_rpm_package
                          rpm_options                  <ustring>
                          rpm_report_log               <ustring>
                          rpm_install_type             <pathname>
                          rpm_install_options          <ustring>
                          rpm_install_force            <bool>
                          rpm_install_nodeps           <bool>
                          rpm_verify_options           <ustring>
                          rpm_remove_options           <ustring>
                          rpm_remove_nodeps            <bool>
                          rpm_report_log               <bool>
                                  #sequence of rpm_file
                                    source_dir                    <pathname>
                                        image_dir                 <pathname>
                                    is_image_remote               <bool>
                                keep_images                       <bool>
                                    rpm_package_name              <ustring>
                                        rpm_package_file          <ustring>

  end
end
```

Some Software Distribution operations are limited or not available in the install_rpm_package stanza. Table 33 describes the mappings between Software Distribution and Linux operations:

Table 33. Supported Software Distribution Operations in install_rpm_package stanza

Software Distribution Operations	RPM Command Options
install	rpm -i rpm -f rpm -u
remove	rpm -e
verify	rpm -v

Attributes in install_rpm_package stanza

Table 34 on page 109 shows the attributes that can be defined in the install_rpm_package stanza.

Table 34. SPD file attributes in Install RPM stanza

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
compression_method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	rpm_file
rpm_install_force	Performs the installation ignoring any package or file conflicts.			
	y: yes **n**: no	No	**n**	install_rpm_package
rpm_install_nodeps	Ignores any dependency-related problems and completes the package installation.			
	y: yes **n**: no	No	**n**	install_rpm_package
rpm_install_options	Specifies the options related to rpm installation.			
	String	No	**Install**	install_rpm_package
rpm_install_type	Specifies the options related to rpm installation type.			
	Install, Update, Freshen	No	**Install**	install_rpm_package
rpm_options	Specifies the options related to rpm.			
	String	No	**n**	install_rpm_package
rpm_remove_nodeps	Ignores any dependency-related problems and completes the package installation.			
	y: yes **n**: no	No	**n**	install_rpm_package
rpm_remove_options	Specifies the options related to rpm removal.			
	String	No	**n**	install_rpm_package
rpm_report_log	Specifies the options related to rpm log.			
	y: yes **n**: no	No	**n**	install_rpm_package
rpm_verify_options	Specifies the options related to rpm verification.			
	String	No	None	install_rpm_package
rpm_file	Name of the RPM file of the product you want to install			
	Vector	Yes	None	install_rpm_package

Attributes in the rpm_file sub-stanza

Table 35 on page 110 shows all the attributes that can be defined in the rpm_file sub-stanza.

Table 35. SPD file attributes in rpm_file sub-stanza

Attribute	Comments			
	Values	**Required**	**Default**	**sub-stanza**
image_dir	Full pathname of the directory on the target system that contains the product images. For bundled installations this is a local path; for redirected installations it is a network path. The directory structure where the rpm file is located must contain only the images necessary to successfully install the product.			
	string	Yes	None	rpm_file
is_image_remote	If set to **y**, specifies a redirected installation, where image files are obtained from a directory on a remote server, at installation time.			
	y: yes **n**: no	No	**n**	rpm_file
keep_images	Specifies whether product images are to be stored on the target system following installation. This parameter is relevant only for bundled installations.			
	y: yes **n**: no	No	**n**	rpm_file
rpm_package_file	Specifies the name of one or more rpm files to be removed or verified. The .rpm extension is omitted.			
	y: yes **n**: no	No	**n**	rpm_file
rpm_package_name	Specifies the name of one or more package files to be installed (*.rpm).			
	y: yes **n**: no	No	**n**	rpm_file
source_dir	Pathname for the directory on the source host machine that contains the product images. This path must only be specified for a bundled installation. In this case, the image files are distributed with the package and downloaded to the directory identified in the image_dir attribute. The directory structure where the MSI file is located must contain only the images necessary to successfully install the product.			
	String	Conditional	None	rpm_file

SPD File Example: install_rpm_package

The following section shows an example of software package definition file containing an install_rpm_patchage stanza:

```
install_rpm_package
                    caption = peng
             rpm_options = n
                    rpm_install_type = Install
                    rpm_install_force = n
                    rpm_install_nodeps = y
                    rpm_remove_nodeps = n
                            rpm_file
                       image_dir = /tmp/rpm
                        source_dir = /source/test/rpm
                        is_image_remote = y
                        keep_images = y
```

```
                        rpm_package_name = xpenguins-2.0-1
                        rpm_package_file = xpenguins-2.0-1.i386.rpm
                end
end
```

install_hp_package

The install_hp_package action enables you to distribute software products and updates using packages containing native HP-UX actions.

The SPD file stanza includes the following types of information:

- Identification information required for the change management operation you want to perform, such as the name of the package file and the location of product image files to be used in the installation.
- Log options, including the level of logging, the location of the log file and whether the log must be reported to the Software Distribution server.
- Parameters that determine the method of installation. For example, whether the product images are included in the package (bundled installation) or located on a remote server (redirected installation).
- A stanza for each software selection.

  Each stanza includes the name of the software selection, its revision level and description.

Table 32 on page 105 provides details of the attributes of the install_hp_package stanza.

The following is the format of the install_hp_package stanza:

```
install_hp_package
     # Inerited from install_native_package
               source_dir                <pathname>
               image_dir                 <pathname>
               keep_images               <bool>
               is_image_remote           <bool>
               compression_method        <deflated, stored>

        # Specific HP-UX attributes

               log_path              <pathname>
               log_mode              <none,low, medium, high, very_high>
               report_log            <bool>
               gui_interaction       <bool>
               gui_options           <string>
               response_file_catalog <pathname>
               input_session_file    <pathname>
               output_session_file   <pathname>
               options               <string>
               options_file          <pathname>
               package_file          <string>
               is_patch              <bool>
               software_file         <pathname>

     # List of software selections

               software_selection
                         name        <string>
                         revision    <string>
                         description <string>
               end
     end
```

Some Software Distribution operations are limited or not available in the install_hp_package stanza. Table 30 on page 104 describes the mappings between

the Software Distribution operations and the HP-UX commands:

Table 36. Supported Software Distribution Operations in the install_hp_package stanza

Software Distribution Operation	HP-UX Command
install	swinstall -s *depot software_selection*
install and accept	Supported only against patches. Results in the following sequence of commands: 1. swinstall -s *depot* -x patch_save_files=true *software_selection* 2. swmodify -x patch_commit=true *software_selection*
accept	Supported only against patches. swmodify -x patch_commit=true *software_selection*
install undoable	Supported only against patches. swinstall -s *depot* -x patch_save_files=true *software_selection*
verify	Results in the following sequence of commands: 1. swlist -R *software_selection* 2. swverify *software_selection*
undo	Supported only against patches. swremove *software_selection*
remove	If the object is a patch that has been accepted, Software Distribution does not perform any action on the object. The entry created in the catalog is removed. Otherwise, the operation results in the following command: swremove *software_selection*
install repair	swinstall -s *depot* -x reinstall=true *software_selection*

Note: The undo operation is allowed only on packages that contain HP-UX native patches. According to the HP-UX behavior, when a patch is committed (In the Software Distribution environment when a patch is accepted), it cannot be removed from the system unless you remove the related application. For this reason, when you remove a software package that contains an HP-UX patch already accepted, Software Distribution removes only the related entry in the catalog. If you want to maintain the capability to remove HP-UX patches you must create a software package to install the application and a software package to install the related patches.

Attributes in HP-UX Package Stanza

Table 32 on page 105 shows a list of all the attributes that can be defined in the install_hp_package stanza.

Table 37. SPD file attributes in install_hp_package stanza

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
compression_method	Specify **deflated** for compressed, **stored** for uncompressed.			
	deflated, **stored**	No	**stored**	install_hp_package

Table 37. SPD file attributes in install_hp_package stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
description	Specifies the description of the software selection to be installed or removed			
	String	No		software_selection
gui_interaction	Specifies whether to turn on the following GUI options: • Background • Foreground • Display • Name • XRM If you specify this attribute the SD Software Selection window is displayed when the required change management operation for the HP-UX package is performed. The SD Software Selection window uses the XToolkit options you defined on this page. Refer to the XToolkit documentation for a detailed explanation of the GUI options.			
	y: yes **n**: no	No	**n**	install_hp_package
gui_options	Specifies the values associated to the GUI options. This attribute, although set, has no meaning unless the gui_interaction attribute is enabled.			
	String	No	bg=black fg=green display= name=ITCM xrm=	install_hp_package
image_dir	Specifies the directory where the product images are stored and from which the installation will be launched. This location must be accessible from all target systems. It can be a local directory on each target system or a directory on a network drive that is accessible from all the target systems. If you specify is_image_remote, image_dir contains the name of the file that contains the product images.			
	String	Yes		install_hp_package
input_session_file	Specifies the name of the input session file that contains the options and operands you used in the previous session command.			
	String	No		install_hp_package
is_image_remote	If set to **y**, specifies a redirected installation, where image files are obtained from a directory on a remote server, at installation time.			
	y: yes **n**: no	No	**n**	install_hp_package
is_patch	Specifies if you are creating a software package containing a patch. If set to **y** the installation of the package can be performed in undoable mode so that you can accept or undo the patch at a later time.			
	y: yes **n**: no	No	**n**	install_hp_package

Table 37. SPD file attributes in install_hp_package stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
keep_images	Specifies whether or not product images are to be stored on the target system after the installation is performed. Set this attribute to **n** to delete the images after the installation operation is performed.			
	y: yes **n**: no	No	**n**	install_hp_package
log_mode	Specifies the level of details to be written in the log stored on the server.			
	None Low Medium High Very high	No	None	install_hp_package
log_path	Specifies the fully qualified path where the log is to be saved. This attribute, although set, has no meaning unless the log_mode attribute is enabled.			
	String	Conditional		install_hp_package
name	Specifies the name of the software selection to be installed or removed			
	String	Yes		software_selection
options_file	Specifies the name of the option file you want to use to perform the required change management operation.			
	String	No		install_hp_package
options	Specifies the options you want to use for the required change management operation.			
	String	No		install_hp_package
output_session_file	Specifies the name of the output session file where you want to save the options and operands you are defining for the current package.			
	String	No		install_hp_package
package_file	Specifies the name of the file that contains the HP_UX source depot.			
	String	No		install_hp_package
report_log	Specifies whether to include log entries in the log stored on the server. This attribute, although set, has no meaning unless the log_mode attribute is enabled.			
	y: yes **n**: no	No	**n**	install_hp_package
response_file_catalog	Specifies the fully qualified path of the response file catalog that contains all the response files you can use to install the package without user intervention.			
	String	No		install_hp_package
revision	Specifies the revision level of the software selection to be installed or removed			
	String	No		software_selection
software_file	Specifies the name of the software file that contains the set of software selections.			
	String	No		install_hp_package

Table 37. SPD file attributes in install_hp_package stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
source_dir	Specifies the absolute path where the HP-UX source depot is located. If the images to be installed are stored in a file, specifies the file name of the HP-UX software depot. This path must only be specified for a bundled installation. In this case, the image files are distributed with the package and downloaded to the directory identified in the **image_dir** attribute. The directory structure where the HP-UX files are located must contain only the images necessary to successfully install the product.			
	String	Conditional		install_hp_package

SDP File Example: install_hp_package

This section contains the following examples:

- An install_hp_package stanza to install a patch.
- An install_hp_package stanza to install a product.

```
install_hp_package
                caption = "Netscape 7.0"
                image_dir = /tmp/Netscape
                source_dir = /depots/Netscape
                is_image_remote = n
                keep_images = n
                compression_method = stored
                log_mode = none
                gui_interaction = n
                is_patch = n
                software_selection
                        name = B6835AA
                        description = "Netscape  7.0"
                            revision ="7.0"
                end
        end

install_hp_package
             caption = "OS Patch"
             image_dir = /cdrom/GOLDQPK11i
             is_image_remote = y
             keep_images = n
             compression_method = stored
             log_mode = none
             gui_interaction = n
             options = "swinstall.autoreboot=true,swremove.autoreboot=true,
                        swinstall.patch_match_target=true"
           is_patch = y
 end
```

execute_user_program

The execute_user_program stanza allows you to launch programs from a software package. It can be used to execute pre- or post-installation programs, and as any program that terminates its execution without user intervention. To execute programs that do not terminate without intervention, they must be started from a batch or script file. For example, the following simple program (called pinging.sh) opens an AIX window, starts the **ping** command, then returns control to the system:

```
aixterm -e ping lab15027 &
```

The execute_user_program stanza starts the program pinging.sh. The output status and return status of the program are reported to stdout and stderr in the error log of the server.

Note: If you are not sure if the program will terminate, you can use the timeout attribute to terminate it after a given time.

You specify Uniform Naming Convention (UNC) names in the following format:

```
\\<servername>\<shared_resource>\<program.exe>
```

To use this name or **NET USE**, or any program that runs in the context of a particular user account, the endpoint must be installed specifying a particular account name that is authorized to access that resource. Alternatively, you can set access rights to resources on the endpoint by using the command **wlcftap.exe**. For more information about this command, see the *Tivoli Management Framework: Reference ManualTivoli Management Framework Reference Manual*.

You cannot use network paths to access shared, remote resources from within a software package. To access such resources, you must map the remote resource to a drive before submitting a distribution that requires access. For more information, see Article ID-Q124184 in the Microsoft Knowledge Base, which is available from the Microsoft Developers Network (MSDN). The URL of the MSDN Online Web site is: **http://msdn.microsoft.com/**.

Notes:

1. Change management (CM) operations, such as install, remove, commit, accept, verify, and undo, cannot be issued from a user program that is called from the execute_user_program stanza.
2. If execute_user_program.timout is equal to 0 and the user program is the last action in the software package, the timeout value set for the gateway repeater is used.
3. To execute graphical user programs that display dialog boxes and require user interaction, you should ensure that the workstation is neither locked nor logged off, and specify the attribute user_input_required=y. Insert the following lines in the execute_user_program action to run the program in foreground to the endpoint:

   ```
   Program path:                start
   Advanced options:    /max program_name_full_path
   ```

Running NetWare User Programs

Software Distribution supports the use of both .NCF (NetWare Control File) and .NLM (NetWare Loadable Module) user programs on target systems running the NetWare platform.

You can specify parameters to be used during the execution of an .NCF program. In addition, to receive a return code at the completion of the program, insert the following line at the end of the .NCF file:

```
load wsetrc <rc>
```

where *rc* is the desired return code.

Note: Return code -32767 is reserved and must not be used in this case.

However, since .NCF programs are not run in synchronous mode (that is, Software Distribution launches the .NCF program, then continues with other operations

without waiting for the program to complete), standard output or standard errors are not sent back to the log file or the server. Also, when the return code you set is returned, it signifies only that the program has completed, not that the commands inserted in the .NCF program have completed successfully.

Note: The default timeout value is -1 that is (infinite). You must specify the **wsetrc** command at the end of an NCF program, or define a finite timeout value, otherwise the execute_user_program does not end.

On the other hand, .NLM user programs are run in synchronous mode (that is, the program must complete before subsequent operations are begun).

Because of technical limitations in the NetWare NLM software developer's kit (SDK), you cannot pass a return code from a child NLM to its parent. To remedy this problem, the endpoint exports a symbol that refers to a function call within the endpoint. This function sets the return code with an endpoint before it returns.

When running a Perl script on a NetWare endpoint, observe the following guidelines:

- Use only Perl 5, as other versions are not supported.
- Make sure that Perl is not running on the server.
- Specify the fully qualified path to the script without specifying the Perl interp.
- In the last line of the script, insert the following instruction:

  ```
  system("unload perl")
  ```
- To set the return code, call the wsetrc.nlm program and add the following line:

  ```
  system("load wsetrc ReturnCode")
  ```

The following is an example of a task NLM that uses the Tivoli Management Framework function tiv_lcf_set_task_status and links to the library lcfutil5. This example sets the return code to 58.

```
#include <mrt/tiv_mrt.h>
#include <process.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/stat.h>
#include <mrt/mrtconst.h>
#include <mrt/log.h>
#include <cpl/printf.h>
#include <cpl/env.h>

void main ( int argc, char *argv[] )
{
int returncode = 58;
FILE* in = NULL;
FILE* out = NULL;
FILE* err = NULL;
char buf[BUFSIZ];
char* p;

SetAutoScreenDestructionMode(1);
/* do your task stuff first, setting return
   code where necessary */

/* TASK STUFF HERE*/

/* getting stdin information: the
   tiv_lcf_cpl_getenv is exported by
   lcfutil5.nlm
*/
p = tiv_lcf_cpl_getenv("TIV_LCF_STDIN")
                in = fopen(p, "r");
```

```
              /* getting stdout information */
p = tiv_lcf_cpl_getenv("TIV_LCF_STDOUT")
                out = fopen(p, "w");

              /* getting stderr information */
p = tiv_lcf_cpl_getenv("TIV_LCF_STDERR")
                err = fopen(p, "w");

if (out) {
                                  /*
                                 ..............
                                    */
                  fclose(out);
}
if (err) {
                                  /*
                                 ..............
                                    */
                  fclose(err);
}
if (in) {
                                  /*
                                 ..............
                                   */
                  fclose(in);
}

delay(10000);

     /* setting the return code as desired */

printf("calling tiv_lcf_set_task_status(%d)\n",
  returncode);

tiv_lcf_set_task_status(returncode);

 /* function exported by lcfutil5.nlm */
}
```

Format of the execute_user_program Stanza

The following is the format of the execute_user_program stanza, which includes a list of programs that can be executed during the various phases, for example, during_backup or during_remove:

```
execute_user_program
     # Inherited from action
     condition       = <string constrain>

     transactional   = <Boolean>

       # Sequence of <during_install, during_backup, during_remove,
       during_undo, during_accept, during_commit, during_verify,
       during_cleanup>
end
```

The elements of the sequence are user programs that are specified as follows. The same attributes apply to all phases, not only to the during_install phase shown below.

```
 during_install
     # Inherited from program
     caption                   = <string>
     path                      = <pathname>
     input_file                = <pathname>
     arguments                 = <string>
     inhibit_parsing           = <Boolean>
```

```
     working_dir               = <pathname>
     timeout                   = <integer>
     unix_user_id              = <integer>
     unix_group_id             = <integer>
     user_name                 = <string>
     user_input_required       = <Boolean>
     environment               = <string expression>

     #Specific user program attributes
     bootable                  = <Boolean>
     retry                     = <integer>
     exit_codes
          success                         = <range_value>
          success_reboot_now              = <range_value>
          success_reboot_now_reexecute    = <range_value>
          success_reboot_after            = <range_value>
          success_reboot_after_reexecute  = <range_value>
          success_in_a_reboot             = <range_value>
          warning                         = <range_value>
          failure                         = <range_value>
          fatal_failure                   = <range_value>
    end

    reporting_stdout_on_server = <Boolean>
    reporting_stderr_on_server = <Boolean>
    max_stdout_size            = <integer>
    max_stderr_size            = <integer>
    output_file                = <pathname>
    error_file                 = <pathname>
    output_file_append         = <Boolean>
    error_file_append          = <Boolean>

    corequisite_files
        # Sequence of <file, directory>
    end
end
```

A during_*operation* phase can contain one or more file or directory stanzas (for file system objects) called corequisite_files. You can use such a stanza when the program requires files or directories to be present on the target system during the execution of the program. These files and directories are temporary. That is, they are always moved from the service directory (to where they were downloaded during_install and from where they will be removed during_remove) to the specified destination location when the related program begins, and deleted when the program ends.

When you create a package that contains:

- user_program during_install with corequisite files (for example a file named coreq_install)
- user_program during_commit with corequisite files (for example a file name coreq_commit)

and install it in transactional, the two corequisite files (coreq_install and coreq_commit) are downloaded on the target. During the package installation, the coreq_install file is used and then deleted. During the package commit, the coreq_commit file is used but it is not deleted, because it might be used in the remove operation if performed in transactional mode.

The corequisite_file stanza comprises two different sets of attributes, the add_file set and add_directory set.

The add_file set comprises the following attributes:

- replace_if_existing
- replace_if_newer
- remove_if_modified
- rename_if_locked
- compression_method
- translate
- name
- destination

The add_directory set comprises the following attributes:
- replace_if_existing
- replace_if_newer
- remove_if_modified
- rename_if_locked
- compression_method
- location
- name
- destination
- descend_dirs

For more information about these attributes, see Table 11 on page 36.

If specified, a during_cleanup program is used to restore the machine to a stable state by completing operations that were pending during the execution of the package actions. A during_cleanup program can run during any software package operation, with the exception of the verify operation. To avoid placing the entire package in error status when its principal actions have been successfully completed, an error during the during_cleanup phase is managed not as a failure, as is the case for the other during_*operation* phases, but as a warning.

You can run a during_backup program during any distribution (either install or remove) that is run in undoable mode.

You can set a timeout attribute when you specify a program to be executed as part of a software distribution.

The value of this attribute can be set to either a number of seconds or to -1. If you set the timeout to a number of seconds and the program does not complete before the timeout expires, Software Distribution interrupts the program and the distribution ends in error status. If the timeout is set to -1, Software Distribution does not perform any action on the program. The distribution waits for the execute_timeout. If the value defined in the swdis.ini file for the send_timeout attribute is higher than the value of the execute_timeout attribute, the distribution waits for a number of seconds equal to the send_timeout value. If the execute_timeout is reached and the program did not complete the distribution interrupts. At the next retry, if the program did not complete, the distribution ends in changing (C) status. For more information, refer to *IBM Tivoli Configuration Manager: User's Guide for Software Distribution*.

Within the execute program stanzas, the timeout attribute can be defined for the following phases:
- during_install
- during_backup
- during_remove
- during_undo
- during_accept

- during_commit
- during_verify
- during_cleanup

Every operation, such as **winstsp**, **wremovsp -uy**, or **wcommtsp -cn**, corresponds to one or more program phases that can be executed, all specified in stanzas such as during_install, during_remove, and during_cleanup. Table 38 shows the program phases that correspond to each command.

Table 38. Commands and corresponding execute_user_program phases

Command	Program Phase
waccptsp	during_accept
	during_cleanup
wcommtsp	during_commit
	during_cleanup
winstsp	during_install
	during_cleanup
winstsp -ty	during_install [1]
	during_cleanup
winstsp –uy	during_backup
	during_install
	during_cleanup
winstsp –uy -a	during_backup
	during_install
	during_accept
winstsp –uy	during_backup
	during_install
	during_cleanup
wremovsp	during_remove
	during_cleanup
wremovsp –t y [–c y | o | r]	during_remove
	during_commit
	during_cleanup
wremovsp -uy	during_remove
	during_undo
	during_cleanup
wremovsp -ty	during_remove
	during_cleanup
wremovsp –uy -a	during_cleanup
	during_install
	during_accept
wundosp	during_undo
	during_cleanup

Table 38. Commands and corresponding execute_user_program phases (continued)

Command	Program Phase
wundosp --t y [–c y | o | r]	during_undo
	during_commit
	during_cleanup
wversp	during_verify

Note: This phase is performed only if in the execute_user_program stanza the transactional attribute is set to **yes**.

Setting Timeout Values for a Distribution

Tivoli Management Framework provides server- and client-level timeout parameters that enable you to specify a time interval after which either the server or client interrupts a distribution. Setting timeout parameters can avoid hung distributions caused by the following problems:

- Looping or hanging scripts, including all user programs or the type of script that requires user intervention, such as closing a status dialog box.
- ″Down″ nodes, including systems that are in the process of rebooting, offline, or completely disconnected from the network.
- Communication problems, such as breaks in the communication channel during distribution.

You must set distribution timeouts before distributing a Software Distribution profile. These timeouts include the following:

- Repeater timeout—sets the parameter to specify a timeout value for connections between a repeater and its endpoints or between repeaters. This value is set individually for each repeater using the **wmdist –s** command.
- User program timeout—sets the timeout keyword in the SPD file to specify a timeout value for user programs running on the client.

For more information about setting timeout values for a distribution, refer to the *User's Guide for Software Distribution*.

Exit Codes

Each user program has a sub-stanza where exit codes are defined. A range of numeric values between 0 and 65535 must be assigned to the exit codes for them to be interpreted as internal completion codes.

Table 39 details the various exit codes that can be returned from user programs, along with their effect on program operations:

Table 39. Exit code values

Exit Code	Operation Flow
success	Continue to the next action in the package.
success_reboot_now	Reboot immediately and continue to the next action in the package.
success_reboot_now_reexecute	Reboot immediately and run the operation again after the reboot.
success_reboot_after	Continue to the next action in the package and automatically reboot after all actions in the package have been run.

Table 39. Exit code values (continued)

Exit Code	Operation Flow
success_reboot_after_reexecute	Continue to the next action, reboot after all actions in the package have been run, and run the operation again.
success_in_a_reboot	Continue to the next action in the package. The status of the package is IC- - BC. The system must be manually rebooted.
warning	Continue to the next action and display a warning message.
failure	The operation cannot proceed because of errors. Continue to the next action if **stop_on_failure = no**, otherwise stop running.
fatal_failure	Stop running. If any corequisite files are specified in the software package, they are not removed.

Attributes in the execute_user_program Stanza and Its Elements

Table 40 shows a list of all the attributes that can be defined in the execute_user_program stanza and the during_*operation* elements that it includes.

Table 40. SPD file attributes in the execute_user_program stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
arguments	Only arguments of the program being executed are allowed. Command line or shell arguments such as the ampersand (&) character, which returns control to the shell, are not allowed.			
	String	No	None	execute_user_program
bootable	The program can issue a reboot command.			
	y: yes **n**: no	No	**n**	execute_user_program
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	execute_user_program
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	execute_user_program
environment	A list of environmental variables. The items in the list must be separated by commas. `path=c:\bin , TMP=C:\TEMP`			
	String expression	No	None	execute_user_program

Table 40. SPD file attributes in the execute_user_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
error_file	The full path of the file where the standard error (stderr) of the program is saved on the target system.			
	String	No	None	execute_user_program
error_file_append	If set to **y**, the standard error is appended to the existing error_file. If set to **n**, the existing error_file is overwritten.			
	y: yes **n**: no	No	**n**	execute_user_program
exit_codes	The sub-stanza for exit codes can include any of the following attributes: - success, success_reboot_now, success_reboot_now_reexecute, success_reboot_after, success_reboot_after_reexecute, success_in_a_reboot, warning, failure, fatal_failure Each exit code must be interpreted as an internal completion code. It must be specified as **min.** (minimum value in range), **max** (maximum value in range)			
	Range	No	For **success**, the default is **0**. For the other attributes the default ranges from **1** to **65535**.	execute_user_program
group_name	Specifies the group name necessary to obtain the specific GID under which the program is executed on a UNIX machine. Use this attribute in case the user name is present in several groups and you want to use a specific GID. This value overrides the GID value set at the package level. If no GID value is set, that is neither unix_group_id nor group_name are specified, the program is executed with root group privileges. This feature is supported for all UNIX endpoints excepting OS/400. If you do not specify this attribute the value specified for the unix_group_id attribute is taken.			
	String	No	None	execute_user_program
input_file	An input file to the program.			
	String	No	None	execute_user_program

Table 40. SPD file attributes in the execute_user_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
inhibit_parsing	Prevents the standard parsing of the values defined in the arguments attribute, if any. This value, although set, has no meaning unless the arguments attribute is defined. Normally, you do not need to modify this attribute. However, the following cases have been tested and found to work correctly: • Installation of Norton AntiVirus 7.5.1 • Installation of Microsoft Internet Explorer 5.5 using the syntax described in Article ID-Q260090 in the Microsoft Knowledge Base, which is available from the Microsoft Developers Network (MSDN). The URL of the MSDN Online Web site is: **http://msdn.microsoft.com/** This attribute is valid only on supported Windows operating systems, and is ignored on UNIX operating systems.			
	y: yes **n**: no	No	**n**	execute_user_program
max_stderr_size	Maximum number of standard error bytes sent to the server.			
	Integer	No	**10 000**	execute_user_program
max_stdout_size	Maximum number of standard output bytes sent to the server.			
	Integer	No	**10 000**	execute_user_program
output_file	The full path of the file where the standard output (stdout) of the program is saved on the target system.			
	String	No	None	execute_user_program
output_file_append	If set to **y**, the standard output is appended to the existing output_file. If set to **n**, the existing output_file is overwritten.			
	y: yes **n**: no	No	**n**	execute_user_program
path	The full path of the program.			
	String	Yes	None	execute_user_program
reporting_stderr_on_server	If set to **y**, the standard error produced by the program is sent to the server to be inserted in the log file of the server.			
	y: yes **n**: no	No	**n**	execute_user_program
reporting_stdout_on_server	If set to **y**, the standard output produced by the program is sent to the server to be inserted in the log file of the server.			
	y: yes **n**: no	No	**n**	execute_user_program

Table 40. SPD file attributes in the execute_user_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
retry	The maximum number of times the program is to be re-executed after it reboots.			
	Integer	No	**0**	execute_user_program
Sequence of	Specifies the different phases, during which specified programs are to run. For each phase specified here, you must define the attributes of the program. The remaining attributes listed in this table must be defined for each phase you include in the Sequence of attribute.			
	during_backup during_remove during_undo during_accept during_commit during_verify during_cleanup	Yes	None	execute_user_program
timeout	Time expressed in seconds to wait for the program's completion. Use this value only if you are certain that your program will complete. See "Format of the execute_user_program Stanza" on page 118 for detailed information.			
	Integer	No	**–1**	execute_user_program
transactional	Specifies whether the program must be executed in transactional mode. If set to **y**, you perform either the install, remove, or undo operation transactionally, and the program is executed. If set to **n**, the program is not executed during the initial (preparation) phase.			
	y: yes **n**: no	No	**n**	execute_user_program
unix_group_id	The UNIX group ID under which the program will run.			
	Integer	No	None	execute_user_program
unix_user_id	The UNIX user ID under which the program will run.			
	Integer	No	None	execute_user_program

user_input_ required	Set to **y** if the program requires user interaction. In addition to the **y** setting, in order to run a graphical user program, the workstation must be neither locked nor logged off. Set this attribute to **n** if the program does not open a user interface or require other user interaction. Insert the following lines in the execute_user_program action to run the program in foreground to the endpoint:
	`Program path: start`
	`Advanced options: /max/program_name_full_path`			

	y: yes **n**: no	No	**n**	execute_user_program

Table 40. SPD file attributes in the execute_user_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
user_name	The OS/2 user required to execute the program.			
	String	No	None	execute_user_program
working_dir	The directory from which the program starts.			
	String	No	None	execute_user_program

SPD File Example: execute_user_program

The following section shows an example of software package definition file containing an execute_user_program stanza:

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                          = "Appsample"
  title                         = "Sample Software Package"
  version                       = "1.0"

  execute_user_program
    transactional = n

    during_install

      exit_codes
        success                 = 0,0
        failure                 = 1,65535
      end

      path                      = "$(temp_dir)\log_aft.exe
      inhibit_parsing           = n
      timeout                   = -1
      user_input_required       = n
      output_file_append        = n
      error_file_append         = n
      reporting_stdout_on_server = y
      reporting_stderr_on_server = y
      max_stdout_size           = 10000
      max_stderr_size           = 10000
      bootable                  = n
      retry                     = 1

      corequisite_files

        directory
          location              = "C:\"
          name                  = "TMP"
          destination           = "$(temp_dir)"

          file
            name                = "log_aft.exe"
            destination         = "log_aft.exe"
            translate           = y
          end
        end
      end
    end
  end
end
```

execute_cid_program

The execute_cid_program action enables you to submit the installation of OS/2 applications that use the OS/2 Software Installer. The OS/2 Software Installer response file (.rsp file) is used as input to the SPD file.

```
execute_cid_program
     # Inherited from action
     condition             = <string constrain>

     # Sequence of <during_install,
       during_backup,
       during_remove,
       during_undo,
       during_accept,
       during_commit,
       during_verify,
       during_cleanup>

     maintenance_programs
        # Sequence of <cid_program>
     end
end
```

The elements of the sequence must be configuration, installation, and distribution (CID) programs specified as follows:

```
during_<operation>
     # Inherited from program
     caption               = <string>
     path                  = <pathname>
     input_file            = <pathname>
     arguments             = <string>
     working_dir           = <pathname>
     timeout               = <integer>
     unix_user_id          = <integer>
     unix_group_id         = <integer>
     user_name             = <string>
     user_input_required   = <Boolean>
     environment           = <string expression>

     # Specific cid program attributes
     reboot                = <Boolean>
     failure_as_fatal      = <Boolean>
end
```

Attributes in the execute_cid_program stanza and its Elements

Table 41 shows a list of all the attributes that can be defined in the cid_program stanza and the during_*operation* elements that it includes.

Table 41. SPD file attributes in the execute_cid_program stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
arguments	Only arguments of the program being executed are allowed. Command line or shell arguments such as the ampersand (&) character, which returns control to the shell, are not allowed.			
	String	No	None	cid_program

Table 41. SPD file attributes in the execute_cid_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	cid_program
condition	A valid expression (see "Defining Dependencies and Conditions " on page 8 for more information).			
	String constrain	No	None	cid_program
environment	The string that describes the environment under which the program must run, for example: `path=c:\bin;`			
	String expression	No	None	cid_program
failure_as_fatal	Specifies whether to handle a return code of failure as a fatal failure.			
	y: yes **n**: no	No	**n**	cid_program
input_file	An input file to the program.			
	String	No	None	cid_program
maintenance_programs	List of maintenance programs to be run before another CID program is run, for example, to prepare the environment.			
	List of CID programs	No	None	cid_program
path	The full path of the program.			
	String	Yes	None	cid_program
reboot	If set to **y**, the CID reboot return code is interpreted as an immediate reboot. If set to **n**, the CID reboot return code is interpreted as a reboot after the execution of all other commands in the software package.			
	y: yes **n**: no	No	**n**	cid_program
Sequence of	Specifies the different phases, during which specified programs are to run. For each phase specified here, you must define the attributes of the program. The remaining attributes listed in this table must be defined for each phase you include in the Sequence of attribute.			
	during_backup during_remove during_undo during_accept during_commit during_verify during_cleanup	Yes	None	cid_program

Table 41. SPD file attributes in the execute_cid_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
timeout	Time expressed in seconds to wait for the program's completion. Use this value only if you are certain that your program will complete. See "Format of the execute_user_program Stanza" on page 118 for detailed information.			
	Integer	No	**–1**	cid_program
unix_group_id	The UNIX group ID under which the program will run.			
	Integer	No	None	cid_program
unix_user_id	The UNIX user ID under which the program will run.			
	Integer	No	None	cid_program

user_input_ required	Set to **y** if the program requires user interaction. In addition to the **y** setting, in order to run a graphical user program, the workstation must be neither locked nor logged off. Set this attribute to **n** if the program does not open a user interface or require other user interaction. Insert the following lines in the execute_user_program action to run the program in foreground to the endpoint:
	`Program path: start`
	`Advanced options: /max/program_name_full_path`			

	y: yes **n**: no	No	**n**	cid_program
user_name	The Windows NT user required to execute the program.			
	String	No	None	cid_program
working_dir	The directory from which the program starts.			
	String	No	None	cid_program

SPD File Example: execute_cid_program

The following section shows an example of software package definition file containing an execute_cid_program stanza:

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                    = "cidret"
  version                 = "1.0"
  undoable                = o
  committable             = o
  history_reset           = n
  stop_on_failure         = y

  execute_cid_program

    during_install

      path                = c:\os2verify\maint.cmd
      timeout             = -1
      user_input_required = n
      reboot              = n
      failure_as_fatal    = y

    end

   maintenance_programs
    cid_program
```

```
     path                = c:\os2verify\ci1.cmd
    end
    cid_program
     path                = c:\os2verify\ci2.cmd
    end
    cid_program
     path                = c:\os2verify\ci3.cmd
    end
   end
 end
end
```

execute_mssetup_program

The execute_mssetup_program action enables you to run installations of Windows applications that use the Microsoft Setup installer. The information contained in the Microsoft Setup PDF file is used as input to create an SPD file. The following example is an excerpt from a Microsoft Setup PDF file, version 1.0.

```
[PDF]
   Version=1.0

    [Package Definition]
       Product=Microsoft Office 98
       Version=7.0
       Comment=Microsoft Office 98 Standard
       SetupVariations=Compact, Typical, Complete, Workstation,
         Custom, Uninstall

    [Compact Setup]
       CommandLine=setup.exe /Q1 /B2
       CommandName=Compact
       UserInputRequired=FALSE
       SupportedPlatforms=Windows98 (x86)
```

The following example is an excerpt from a Microsoft Setup PDF file, version 2.0

```
 [PDF]
     Version=2.0

         [Package Definition]
                                        Name=Norton AntiVirus Corporate Edition
            Version=7.5
            Comment=Norton AntiVirus Corporate Edition 7.5
            Publisher=Symantec
            Language=English
            Programs=Complete, Deinstall, Manual

         [Complete]
            Name=Complete
            CommandLine=setup.exe /v"/qn /mnavwnt"
            StartIn=.
            EstimatedDiskSpace=Unknown
            EstimatedRunTime=Unknown
            SupportedClients=Win NT (i386), Win 9x
            Win NT (i386) MinVersion1=4.00.0000.0
            Win NT (i386) MaxVersion1=4.00.9999.9999
            Win NT (i386) MinVersion2=5.00.0000.0
            Win NT (i386) MaxVersion2=5.00.9999.9999
            UserInputRequired=FALSE
            AdminRightsRequired=TRUE

         [Deinstall]
            Name=Deinstall
            CommandLine=setup.exe /x /v/qn
            StartIn=.
            EstimatedDiskSpace=Unknown
```

```
   EstimatedRunTime=Unknown
   SupportedClients=Win NT (i386), Win 9x
   Win NT (i386) MinVersion1=4.00.0000.0
   Win NT (i386) MaxVersion1=4.00.9999.9999
   Win NT (i386) MinVersion2=5.00.0000.0
   Win NT (i386) MaxVersion2=5.00.9999.9999
   UserInputRequired=FALSE
   AdminRightsRequired=TRUE

[Manual]
   Name=Manual
   CommandLine=setup.exe
   StartIn=.
   EstimateDiskSpace=Unknown
   EstimateRunTime=Unknown
   SupportedClients=Win NT (i386), Win 9x
   Win NT (i386) MinVersion1=4.00.0000.0
   Win NT (i386) MaxVersion1=4.00.9999.9999
   Win NT (i386) MinVersion2=5.00.0000.0
   Win NT (i386) MaxVersion2=5.00.9999.9999
   UserInputRequired=TRUE
   AdminRightsRequired=TRUE
```

In this examples, the information in the CommandLine entry is used to specify the program name and arguments in the SPD file.

Following is a sample execute_mssetup_program SPD file stanza:

```
execute_mssetup_program
      # Inherited from command
        condition         = <string constrain>

      # Sequence of <during_install, during_remove>
end
```

The elements of the sequence must be Microsoft Setup programs specified in the following way:

```
during_<operation>
      # Inherited from program
      caption               = <string>
      path                  = <pathname>
      arguments             = <string>
      working_dir           = <pathname>
      timeout               = <integer>
      user_input_required   = <Boolean>

      # No specific mssetup program attributes

      corequisites
         # Sequence of <file, directory>
      end
end
```

Attributes in the execute_mssetup_program stanza and its Elements

Table 42 on page 133 shows a list of all the attributes that can be defined in the execute_mssetup_program stanza and the during_*operation* elements that it includes.

Table 42. SPD file attributes in the execute_mssetup_program stanza

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
arguments	Only arguments of the program being executed are allowed. Command line or shell arguments such as the ampersand (&) character, which returns control to the shell, are not allowed.			
	String	No	None	execute_mssetup_ program
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	execute_mssetup_ program
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	execute_mssetup_ program
path	The full path of the program.			
	String	Yes	None	execute_mssetup_ program
Sequence of	Specifies the different phases, during which specified programs are to run. For each phase specified here, you must define the attributes of the program. The remaining attributes listed in this table must be defined for each phase you include in the **Sequence of** attribute.			
	during_install during_remove	Yes	None	execute_mssetup_ program
timeout	Time expressed in seconds to wait for the program’s completion. Use this value only if you are certain that your program will complete. See “Format of the execute_user_program Stanza” on page 118 for detailed information.			
	Integer	No	**–1**	execute_mssetup_ program

user_input_ required	Set to **y** if the program requires user interaction. In addition to the **y** setting, in order to run a graphical user program, the workstation must be neither locked nor logged off. Set this attribute to **n** if the program does not open a user interface or require other user interaction. Insert the following lines in the execute_user_program action to run the program in foreground to the endpoint:
	`Program path: start`
	`Advanced options: /max/program_name_full_path`			

	y: yes **n**: no	No	**n**	execute_mssetup_ program
working_dir	The directory from which the program starts.			
	String	No	None	execute_mssetup_ program

SPD File Example: execute_mssetup_program

This example shows the usage of the execute_mssetup_program action to install a Microsoft Office 98 application. The install action accesses the network drive

containing the application images, installs the application, then disconnects from the network drive. The remove action accesses the network drive containing the application images, uninstalls the application, then disconnects from the network drive.

```
'TIVOLI software package v4.3.1 - SPDF'
package
    name                      = "MS_OFFICE_98"
    title                     = "Office 98 setup"
    description               = "Execute MS setup to
      Install Office"
    copyright                 = "Tivoli"
    version                   = 1.0
    stop_on_failure           = y

    default_variables
      install_dir             = $(freedrive_01)\office98
    end

     execute_user_program  #Connect the network drive where
     # application images are stored

      condition = "$(os_name) == Windows_NT
       during_install
          path                = $(system_dir)\net
          arguments           = "use $(freedrive_01)
             \\argus\scratch"
          working_dir         = $(system_dir)
          user_input_required = n
          timeout             = -1
       end  # during_install

      during_remove
          path                = $(system_dir)\net
          arguments           = "use $(freedrive_01)
            \\argus\scratch"
          working_dir         = $(system_dir)
          user_input_required = n
          timeout             = -1
      end  # during_remove
    end  # execute_user-program

    execute_mssetup_program  # Start MSsetup
    # installation/uninstallation

      condition = "$(os_name) == Windows_NT
      during_install
          path                = $(install_dir)\setup.exe
          arguments           = "/Q1 /B4"  #execute
          #workstation setup
          working_dir         = $(install_dir)

          user_input_required = n
          timeout             = -1
      end  # during_install

      during_remove
          path                = $(install_dir)\setup.exe
          arguments           = "/Q1 /U"    #execute uninstall
          working_dir         = $(install_dir)
          user_input_required = n
          timeout             = -1
      end  # during_remove
    end  # execute_mssetup_program

   execute_user_program   # Disconnect network Drive
     condition = "$(os_name) == Windows_NT
```

```
      during_install
          path                = $(system_dir)\net
          arguments           = "use $(freedrive_01) /delete"
          working_dir         = $(system_dir)

          user_input_required = n
          timeout             = -1
      end  # during_install

      during_remove
          path                = $(system_dir)\net
          arguments           = "use $(freedrive_01) /delete"
          working_dir         = $(system_dir)
          user_input_required = n
          timeout             = -1
      end  # during_remove
    end  # execute_user-program
  end
```

execute_installshield_program

The execute_installshield_program action enables you to run installations of Windows applications that use the InstallShield installation program. The InstallShield .ISS response file is used as input to an SPD file.

Use the following criteria to determine if this stanza should be used to install a particular application program:

- The program must create an InstallShield log file located in the position specified with the log_file_path attribute.
- The program must allow the use of the **–f2** command line switch to customize the location of the log file. If **–f2** is not allowed, the value specified for log_file_path must be the same location where the log file is to be created.

If these criteria are not met, use the execute_user_program action instead (see “execute_user_program” on page 115).

The following is the format of the **execute_installshield_program** stanza:

```
execute_installshield_program
      # Inherited from command
      condition            = <string constrain>
      automatic_uninstall  = <Boolean>

      <during_install>
end
```

The elements of the sequence must be InstallShield programs specified in the following way:

```
during_<operation>
      # Inherited from program
      caption              = <string>
      path                 = <pathname>
      timeout              = <integer>

      # Specific InstallShield program attributes
      response_file_path   = <pathname>
      log_file_path        = <pathname>
      silent               = <Boolean>

      corequisites
         # Sequence of <file, directory>
      end

end
```

Attributes in the execute_installshield_program stanza and its Elements

Table 43 shows a list of all the attributes that can be defined in the execute_installshield_program stanza and the during_*operation* elements that it includes.

Table 43. SPD file attributes in the execute_installshield_program stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
automatic_uninstall	During the remove phase, if set to **y**, the Software Distribution agent automatically runs the InstallShield uninstall program that is registered in the registry database. Otherwise, no remove action is run.			
	y: yes **n**: no	No	**y**	execute_installshield_program
uninstall_response_file	If set to **y**, the InstallShield response file is uninstalled when the InstallShield uninstall program is run.			
	y: yes **n**: no	No	**n**	execute_installshield_program
caption	The name of the program, by default, the file name (final) token of the value of the path attribute.			
	String	No	File name token of the path attribute	execute_installshield_program
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	execute_installshield_program
Sequence of during_install	Specifies the attributes to be used during the installation. The remaining attributes are included within this sequence.			
	during_install during_remove	Yes	None	execute_installshield_program
log_file_path	Pathname for the InstallShield log file.			
	String	No	**instshld.log**	execute_installshield_program
path	The full path of the program.			
	String	Yes	None	execute_installshield_program
response_file_path	Pathname for the InstallShield response file.			
	String	No	**setup.iss**	execute_installshield_program
silent	If set to **y**, specifies a silent installation. If set to **n**, specifies an interactive installation.			
	y: yes **n**: no	No	**y**	execute_installshield_program

Table 43. SPD file attributes in the execute_installshield_program stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
timeout	Time expressed in seconds to wait for the program's completion. Use this value only if you are certain that your program will complete. See "Format of the execute_user_program Stanza" on page 118 for detailed information.			
	Integer	No	**–1**	execute_installshield_program

SPD File Example: execute_installshield_program

The following section shows an example of software package definition file containing an execute_installshield_program stanza:

```
'TIVOLI software package v4.3.1 - SPDF'
package
  name                  = "InstallShield"
  title                 = "InstallShield"
  description           = "Install Remote_Control pgm"
  copyright             = "Tivoli"
  version               = 1.0
  stop_on_failure       = y

  default_variables
    install_dir         = C:\usr\sd40\install\rem_control
  end

  execute_installshield_program
    condition           = "$(os_name) == Windows_NT
    automatic_uninstall  = y

    during_install
      path              = $(install_dir)\setup.exe
      timeout           = -1
      response_file_path = $(install_dir)\setup.iss
      log_file_path     = $(install_dir)\install.log
      silent            = y
    end  # during_install
  end  # execute_installshield_program
end
```

System Actions

This section describes the following stanzas:

restart Insert this action in your software package if a computer or operating system restart is necessary with the installation of an application.

check disk space
Include this check in your software package to be sure there is sufficient disk space on the target machine

logoff Include this check in your software package if a computer or operating system logoff is necessary with the installation of an application.

restart

In this section, the format of the restart stanza, and the details on each attribute of restart stanza, are provided.

Note: The restart stanza is ignored totally on all UNIX platforms, and in transactional operations on all other platforms.

```
restart
     # Inherited from action
     condition       = <string constrain>

     during_install  = <none, after, immediately>
     during_remove   = <none, after, immediately>
     during_undo     = <none, after, immediately>
     during_commit   = <none, after, immediately>
end
```

Table 44. SPD file attributes of the restart stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanzas**
condition	A valid expression (see “Defining Dependencies and Conditions ” on page 8 for more information).			
	String constrain	No	None	restart
during_install	Specifies when a restart must be performed during the installation of the package. If set to **immediately**, the restart takes place when the restart statement is encountered. If an install transactional operation is performed, this attribute is ignored and the machine is restarted when the commit operation is performed.			
	none, **after**, **immediately**	No	**after**	restart
during_remove	Specifies when a restart must be performed during the removal of the package. If set to **immediately**, the restart takes place when the restart statement is encountered.			
	none, **after**, **immediately**	No	**after**	restart
during_undo	Specifies when a restart must be performed during the undo of the package. If set to **immediately**, the restart takes place when the restart statement is encountered.			
	none, **after**, **immediately**	No	**after**	restart
during_commit	Specifies when a restart must be performed during the commit of the package. If set to **immediately**, the restart takes place when the restart statement is encountered.			
	none, **after**, **immediately**	No	**after**	restart

Table 44. SPD file attributes of the restart stanza (continued)

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
force_restart	Specifies if the reboot action on the endpoint must be forced. This attribute is valid only on Windows systems. Possible values are **y** and **n**. The default value is **n**. This attribute interacts with the timeout attribute, as described in the following list: • If the default values are used, that is the timeout attribute is set to -1 and the force_restart attribute is set to **n**, a soft reboot is invoked and, in case it fails, Software Distribution retries for an infinite time. • If the timeout attribute is set to -1 and the force_restart attribute is set to **y**, a hard reboot is performed immediately. • If the timeout attribute is higher than or equal to zero and the force_restart attribute is set to **n** a soft reboot is invoked. If the reboot fails, the distribution fails after the timeout expires. • If the timeout attribute is greater than or equal to zero and the force_restart attribute is set to y, a soft reboot is invoked. If it fails, a hard reboot is performed after the timeout expires and when the gateway reconnects to the endpoint.			
	y, **n**	No	**n**	restart
timeout	Specifies the number of seconds Software Distribution must wait before the reboot fails. If a retry interval occurs during this timeout, the distribution interrupts and the checkpoint and restart feature is used to retry later. The default value is -1. This means that Software Distribution retries the operation for an infinite time.			
	Integer	No	**-1**	restart

check_disk_space

To perform a check of the disk space available on the target system, use the check_disk_space stanza, the format of which is as follows:

```
check_disk_space
     # Sequence of <volume>
end
```

Table 45. SPD file attribute of the check_disk_space stanza

Attribute	Comments			
	Values	**Required**	**Default**	**Stanzas**
volume	The volume must be specified as follows: *drive*,*space*, for example, `Volume = c:,10M`. Space can be expressed in the following units: **B** Bytes **K** Kilobytes **M** Megabytes **G** Gigabytes If only a number is indicated, the default unit is **B** (bytes).			
	String expression	Yes	None	check_disk_space

SPD File Example: check_disk_space

The following section shows an example of software package definition file containing a check_disk_space stanza:

```
'TIVOLI Software Package v4.3.1 – SPDF'

package
```

```
  ##
  ## Package attributes
  ##
  name              = "Check_disk"
  title             = "check disk space"
  description       = "Checks free disk space"
  copyright         = "Tivoli Systems"
  version           = 1.0
  stop_on_failure   = n

    check_disk_space

    volume          = /,10M

  end
end
```

Note: On UNIX systems, the check disk space action searches for the file system beginning with the final token of the path specified, and moves to the previous token, until it finds the file system. By default, if no file system is detected, the disk space is checked on root. No warning is issued if no file system is available in the specified token.

logoff

To perform a logoff operation on Windows endpoints, use the logoff stanza, the format of which is as follows:

```
logoff
force = y/n (default = n)
force_if_locked = y/n (default = n)
during_install = y/n (default = n)
during_commit = y/n (default = n)
during_undo = y/n (default = n)
during_remove = y/n (default = n)
end
```

Table 46. SPD file attribute of the logoff stanza

	Comments			
Attribute	**Values**	**Required**	**Default**	**Stanza**
force	The logoff operation is forced also if any applications are currently active on the workstation.			
	String expression	No	n	logoff
force_if_locked	The logoff operation is forced if the workstation is locked.			
	String expression	No	n	logoff
during_install	The logoff operation is performed during the during_install phase.			
	String expression	No	n	logoff
during_commit	The logoff operation is performed during the during_commit phase.			
	String expression	No	n	logoff
during_undo	The logoff operation is performed during the during_undo phase.			
	String expression	No	n	logoff
during_remove	The logoff operation is performed during the during_remove phase.			
	String expression	No	n	logoff

SPD File Example: logoff

The following section shows an example of software package definition file containing a logoff stanza:

```
'TIVOLI Software Package v4.3.1 – SPDF'
package
##
## Package attributes
##
logoff
force = y
force_if_locked = n
during_install = n
during_commit = y
during_undo = n
during_remove = n
end
end
```

The logoff Check

Chapter 2. Performing Change Management Operations

After you create a software package and catalog it at the Software Distribution server, you can use change management operations to manage how package actions are performed. Change management operations, which are listed under "Types of Change Management Operations" on page 144, can be performed in the following modes:

- Transactional
- Undoable
- Undoable-in-transactional

Transactional Mode

Transactional mode splits the execution of an install or remove operation into two phases: the *preparation phase* and the *commit phase*. During the preparation phase, each action in the package prepares the conditions for the successful execution of the requested operation, which reduces the risk of failure during the commit phase. If the preparation phase completes normally, in the case of an install, the files are installed in the staging area. At this point the commit phase begins, where the updates take effect, that is, files are moved from the staging area to the production area. If the preparation phase fails, the system is returned (rolled back) to its original stable state, as shown in the following figure.

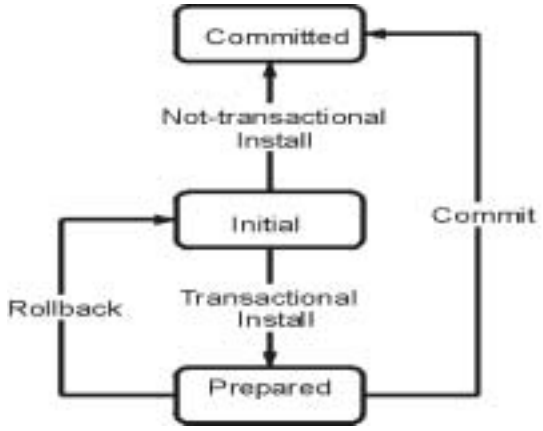


Figure 10. Transactional mode

See "Transactional Cycle" on page 153 for examples of operations using this mode.

Undoable Mode

Undoable mode enables you to undo an operation, even if you have installed it in the active area (and if it is already committed). This mode is useful when you are not sure about the results of an operation, because you can return the system to its previous state by undoing the operation. Also, if an operation run in undoable mode is unsuccessful, the system will automatically be rolled back to its original stable state.

An operation to remove an object cannot be undone, unless the object has been previously installed with the undoable option.

Undoable Mode

Performing an operation in undoable mode involves the creation of a backup package, which could be quite large, depending on the size of the system. When you perform an accept operation, the backup package that was created is deleted and the space in the backup area is made available, so the previous operation is no longer undoable.

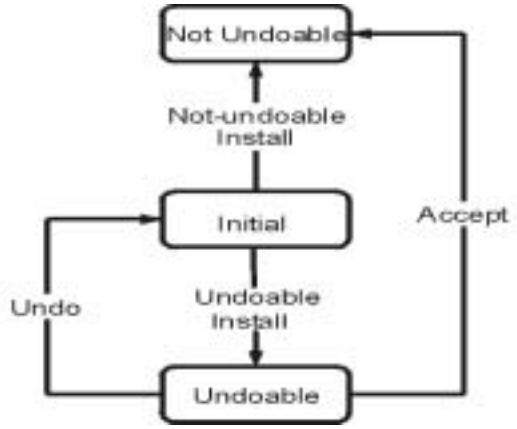


Figure 11. Undoable mode

See "Undoable Cycle" on page 154 for examples of operations using this mode.

Undoable-in-transactional Mode

In the *undoable-in-transactional* mode, the operation is performed in two phases: preparation and commit. The undoable-in-transactional mode defines that an operation is undoable and that a backup package is built during the commit phase, during which all updates performed in the preparation phase take effect. This mode requires reserved space for both a staging area and a backup area.

See "Undoable-in-Transactional Cycle" on page 154 for examples of operations using this mode.

Transactional-and-undoable Mode

Transactional-and-undoable mode is the union of the transactional and undoable modes. The operation is performed in a transactional way and is undoable, however, the backup package is not built in a transactional way.

See "Transactional-and-Undoable Cycle" on page 155 for examples of operations using this mode.

Types of Change Management Operations

Software Distribution supports various operations that act on software packages:

install Performs the actions listed in a software package.

remove Uninstalls a software package.

undo Returns the system to its previous state, prior to the last install or remove operation.

accept Deletes the backup package, so the previous operation can no longer be undone.

commit Causes all the updates performed in the preparation phase to take effect.

verify Verifies the consistency of the software package and the object on the target system. If the verify operation fails, the software package is placed in an error state.

load Loads software packages on a repeater depot for subsequent distribution. This operation is valid only for built software packages.

unload Removes software packages from a repeater depot. This operation is valid only for built software packages.

Each of these operations is explained in more detail in the following sections.

Data Moving Operations

Software Distribution provides the following commands to send, retrieve, and delete files from machines in a Tivoli Management environment:

Send Transfers a single file, multiple files, or directories from an origin system, including endpoints, to a set of specified Tivoli endpoints in your Tivoli environment.

Retrieve
Transfers a single file, multiple files, or directories from specified Tivoli endpoints to a source host.

Delete Deletes a single file, multiple files, or directories from one or more endpoints.

For more information on data moving operations, see the command “wspmvdata” on page 229 and refer to the *User’s Guide for Software Distribution*.

Install Operation

The install operation performs the actions listed in a software package. For example, the add file action copies a file to the target file system, while the add win_registry_key action adds a registry key to the Windows registry. The install operation options include:

transactional
Requests the system to either leave itself in a consistent state after an operation or, if at least one action does not succeed, to abort the operation and return itself to its initial state. The operation is performed in two phases: preparation and commit.

preferably-not-transactional
Requests the system to execute an operation only if it detects that it cannot proceed because of temporary errors that could disappear during the commit phase.

auto-commit
Requests the system to automatically commit a pending operation.

undoable
Requests the ability to return the system to its previous state because a backup package is created.

Note: This option is recommended when processing a software package that installs system files, because the operation can then be undone without damaging the system.

preferably-undoable
Requests the system to continue an operation even if the attempt to acquire backup space fails.

auto-accept
Requests the system to automatically accept an undoable operation.

undoable-in-transactional
Requests the system to create the backup package to be used in the undo phase during the commit phase, rather than during the install phase.

Installation Options

The installation options modes that you can specify for file formats other than software package block (built format) include the following:

all Installs all the files in the software package. This is the default option.

src Installs only those source host files that have been modified since the last successful distribution to the target system. This mode is applicable only to unbuilt software packages.

repair Installs the following:
- The source objects that have been corrupted, or modified since the time of the last successful installation, or are not present on the target. This makes the target objects consistent with the source objects.
- The objects and actions on the target that have been changed or corrupted since the time of the last successful installation.

delta Creates a software package that contains only the changes that the new package defines compared to a base package that is already installed on the target system. For a detailed explanation of how byte-level differencing works, see “Byte-level Differencing” on page 148.

Remove Operation

The remove operation uninstalls a software package, that is, it performs the opposite action of the install operation.

Note: Do not use the remove operation when removing a software package that has installed system files. Use the undo operation, if applicable.

The remove options include:
- Transactional
- Preferably-not-transactional
- Auto-commit
- Undoable
- Preferably-undoable
- Auto-accept
- Undoable-in-transactional

Software packages are removed by default in the inverse order with respect to the installation order. This means that the last software package to be installed is the first to be removed.

To modify this behavior, you can set the _REVERSE_ORDER_DURING_REMOVE variable to **Yes** at the Tivoli server level. In this case, the packages are removed in the order in which they were installed and this setting applies to all packages

created in the Tivoli region. You can also choose to ignore this setting for single packages by adding the INHIBIT_REVERSE_ORDER_DURING_REMOVE variable in the default_variables stanza in each software package and setting it to **YES**. For more information on the default_variables stanza, see Default variables on page 8.

Undo Operation

The undo operation returns the system to the state prior to the execution of the last install or remove operation. This operation differs from a remove operation, which effectively reverses the results of a previous operation. For example, consider an initial state represented by the following graphic:

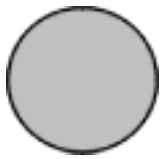


Figure 12. Initial state

After executing, for example, a draw circle action, the result is as follows:

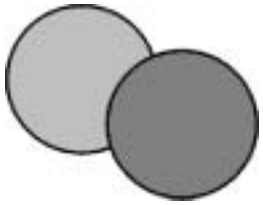


Figure 13. Result of draw circle action

Undoing the previous operation returns to the initial state represented by Figure 12 while the remove operation produces the following result:

Figure 14. Result of remove operation

Because of this behavior, in the case of software packages that install system files, you should not remove such packages with the remove operation. Instead, when installing these packages, use the undoable option, so that the installation can be undone, if necessary.

The undo options include:
- Transactional
- Preferably-not-transactional
- Auto-commit

Accept Operation

The accept operation deletes the backup package, so the previous operation can no longer be undone.

Commit Operation

The commit operation causes all the updates prepared in the preparation phase to take effect. When defining a commit operation, you can specify different reboot options, as described below:

in-a-reboot
To perform the commit operation with a reboot when only the operating system is running. This option prepares the system to complete the commit phase the next time the machine is rebooted by the user. If you perform a transactional installation and select this option from the command line, the report containing the result of the distribution is sent to the Tivoli server after a second operation is performed on the target.

not-in-a-reboot
To perform the commit operation without a potentially unnecessary machine reboot; however, this option increases the possibility of errors due to locked resources.

auto-reboot
Same as the in-a-reboot option, but the reboot operation is automatically run by the Software Distribution agent.

reboot-only-if-necessary
To perform the commit operation with a reboot only if files are locked on the target system at the time of distribution.

Verify Operation

The verify operation analyzes the consistency of each action in a software package with the state of the target system. If this operation fails, the software package is placed in an error state and you are notified of the list of actions that failed.

The verify operation checks whether files contained in the package are also present on the target system. The operation is successful if the date of the file on the target is the same as or later than the date of the same file in the package. If the date of the file is older than the date of the same file in the package, the operation fails.

The verify operation does not check whether a file has been changed since it was installed. To perform this check, run a repair operation.

Note: If a package contains two actions that perform opposite operations, such as adding and removing the same object, one of the operations will fail verification.

Load and Unload Operations

The load and unload operations respectively store software packages in a directory on a repeater depot and remove those software packages from the repeater depot. These operations are valid only for built software packages.

Byte-level Differencing

Byte-level differencing installs a software package using the byte-level differencing technology. The byte-level differencing function compares a software package to be installed (version package) with the base version (base package) that already exists on the target. On the source host a delta package is created for all the differences found between the version package and the base package, where both the base package and the version package reside on the source host. Because only the delta

package, which is typically much smaller than either the version or base packages, is sent on the network, network traffic is considerably reduced. The new version of the software package is recreated on the target by applying the changes contained in the delta package to the base package.

How Byte-level Differencing Works

Byte-level differencing compactly encodes a version of a file as a set of changes from a previous version. The delta file consists of any new files not in the base version and the differences between existing files in the two versions. A differencing algorithm finds and outputs the changes made between two versions of the same file by locating common strings to be copied and unique strings to be added.

The process can be represented as follows:

Version File - Base File = Delta File

Reconstruction, the inverse operation, requires the base file and a delta file to rebuild a version:

Base File + Delta File = Version File

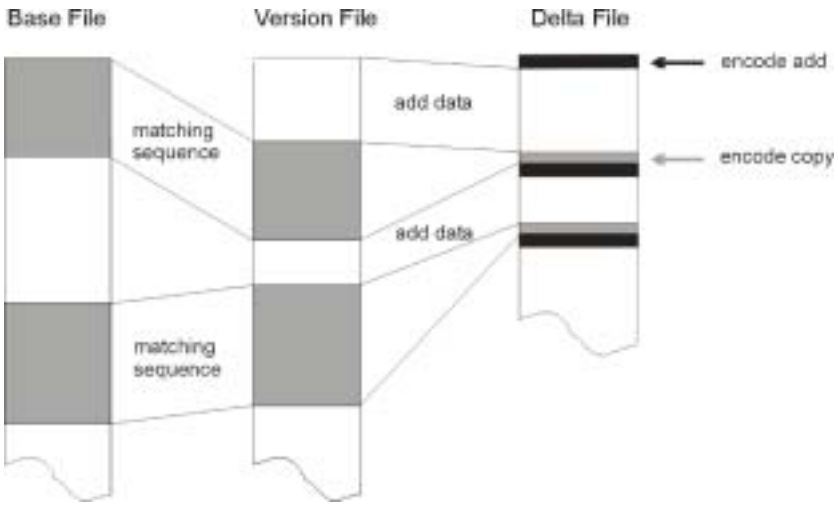


Figure 15. Byte-level differencing

The delta file is smaller than the version file.

How Software Distribution Uses Byte-level Differencing

Any distributed application that updates data frequently should take advantage of byte-level differencing to reduce the network traffic. Software Distribution can use byte-level differencing to distribute upgrades of software applications to a wide range of endpoints. To upgrade an already existing software package (base package) to a new version (version package), Software Distribution applies the differencing algorithm to each file contained in the version package that is delta compressible and that is found in the base package with the same fully qualified path. For a detailed explanation on how to define a file as delta-compressible see Table 11 on page 36.

To apply the delta installation, the base and the version packages must have the same nested structure. In addition, if the base and the version packages use the same file in the same directory, the packages must be in software package block format, otherwise the whole file is distributed. The above operation is performed on the source host and generates a delta package that is sent to the target.

On the target, Software Distribution uses the files contained in the delta package to reconstruct each version file starting from the base file already installed on the target and applying to it the corresponding delta file contained in the delta package. If any of the installed files to be reconstructed are not found on the target, have been modified, or are locked, the delta installation fails. Read-only files, if any, are overwritten.

The operations of loading and installing from a depot can also take advantage of the byte-level differencing technology because only the delta package is loaded on the depot for a subsequent byte-level differencing installation. The delta package can be also unloaded from a depot.

Byte-level differencing is most efficient when the new version of the software package that you want to install contains few differences compared with the base version. If the differences are more substantial, it is more efficient to reinstall the new version.

Software Package States

The operational state of a software package can be viewed using Inventory, by viewing the system log file, or by running the **wdlssp** command locally on an endpoint system where the Software Package Editor has been installed. Running this command produces a list of the software packages installed on a target system, as well as the version and the state of each software package. For more information on the use of the **wdlssp** command, see “wdlssp” on page 286.

The state is represented by a five-character string. Each character of the string represents a category of information about the package, which can be assigned one of a number of values. Table 47 summarizes the character positions, categories, meanings, and possible values of the state string.

Table 47. Software package states

Position	Category	Meaning	Possible Values
1	Operation name	The last operation performed on the package	**I**: installed **R**: removed
2	Package state	The state of the package	**P**: prepared **C**: committed
3	Backup package (undo) state	The state of the backup package (if any)	**P**: prepared **R**: restored **U**: undoable **–**: none
4	Reboot state	Whether a reboot is needed to complete the operation.	**B**: reboot **–**: none

Table 47. Software package states (continued)

Position	Category	Meaning	Possible Values
4	Discovered software state	Whether the package was added to the catalog using the **wdsetsps** command, or the package was discovered by an Inventory scan.	**D**: discovered **–**: none
4	Hidden state	Whether the package is hidden following undoable install of a versionable package.	**H**: hidden **–**: none
5	Operational flag	The error condition of the package (if any)	**C**: changing **E**: error **–**: none

For example, following are some software package states:

I C - - -
An install has been committed.

I C U -
An install has been committed and can be undone

I P - B C
An install has been prepared and will be committed during the next reboot.

R C U - -
A remove has been committed, but it can be undone.

I C - - E
An install has been committed, but the software package is in error (the application may not work properly).

I C - D -
The package has been added using the **wdsetsps** command or discovered by an Inventory scan. For more information on signatures, refer to *IBM Tivoli Configuration Manager: User's Guide for Software Distribution*.

I C - H -
The package has been superseded by a versionable package installed in undoable mode.

I PP - -
The package has been installed in transactional mode and the backup package is also in transactional mode.

Synchronization and Discovery of Software Packages

Records of the states of software packages are stored both in the Inventory database and in catalogs on the endpoints. These two sets of details can become inconsistent with each other. One reason for inconsistency is that a disconnected CLI, for example, **wdinstsp** or **wdcmmtsp**, has been used to perform change management operations.

When you perform a change management operation on an endpoint, using a server command, for example **winstsp** or **wcommtsp**, information about changed states is retrieved and the Inventory database is updated.

You can also use the catalog synchronization command, **wsyncsp**, to reconcile the information on an adhoc basis. The **wsyncsp** command triggers a lenient and dataless distribution to the specified targets. By default, it returns information about changed software packages to the server. You can also specify an argument that causes the **wsyncsp** command to return all information about changed and unchanged software packages. This is useful, for example, if the cm_status information in the Inventory database is lost.

The **wsyncsp** command creates a log file on the Tivoli management region server, wsyncsp.log, where the results of the synchronization are recorded. When the synchronization causes a change in the state of a package that exists in the Tivoli database, this information is also written to the software package log for that package.

For a description of the **wsyncsp** command, see “wsyncsp” on page 261.

A related issue is the capability to bring externally installed software under the control of Software Distribution, as this allows a true picture of the situation on the endpoints to be recorded in the Inventory database.

This capability is provided by the discovered software package command, **wdsetsps**. When you execute this command on a disconnected target system, the specified software package is assigned a state of IC-D-, indicating that it is installed and ″discovered″.

Once the software package has been recorded in the endpoint catalog with the IC-D- state, you can use the **wsyncsp** command to record its information in the Inventory database.

There are limitations to the change management operations that can be used for a discovered software package. Only the following operations are available:
- Remove software package
- Force install software package

Software Package Processing Cycles

Only certain operations can be performed on a software package at each stage in its processing *cycle*. In addition, the software package state that results after an operation may vary based on the history of the package, that is, depending on what operations, such as install, have been previously performed on it.

See Chapter 3, “Using Commands,” on page 157 for more information on the syntax of the commands used during the processing cycle of a package.

Install and Remove Cycle

The following diagram illustrates the basic operations of the install (I) and remove (R) operations and the effects of these operations on the state of a software package, which is identified in its initial state as `Init`. For additional details on the software package state abbreviations used in this and the following sections, see “Software Package States” on page 150.

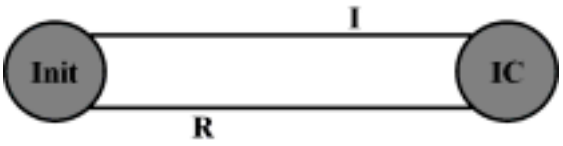


Figure 16. The install and remove cycle

Following are examples of the command line interface (CLI) syntax that produces these results. In each case, the first command listed is the server command and the second is the command that is run on a disconnected target system:

- Install (I)

```
winstsp @SPlabel subscribers...
wdinstsp SPB-file
```

- Remove (R)

```
wremovsp @SPlabel subscribers...
wdrmvsp SPlabel
```

Transactional Cycle

The following diagram illustrates running the install-transactional (It), rollback (B), commit (C), and remove-transactional (Rt) operations and the effects of these operations on the state of a software package.

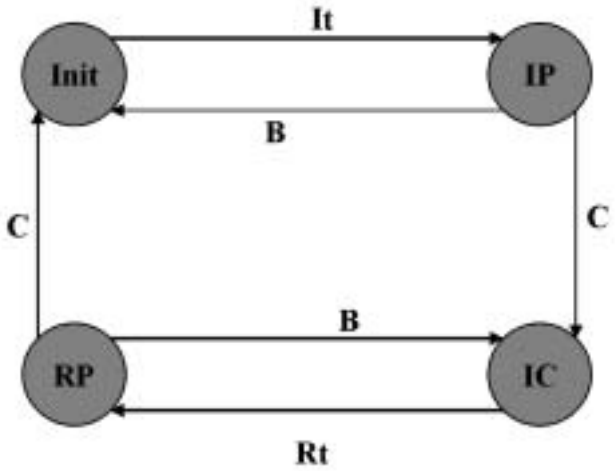


Figure 17. The transactional cycle

Following are examples of the command line interface (CLI) syntax that produces these results. In each case, the first command listed is the server command and the second is the command that is run on a disconnected target system:

- Install-transactional (It)

```
winstsp -ty @SPlabel subscribers...
wdinstsp -ty SPB-file
```

- Rollback (B)

```
wundosp @SPlabel subscribers...
wdundosp SPlabel
```

- Commit (C)

```
wcommtsp @SPlabel subscribers...
wdcmmtsp SPlabel
```

- Remove-transactional (Rt)

```
wremovsp -ty @SPlabel subscribers...
wdrmvsp -ty SPlabel
```

Undoable Cycle

The following diagram illustrates running the install-undoable (Iu), undo (U), accept (A), and remove-undoable (Ru) operations and the effects of these operations on the state of a software package.

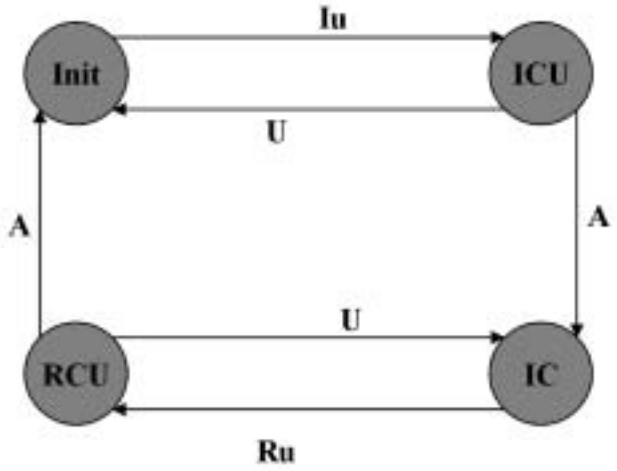


Figure 18. The undoable cycle

Following are examples of the command line interface (CLI) syntax that produces these results. In each case, the first command listed is the server command and the second is the command that is run on a disconnected target system:

- Install-undoable (Iu)

```
winstsp -uy @SPlabel subscribers...
wdinstsp -uy SPB-file
```

- Undo (U)

```
wundosp @SPlabel subscribers...
wdundosp SPlabel
```

- Accept (A)

```
wacctsp @SPlabel subscribers...
wdacptsp SPlabel
```

- Remove-undoable (Ru)

```
wremovsp -uy @SPlabel subscribers...
wdrmvsp -uy SPlabel
```

Undoable-in-Transactional Cycle

The following diagram illustrates running the install-undoable-in-transactional (Iut) and remove-undoable-in-transactional (Rut) operations and the effects of these operations on the state of a software package.

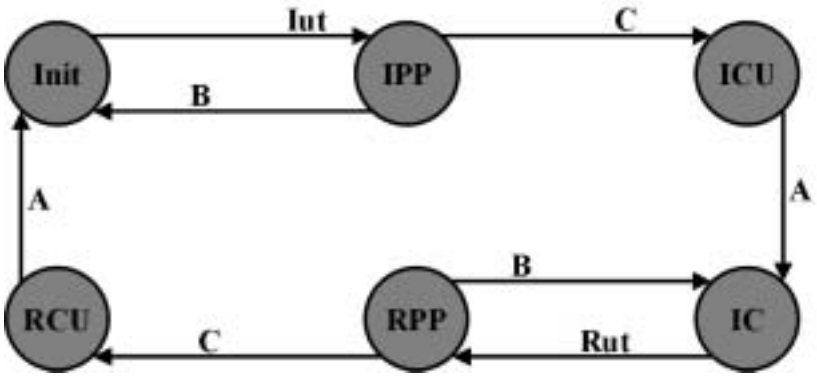


Figure 19. The undoable-in-transactional cycle

Following are examples of the command line interface (CLI) syntax that produces these results. In each case, the first command listed is the server command and the second is the command that is run on a disconnected target system:

- Install-undoable-in-transactional (Iut)

```
winstsp -ty -uu @SPlabel subscribers...
wdinstsp -ty -uu SPB-file
```

- Remove-undoable-in-transactional (Rut)

```
wremovsp -ty -uu @SPlabel subscribers...
wdrmvsp -ty -uu
```

Transactional-and-Undoable Cycle

The following diagram illustrates running the install-transactional-and-undoable (It&u) and remove-transactional-and-undoable (Rt&u) operations and the effects of these operations on the state of a software package:

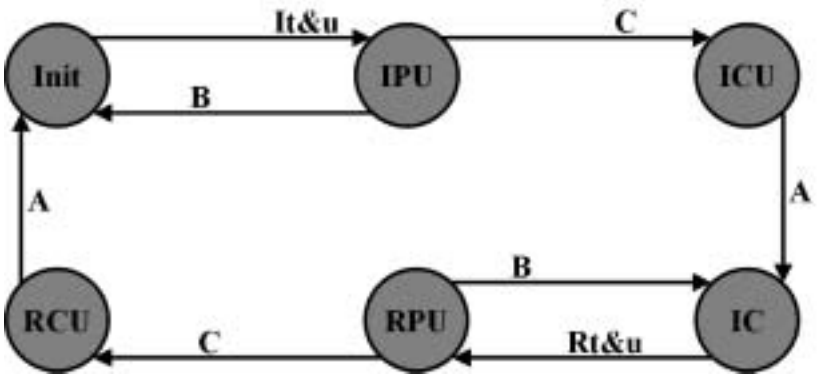


Figure 20. The transactional-and-undoable cycle

Following are examples of the command line interface (CLI) syntax that produces these results. In each case, the first command listed is the server command and the second is the command that is run on a disconnected target system:

- Install-transactional-and-undoable (It&u)

```
winstsp -ty -uy @SPlabel subscribers...
wdinstsp -ty -uy SPB-file
```

- Remove-transactional-and-undoable (Rt&u)

```
wremovsp -ty -uy @SPlabel subscribers...
wdrmvsp -ty -uy
```

Chapter 3. Using Commands

This chapter explains the use of the Tivoli command line interface (CLI). It includes the following sections:

- "Using the CLI"

  This section provides general information about using the CLI. It includes information relevant to all commands, for example, the conventions for defining command syntax and for referencing objects within commands.
- "Server Commands" on page 159

  This section includes descriptions of the commands that you can run on the server system.
- "Disconnected Target Commands" on page 281

  This section includes descriptions of the commands that you can run on a disconnected target system.
- "Preparation Site Commands" on page 298

  This section includes descriptions of the commands that you can use to prepare a software package.

Using the CLI

Commands enable you to perform system operations from a UNIX or PC command line instead of using the Tivoli desktop. Most Tivoli commands begin with the letter w; vowels are often omitted to shorten the name of a command. Commands are named using the *w+verb+object* syntax, which matches the way you might think of the action. For example, to install a software package, you use the **winstsp** command. To set the attributes for a software package object, you use the **wsetspat** command. In addition, those commands that are executed on disconnected target systems, that is, systems not connected to a Tivoli Management Region server, are named with the letter d in the second position, for example, **wdundosp**, which undoes a software package and is executed on a disconnected target system.

It is often necessary or convenient to invoke a Tivoli management application operation from the command line rather than from the desktop. For example:

- If you do not have access to a desktop, for example, if you are connected to the network by a modem.
- If you want to group several operations in a shell script or batch file.
- If an operation is not available using the desktop.
- If you prefer to invoke a command from a shell.

Command Line Syntax

This chapter uses the following special characters to define the syntax of commands:

[ ] Identifies optional attributes. Attributes not enclosed in brackets are required.

... Indicates that you can specify multiple values for the previous attribute.

| Indicates mutually exclusive information. You can use the attribute to the left of the separator or the attribute to its right. You cannot use both attributes in a single use of the command.

{ } Delimits a set of mutually exclusive attributes when one of the attributes is required. If the attributes are optional, they are enclosed in square brackets ([ ]).

\ Indicates that the syntax in an example wraps to the next line.

For example:

wdexptsp [[**–o**] [**–b**] **–f** *export_file*] {{**–s** | **–S**} *spname_path* | \
spname.version}

You must specify one attribute in each set of attributes that is delimited by the logical character (|) or enclosed in braces ({ }), and either the *spname_path* or *sp_name^version* variable. Attributes that are enclosed in brackets ([ ]), such as [–o], are optional.

Object References

When you reference an object in a command, the reference is not an absolute object reference like those used in programming. Instead, the reference is the label you gave the object when it was created. For example, if you created the Engineering policy region in the Tivoli environment, in a command, you refer to it as the Engineering object.

Two different forms of names that can be used with commands:
- Registered names
- Object paths

Tivoli commands support both naming schemes. Sometimes, you will find it more convenient to use one form over the other.

Registered Names

The key concept behind the name registry is a *registered name*. A registered name is a resource instance that is registered with the Tivoli name registry when it is created. Every resource has a name and is of some particular type. For example, a software package object has the name example_pk^1.0 and the type SoftwarePackage. An example of a registered name used as an attribute for the **wundosp** command is:

```
wundosp @SoftwarePackage:example_pk^1.0 @target1
```

The syntax for specifying a resource using a registered name is:

```
@type:name
```

where *type* is the resource type and *name* is the instance on which you wish to perform some operation. You must always specify the at sign (@) before a registered name. In the above example, an undo operation is performed on the software package example_pk^1.0 on the target system target1.

Note that the specification of the resource type, for example, SoftwarePackage, is optional if the resource name is unique, so the following syntax is also valid:

```
wundosp @example_pk^1.0 @target1
```

Object Paths

Object paths provide another way for you to specify an object name. They are similar to paths in file systems and can be relative or absolute. An absolute path begins with a slash character (/). A relative path begins with any character

including the special path components for the current directory (./) and for the parent directory (../). An example of an object path used as an attribute for the **winstsp** command follows:

```
winstsp /Regions/my-region/prf_manager/File_System.1.0
```

The syntax for specifying a resource using an object path is:

```
/Regions/ObjectPath/[type:]name
```

where */Regions/ObjectPath* is the path to the object, *type* is the resource type, and *name* is the particular instance on which you wish to perform some operation. Use the optional *type* specifier if the specified resource has the same name as another resource of a different type.

Getting Help on Commands

The following section lists Software Distribution commands, with syntax and descriptions of their functions. You can access these listings by using the **man** command on UNIX managed nodes. You can also access help information by typing on the command line of any platform the command name without specifying any syntax.

Server Commands

Table 48 lists the Software Distribution commands that you can run from the command line of a server in a distributed environment:

Table 48. Server commands

Server Command	Purpose	See page
waccptsp	Accepts a software package.	161
wcommtsp	Commits a software package.	167
wconvspo	Converts the format of a software package object: • From software package (not-built) to software package block (built). • From software package block (built) to software package (not-built).	173
wexpspo	Exports a software package object in software package definition file format.	175
wgetsnsp	Gets the list of software packages that are nested in a primary software package.	176
wgetspat	Gets the attributes for a software package object.	177
wgetspgs	Gets the program information associated with a software package object.	180
wgetspop	Gets the options for a software package object.	182
wimpspo	Imports a software package object.	184
winstsp	Installs a software package.	186
wldsp	Loads a software package on a depot.	196
wmapsigsp	Returns information about signatures.	199
wmsgbrowse	Allows you to browse and manage the Software Distribution message queue.	200
wmvspobj	Moves a software package object from the lost-n-found collection to a specified profile manager.	203

Table 48. Server commands (continued)

Server Command	Purpose	See page
wremovsp	Removes an installed software package.	205
wsdvers	Gets versioning information or a list of any later versions of the specified software package.	212
wsetsnsp	Specifies a list of software packages to be nested in a primary software package.	213
wsetspat	Sets the attributes for a software package object.	215
wsetspgs	Sets the program information associated with a software package object.	219
wsetspop	Sets the options for a software package object.	222
wsetsps	Synchronizes the status of the software package(s) installed on the endpoints with the database on the Tivoli server.	224
wspmvdata	Distributes or collects data files to and from endpoints.	229
wswdcfg	Displays or sets some source host keys.	246
wswdmgr	Enables and disables the integration between Software Distribution and Inventory, and between Software Distribution and IBM Tivoli Enterprise Console. This command also allows the user to define default values for distribution options in the specified policy regions.	252
wswsprim	Enables and disables the historical and change management status features of Software Distribution Historical Database.	260
wsyncsp	Synchronizes the status information on the server for software packages on specified endpoints with the real situation on the endpoints.	261
wuldsp	Unloads a software package from a depot.	264
wundosp	Undoes an installed software package.	267
wversp	Verifies an installed software package.	274
wwebgw	Enables you to list information about distributions on a resource gateway.	279

waccptsp

Accepts a software package. This command is run against an object for which the previous operation was performed with the undoable option. Running this command deletes any backup copies and changes the status of the package accordingly.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

waccptsp [**-i** [**–v**] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**-T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

waccptsp [**–I**] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**-T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

Description

The **waccptsp** command deletes any backup copies, so the previous operation can no longer be undone.

Options

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–v Specifies that verbose logging is to be used. If verbose logging is not used, any failure in the distribution is indicated by a short message, for example:

```
ep1 - Failed dependency check
```

If verbose logging is used, a full description of the reason for failure is logged.

Use of the verbose logging option causes a significant increase in the size of the log file. Therefore, the option must be used with caution. This option is available only if the **–i** argument is specified.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing an accept operation and the software package is already accepted on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–l *mdist2_token=value*

Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**accept**)*"*, where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, for example, if a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets

simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-X {none|first|middle|last|both}
Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the

series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*
Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last distribution has completed. If the user does not respond to the dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–T *subscribers_file*
Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the “wswdcfg” on page 246 command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...
The names of the target systems or profile manager where the software package is to be accepted. If no target systems are specified, the software package is accepted on all subscribers of the profile manager in which the software package resides. You can specify either one target host, multiple hosts, or none at all. If one or more subscribers are not valid, the operation

fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the "wswdcfg" on page 246 command.

Authorization

admin, senior, or super

Return Values

The **waccptsp** command returns one of the following:

0 Indicates that **waccptsp** started successfully.

-1 Indicates that **waccptsp** failed due to an error.

Examples

1. To accept a software package object called fsys_test^1.0 on the target1 and target2 target systems, enter the following command:

   ```
   waccptsp @fsys_test^1.0 @target1 @target2
   ```

2. To check the accept operation on a software package object called test^1.0 on the target1 and target2 target systems without actually performing it, enter the following command:

   ```
   waccptsp -v -i @test^1.0 @target1 @target2
   ```

See Also

- The **undo** option for **winstsp**, **wremovsp**, and **wundosp**
- **wmdist** in the *Tivoli Management Framework: Reference Manual*
- **wrpt** in the *Tivoli Management Framework: Reference Manual*

wcommtsp

Commits a software package.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wcommtsp [–i [**–v**]] [**–R** {*y* | *n*}] [**–c** |*y* | *n* | *o* | *r*] [**–l** *mdist2_token=value*...] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*] @[**Software Package**:]*spobj_name* [*subscribers*...]

wcommtsp [**–I**] [**–R** {*y* | *n*}] [**–c** |*y* | *n* | *o* | *r*] [**–l** *mdist2_token=value*...] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*] @[**Software Package**:]*spobj_name* [*subscribers*...]

Description

The wcommtsp command causes all updates done during the preparation phase (in transactional mode) to take effect.

Options

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–v Specifies that verbose logging is to be used. If verbose logging is not used, any failure in the distribution is indicated by a short message, for example:

```
ep1 - Failed dependency check
```

If verbose logging is used, a full description of the reason for failure is logged.

Use of the verbose logging option causes a significant increase in the size of the log file. Therefore, this option must be used with caution. This argument is only available if the **–i** argument is specified.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing a commit operation and the software package is already committed on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–c |*y* | *n* | *o* | *r*

Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see “Commit Operation” on page 148.

Note: Only the **–c n** option is available on the UNIX platform.

–l *mdist2_token=value*

Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**commit**)*"* where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop.This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to

trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-X {none | first | middle | last | both}

Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*

Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last distribution has completed. If the user does not respond to the dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–R *y | n*

Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–T *subscribers_file*

Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute pathnames. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the “wswdcfg” on page 246 command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...
The names of the target systems or profile managers where the software package is to be accepted. If no values are specified, the software package is committed to all subscribers of the profile manager in which the software package resides. You can specify either one target host, multiple hosts, or none at all. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the “wswdcfg” on page 246 command.

Authorization

admin, senior, or super

Return Values

The **wcommtsp** command returns one of the following:

0 Indicates that **wcommtsp** started successfully.

-1 Indicates that **wcommtsp** failed due to an error.

Examples

1. To run the commit operation with an automatic reboot for a software package called fsys_test^1.0 for target systems target1 and target2, enter the following command:

   ```
   wcommtsp -c r @fsys_test^1.0 @target1 @target2
   ```

2. To run the commit operation with an automatic reboot to be performed only if necessary for a software package called fsys_test^1.0 for target systems target1 and target2 ignoring any dependencies defined in the SPD file, enter the following command:

   ```
   wcommtsp -c o -R n @fsys_test^1.0 @target1 @target2
   ```

See Also

- The **transactional** option for **winstsp**, **wremovsp**, and **wundosp**
- **wmdist** in the *Tivoli Management Framework: Reference Manual*
- **wrpt** in the *Tivoli Management Framework: Reference Manual*

wconvspo

Converts the format of a software package object from software package (not-built) to software package block (built), or vice versa.

Syntax

wconvspo –t build [**–o**] **–p** *dest_sppath spobj_name*

wconvspo –d *download_path spobj_name*

Description

This command can be used to convert only software package objects that already exist in the object database and that have been created or imported with the **wimpspo** command. The first statement shows how to convert a software package to a software package block, the second statement shows how to convert a software package block to a software package. When converting from not-built to built format, all files must be located on the selected source host, otherwise the operation fails.

Using the command with the **–d** argument, you can download the contents of the software package to a specified location. You can then make changes and rebuild the package.

Note: The size of a software package block cannot exceed two gigabytes.

If you are using the **–d** argument to convert a software package block to an not-built format and save the contents, ensure that the specified location has sufficient space for the contents of the built software package. If there is insufficient space to save the files, the convert fails.

Options

–t build
Specifies to build a software package block. If not specified, the default is to convert a software package block to a software package.

–o Specifies to overwrite an existing *dest_sppath* file on the target. This option is valid only if the destination format is software package block.

–p *dest_sppath*
Path of the software package block on the source host. This parameter can only be used if you want to convert a software package to a software package block.

–d *download_sppath*
Path to which the contents of a built software package are to be downloaded. If you specify this argument, the contents of the software package are downloaded to the specified location, and the source paths in the software package are updated to point to this directory.

spobj_name
Name and version of the software package object to be converted. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, or super

Return Values

The **wconvspo** command returns one of the following:

0 Indicates that **wconvspo** started successfully.

-1 Indicates that **wconvspo** failed due to an error.

Examples

1. To convert the format of a software package called fsys_test^1.0 to a software package block called d:\testdir\package.spb, enter the following command:

   ```
   wconvspo -t build -p d:\testdir\package.spb @fsys_test^1.0
   ```

2. To convert a software package block to a software package and save the contents to the staging directory c:\staging, enter the following command:

   ```
   wconvspo -d c:\staging @name^version
   ```

See Also

wimpspo, **wexpspo, wdubldsp**

wexpspo

Exports a software package object in software package definition file format on a managed node.

Syntax

wexpspo [**–o**] [**–f** [*@managed_node:*]*export_file*] @[**SoftwarePackage:**]*spobj_name*

Description

All keywords in the software package, along with their default values, are exported.

To export to a remote managed node, you must specify its fully qualified path.

Note: Any comment lines (beginning with the number sign [#]) that were included in the original SPD file are not preserved after running the **wexpspo** command.

Options

–o Specifies to overwrite an existing file on the target. If not specified and *export_file* already exists, an error results.

–f [*@managed_node:*]*export_file***
Specifies to export the software package to a file (default standard output) on a managed node.

@[SoftwarePackage:]*spobj_name*
Name and version of the software package object. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, super, or user

Note: On UNIX platforms, when you export a software package to an SPD format, you have read and write privileges to the package, while all other users only have read privileges. For this reason, the owner of the exported SPD file should be the same user who exported the package.

Return Values

The **wexpspo** command returns one of the following:

0 Indicates that **wexpspo** started successfully.

-1 Indicates that **wexpspo** failed due to an error.

Examples

1. To export the software package called fsys_test^1.0 to the SPD file format d:\testdir\fsystem.exp on the managed node mng_node, while overwriting the existing target file, enter the following command:

   ```
   wexpspo -o -f @mng_node:d:\testdir\fsystem.exp @fsys_test^1.0
   ```

2. To export the software package called fsys_test^1.0 to standard output, enter the following command:

   ```
   wexpspo @fsys_test^1.0
   ```

See Also

wimpspo, **wconvspo**

wgetsnsp

Gets the list of software packages nested in a primary software package.

Syntax

wgetsnsp [**–l**] *spobj_name*

Description

This command retrieves the list of software packages that are nested in a primary software package.

Options

–l Specifies to retrieve the list of all nested software packages in the primary package. This list will also contain any software packages that are nested in turn in other nested software packages.

spobj_name
Specifies the registered name and version of the software package whose nested software packages are returned by this command. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, super, or user

Return Values

The **wgetsnsp** command returns one of the following:

0 Indicates that **wgetsnsp** started successfully.

-1 Indicates that **wgetsnsp** failed due to an error.

Examples

1. To retrieve the list of all software packages nested in the software package called fsys_test.1.0, enter the following:

   ```
   wgetsnsp @fsys_test.1.0
   ```

2. To retrieve the list of all software packages nested in the software package called fsys_test.1.0 including any software packages that are nested in turn in other nested software packages, enter the following:

   ```
   wgetsnsp -l @fsys_test.1.0
   ```

See Also

- **wsetsnsp**
- “Nesting Software Packages” on page 16

wgetspat

Gets the attributes for a software package object.

Syntax

wgetspat {**–b | -c | -h | -l | -m | -M | -o | -p | -s | -t | -u | -v | -w**} *spobj_name*

Description

This command retrieves the attributes for a specified software package object. The attributes that can be retrieved are listed under the Arguments heading below.

Options

–b Gets the value of the move_removing_host attribute, which moves the software package to the lost-n-found collection if the log host or the source host of the software package is removed. Possible values are **true** or **false**.

–c Gets the value of the committable attribute, which indicates if you can execute a committable operation on the object. Possible values are:

- **y** The installation and the remove operations that you perform on this software package object must be transactional.
- **n** The installation and the remove operations that you perform on this software package object cannot be performed in transactional mode.
- **o** The installation and the remove operations that you perform on this software package object can be either transactional or not transactional.

–h Gets the value of the source host attribute src_host, which is the object reference of the managed node that is the source host for the file package.

–l Gets the value of the lenient_distribution attribute. Possible values are:

- **true** Allows distributions and removals from the command line to any endpoint or profile manager, even if that endpoint or profile manager is not currently a subscriber to the profile manager of the software package (or to any of its subscribing profile managers).
- **false** Allows distributions and removals from the command line to only endpoints or profile managers that are currently subscribers to the profile manager of the software package (or to any of its subscribing profile managers).

–m Gets the default server mode. Possible values are as follows:

- **all** Installs all the objects in the package.
- **src** Installs only those files on the source that have a modification time later than the time of the last successful installation.
- **repair** Installs only those source objects that have changed on the target system or that are not present on the target system, to make the target system objects consistent with the source system objects.
- **check** Checks the operation only, without submitting it.
- **force** Forces the operation.
- **ignore** Executes the operation on only successfully checked target systems and ignores the target systems for which the check is unsuccessful.

preview
Lists all the actions to be executed on the specified subscribers.

–M Gets the default mode for change management operation. The possible values follow:
- **transactional**
- **not_transactional**
- **prefer_not_transactional**
- **auto_commit**
- **undoable**
- **prefer_undo**
- **auto_accept**
- **undoable_in_transactional**
- **prefer_not_reboot**
- **during_reboot**
- **auto_reboot**
- **force**

See Chapter 2, “Performing Change Management Operations,” on page 143 for more information about these modes.

–o Gets the default change management operation to be performed on the object, which is one of the following:
- **install**
- **remove**
- **commit**
- **accept**
- **undo**

See “Types of Change Management Operations” on page 144 for more information about these operations.

–p Gets the source path where the corresponding software package block is located.

–s Gets the value of the check_no_src_host attribute, which causes a check of the existence of all files on the source host as defined in the software package. Possible values are **true** or **false**.

–t Gets the format for maintaining the software package object file in the object repository, either **not_built** (software package) or **built** (software package block). The default value is **not_built**.

–u Gets the setting for whether you can execute an undoable operation on the object. Possible values are:

y The installation and the remove operations that you perform on this software package object must be undoable.

n The installation and the remove operations that you perform on this software package object cannot be performed in undoable mode.

o The installation and the remove operations that you perform on this software package object can be either undoable or not undoable.

–v Gets the value of the web_view_mode attribute, which specifies access permissions to the package when using the Web Interface feature. Possible values are **hidden** (the default), **subscriber**, or **public**.

–w Gets the value of the staging_area attribute, which is the working directory of the distribution server.

spobj_name
The name and version of the software package whose attributes are returned by this command. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization
admin, senior, super, or user

Return Values
The **wgetspat** command returns one of the following:

0 Indicates that **wgetspat** started successfully.

-1 Indicates that **wgetspat** failed due to an error.

Examples
1. To get the source host attribute for an object called fsys_test^1.0, enter the following:

   ```
   wgetspat -h @fsys_test^1.0
   ```

2. To get the default server mode for an object called fsys_test^1.0, enter the following:

   ```
   wgetspat -m @fsys_test^1.0
   ```

See Also
wsetspat

wgetspgs

Gets the program information associated with a software package object.

Syntax

wgetspgs {**–a**|**-b**|**-i**|**-j**|**-p**|**-P**|**-s**|**-u**|**-U**|**-l**|**-L**|**-c**|**-C**|**-t**|**-T**} *spobj_name*

Description

The program information you can retrieve with this command is listed below.

Options

–a Gets the value for the `after_prog_path` attribute, which is the path of the program to be run on the source host after the build is completed.

–b Gets the value of the `before_prog_path` attribute, which is the path of the program to be run on the source host before the build is begun.

–i Gets the value of the `after_input_path` attribute, which specifies the input parameters for the `after_prog_path` attribute. This value, although set, has no meaning unless the **–a** (after_prog_path) attribute is defined.

–j Gets the value of the `before_input_path` attribute, which specifies the input parameters for the `before_prog_path` attribute. This value, although set, has no meaning unless the**–b** (before_prog_path) attribute is defined.

–p Gets the list of environment variables for the before program.

–P Gets the list of environment variables for the after program.

–s Gets the value of the `before_skip_non_zero` attribute. The value **true** indicates that distribution is skipped if before_prog_path exits with a non-zero exit code; **false** indicates that it is not skipped.

–u Gets the value of the `after_as_uid` attribute, which sets the UNIX user ID under which to run after_prog_path.

–U Gets the value of the `before_as_uid` attribute, which sets the UNIX user ID under which to run before_prog_path.

-l Gets the value of the `lcf_before_prog_path` attribute, which is the pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run.

-L Gets the value of the `lcf_after_prog_path` attribute, which is the pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run.

-c Gets the value of the `lcf_before_prog_args` attribute, which is one or more optional arguments passed to the program in addition to the default ones. Separate arguments with Separate arguments with a blank space. You can also use variables to specify arguments.

-C Gets the value of the `lcf_after_prog_args` attribute, which is one or more optional arguments passed to the program in addition to the default ones. Separate arguments with a blank space. You can also use variables to specify arguments.

-t Gets the value of the `lcf_before_prog_timeout` attribute, which is The

time, expressed in seconds or the value -1, to wait for the completion of the before program. See “Format of the execute_user_program Stanza” on page 118 for detailed information.

-T Gets the value of the `lcf_after_prog_timeout` attribute, which is The time, expressed in seconds or the value -1,in seconds or the value -1, to wait for the completion of the after program. See “Format of the execute_user_program Stanza” on page 118 for detailed information.

spobj_name
The name and version of the software package for which the program information is returned by this command. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization
admin, senior, super, or user

Return Values
The **wgetspgs** command returns one of the following:

0 Indicates that **wgetspgs** started successfully.

-1 Indicates that **wgetspgs** failed due to an error.

Examples
1. To get the value of the `after_as_uid` attribute for the software package object called fsys_test^1.0, enter the following command:

   ```
   wgetspgs -u @fsys_test^1.0
   ```

2. To get the value of the `before_input_path` attribute for the software package object called fsys_test^1.0, enter the following command:

   ```
   wgetspgs -j @fsys_test^1.0
   ```

See Also
wsetspgs, **winstsp**, **wimpspo**

wgetspop

Gets the options for a software package object.

Syntax

wgetspop {**–g | -h | -j | -L | -m | -P- | -r | -u**} *spobj_name*

Description

The options you can retrieve with this command are listed below.

Options

–g Gets the value of the log_gid attribute, which is the UNIX group ID of the log file.

–h Gets the value of the log_host attribute, which specifies the label of the managed node where the log file is generated.

–j Gets the value of the log_mode attribute, which is the UNIX file mode of the generated log file, displayed in octal format.

–L Gets the value of the log_path attribute, which is the path and file name of the generated log file.

–m Gets the value of the mail_id attribute, which is the e-mail address for notification. To specify multiple user IDs, separate each ID with a comma (,).

–P Gets the value of the post_notice attribute, which specifies whether to send a notice. The possible values are **true** or **false**.

-r Gets the value of the no_chk_on_rm attribute, which specifies whether a check is performed on the whole Tivoli Management region when the specified software package is deleted to verify whether it is nested in other software packages. The possible values are **true** or **false**. The default value is **false**.

–u Gets the value of the log_uid attribute, which is the UNIX user ID of the log file.

spobj_name
The name and version of the software package whose options are returned by this operation. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, super, or user

Return Values

The **wgetspop** command returns one of the following:

0 Indicates that **wgetspop** started successfully.

-1 Indicates that **wgetspop** failed due to an error.

Examples

1. To get the log file name for a software package object called fsys_test^1.0, enter the following command:

   ```
   wgetspop -L @fsys_test^1.0
   ```

2. To get the e-mail address for notification for a software package object called fsys_test^1.0, enter the following command:

   ```
   wgetspop -m @fsys_test^1.0
   ```

See Also

wsetspop

wimpspo

Creates or imports a software package object that is in the object database on the server.

Syntax

wimpspo{**–f** [*@input_node:*] *inputfile_path* | **–r** } [**–c** *profile_manager*] [**–h** *src_host*] [**–t build** [**–o**] **–p** *dest_sppath*] [*spobj_name*]

Description

The **wimpspo** command takes the software package from standard input or from an input file, then creates a software package object on the server database.

Note: Any comment lines (beginning with the number sign [#]) that were included in the original SPD file are not preserved after running the **wimpspo** command.

If importing in the built format, the size of the software package block cannot exceed two gigabytes.

Options

–c *profile_manager*
Specifies the profile manager where the new software package will be created or where an existing package will be overwritten. This attribute is mandatory for a new profile. If you do not specify this option, the existing software package with the name specified in *sp_name* will be overwritten.

–h *src_host*
Specifies the source host where the files in the software package block are obtained and where the software package block is stored. The source host can be any of the available managed nodes where the source host is installed. The default value of *src_host* is the name of the Tivoli Management Region server.

–f[*input_node:***]***inputfile_path*
Specifies to take the input file from the managed node or endpoint specified in the path, rather than from standard input. This value should normally be expressed in terms of an absolute path. However, if the input file is present on the local machine and you don't specify a value for *input_node*, you can supply a relative path.

–r Indicates that the data to be imported is from standard input, rather than an input file. If this argument is used, the command awaits the input of data. Press CTRL-C to indicate that you have finished entering data.

–t build
Specifies to import a file in software package block format. If not specified, the default is to import a software package.

–o Specifies to overwrite an existing *dest_sppath* destination file. This attribute cannot be used if the destination file is a software package.

–p *dest_sppath*
Path of the software package block on the source host. This attribute is required only if the destination format is software package block. If the destination format is software package, no path is needed because the software package data is stored as an attribute of the software package object.

sp_name

Object path or registered name of the software package object. The name and version must be specified in the form *@sp_name^version*. If not specified, the value of the keywords ″name″ and ″version″ in the input file are used.

Authorization

- To create a new software package: senior or super.
- To import an existing software package: admin, senior, o super

Return Values

The **wimpspo** command returns one of the following:

0 Indicates that **wimpspo** started successfully.

-1 Indicates that **wimpspo** failed due to an error.

Examples

1. To import a software package into a profile manager, enter the following command. In this example, the software package, c:\test\fsys.spf, is imported into the profile manager, prf_mgr, as a software package object in the software package format (not-built).

   ```
   wimpspo -c @prf_mgr -f c:\test\fsys.spf
   ```

2. To import a software package block into a profile manager, enter the following command. In this example, the software package, c:\test\fsys.spf, is imported into the profile manager, prf_mgr, as a software package object in the software package block format (built).

   ```
   wimpspo -c @prf_mgr -f c:\test\fsys.spf \
   -t build -p c:\test\fsys.spb
   ```

3. To import a software package from a managed node into a profile manager, enter the following command. In this example, the software package, c:\test\fsys.spf, is imported from the managed node, Man_Node, into the profile manager, prf_mgr, as a software package object in the software package format (not-built). This also overwrites the existing file.

   ```
   wimpspo -c @prf_mgr -f @Man_Node:c:\test\fsys.spf
   ```

4. Enter the following command to import the software package, c:\test\fsys.spf, to a software package block that already exists. This also overwrites the existing built file.

   ```
   wimpspo -f c:\test\fsys.spf -t build -o \
   -p c:\test\fsys.spb
   ```

5. To re-import the SPD file, c:\test\test.spd, to the software package @test^1.0 that has already been imported into the object database (and to overwrite the existing software package), enter the following command:

   ```
   wimpspo -f c:\test\test.spd @test^1.0
   ```

See Also

wconvspo, **wexpspo**

winstsp

Installs a software package on a selected group of target subscribers.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*] [[[**–m** *a*|*s*|*r*] | [**–D** *variable=value...*] [**–f** | **–I** | **–i** [**–v**]] [**–R** {*y*|*n*}] [**–t** *y*|*n*|*o* [**–c** *y*|*n*|*o*|*r*]] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file...*] @[**SoftwarePackage:**] *spobj_name* [*subscribers...*]

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*] [[**–p**] [**–m** *r*] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–t** *y*|*o* [**–c** *y*|*n*|*o*|*r*]] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**] *spobj_name subscribers...*

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*] [[**–p**] [**–m s**] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–t** *y*|*o* [**–c** *y*|*n*|*o*|*r*]] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file ...*] @[**SoftwarePackage:**] *spobj_name* [*subscribers...*]

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*] [[**–p**] [**–m** *r*] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–t** *y*|*n*|*o*] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**] *spobj_name subscribers...*

winstsp *[***-d** @[**SoftwarePackage:**] *spobj_name*] [[**–p**] [**–m** *s*] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}][**–t** *y*|*n*|*o*] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file ...*] @[**SoftwarePackage:**] *spobj_name* [*subscribers...*]

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*][[**–p**] [**–m** *r*] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–t** *y*|*o* [**–c** *y*|*n*|*o*|*r*]] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** from_depot=n] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**] *spobj_name*.spb *subscribers...*

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*][[**–p**] [**–m** *r*] | [**–D** *variable=value...*]] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–t** *y*|*n*|*o*] [**–u** *y*|*n*|*o*|*u* [**–a**]] [**–l** from_depot=n] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**] *spobj_name*.spb *subscribers...*

winstsp [**-d** @[**SoftwarePackage:**] *spobj_name*] [**–m** *a*|*s*] [**–D** *variable=value...*] [**–f** | **–I** | **–i** [**–v**] [**–R** {*y*|*n*}] [**–l** *mdist2_token=value...*]]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file ...*] @[**SoftwarePackage:**] *spobj_name resource_group...*

Description

This command performs an install on the target system of the actions described in a software package.

Options

For options specific to device objects, see “Device Object Options” on page 194.

–d *spobj_name*
Specifies the name and version of the base package to which a delta install operation is to be applied. If this argument is specified, the install operation uses byte-level differencing to apply changes to an existing package, rather than using a replacement package. If any of the installed files to be reconstructed are not found on the target, or have been modified, the delta installation fails. Read-only files, if any, are overwritten. To apply the delta installation, the base and the version packages must have the same nested structure. In addition, if the base and the version packages use the same file in the same directory, the packages must be in software package block format, otherwise the whole file is distributed. Byte-level differencing uses dependency checking to verify that the base package has been installed. The base package must be in the IC or ICU state. If you specify **–R n**, no checks are done on the base package. For a detailed explanation of how byte level differencing works, see “Byte-level Differencing” on page 148.

–p Returns to the log file on the server a list of actions that would be carried out if you performed the operation. The operation is not actually carried out. A check is performed on the target and the list of files to be repaired or a list of source files that have been modified is returned to the log file. This attribute can be used in conjunction with the **–m s** (preview source) or **–m r** (preview repair) attributes. The list of subscribers must be specified for the **–p –m r** (preview repair) operation only.

–m *a | s | r*
Specifies the distribution mode as follows:

a **All**: installs all the files in the software package. This is the default.

s **Source**: installs only those source host files that have been modified since the last successful distribution to the target system. This mode is applicable only to unbuilt software packages.

r **Repair:** Installs the following:
- The source objects that have been corrupted, or modified since the time of the last successful installation, or are not present on the target. This makes the target objects consistent with the source objects.
- The objects and actions on the target that have been changed or corrupted since the time of the last successful installation.

Note: The **s** option cannot be applied to built software packages (software package blocks).

The **r** option applies only to software packages with the final status of I C – – – (installed, committed) or I C – – E (installed, committed, error). The **r** option can be used for built software packages, but only if the MDist 2 **from_depot** argument is set to **n**. The **s** and **r** options cannot be used if you are performing a transactional installation (**t** transactional)

–D *variable=value*
Defines the value of a variable used in the software package, to add or

override existing variables. When specifying multiple variables, repeat the **–D** attribute before each *variable=value*. Note that these variables can be resolved only on the endpoint.

–f Specifies forcing the operation, regardless of the state of the package on the target system.

If dependency checking is defined and turned on, this argument cannot be used to force installation of a package when the dependency condition is not true. If you want to override the dependency check, you must use the **–R** argument.

If the package is versionable, the version checks are made on the target even when this option is selected. If the version checks fail, the operation fails. Possible version check failures are:

- A more recent version of the package is already installed.
- The base version for a patch is not installed.

See "Software Package Version Checking" on page 4.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing an install operation and the software package is already installed on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–v Specifies that verbose logging is to be used. If verbose logging is not used, any failure in the distribution is indicated by a short message, for example:

```
ep1 - Failed dependency check
```

If verbose logging is used, a full description of the reason for failure is logged.

Use of the verbose logging option causes a significant increase in the size of the log file. Therefore, this option must be used with caution. This argument is only available if the **–i** argument is specified.

–R *y* | *n*

Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–t *y* | *n* | *o*

Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y* | *n* | *o* | *r*

Specifies the reboot options for the commit operations: **n** (not-in-a-reboot),

which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see "Commit Operation" on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options.

–u *y | n | o | u*
Specifies the undoable option: **n** (no, which is the default), **y** (yes), **o** (preferably), or **u** (undoable-in-transactional).

–a Specifies that the operation should be automatically accepted. This attribute is used in conjunction with the **–u** (undo) attribute.

–l *mdist2_token=value*
Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**install**)" where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

disposable
Specifies if data must be removed from the repeater after distribution. Valid values are **y** (yes) or **n** (no). The default value is

n for a software package block (built format). This attribute is not available for software packages (not-built format).

Note: For a software package block, disposable and from_depot cannot both be set to **y**.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

from_depot
Specifies that the software package to be installed resides on the repeater depot, rather than on the source host. In this case, the package was previously loaded on the gateway using the **wldsp** command. Valid values are **y** (yes) or **n** (no). The default value for a software package block (built format) is **n**. This attribute is not available for software packages (not-built format).

Note: For a software package block, disposable and from_depot attributes cannot both be set to **y**.

from_fileserver
Specifies that the images referenced in the software package are to be retrieved from a file server. File servers must be configured if this argument is used.

Notes:

1. You cannot specify the **from_fileserver** and **from_depot** attributes in the same command. However, you can specify the **from_fileserver** and **from_cd** attributes as alternative locations for the images.
2. Before you can use MDist 2 to install a distribution from a file server, the following file must exist on the target system: $LCF_DATDIR/remote.dir. Refer to *Tivoli Management Framework: User's Guide* for more information about the remote.dir file.
3. This option is not supported for Novell Netware endpoints.

from_cd
Specifies that the images referenced in the software package are to be retrieved from the CD. If you use this argument, you are prompted to insert the CD.

Note: You cannot specify **from_cd** and **from_depot** in the same command. However, you can specify **from_cd** and **from_fileserver** as alternative locations for the software package.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet installed the package.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not installed the package by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
Indicates whether the installation on mobile targets is to be hidden. Non-hidden installations on mobile targets can be deferred. Hidden installations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y** indicates that the

distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n** indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-X {none | first | middle | last | both}
Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*
Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last distribution has completed. If the user does not respond to the dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–T *subscribers_file*
Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
The name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...
The names of the target systems or profile manager to receive the software package. If no target systems are specified, the software package is distributed to all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Device Object Options: When you use the **winstsp** command on devices, you can use only the following tokens for the **–l** option. Descriptions and valid values are listed under “Options” on page 187.

deadline

default_timeout

enable_notification

execute_timeout

is_multicast

label: only the **install** operation is supported.

notify_interval

priority

retry_unicast

roam_endpoints

send_timeout

user_notification

wake_on_lan

Authorization

admin, senior, or super

Return Values

The **winstsp** command returns one of the following:

0 Indicates that **winstsp** started successfully.

-1 Indicates that **winstsp** failed due to an error.

Examples

1. To install software package object fsys^1.0 in transactional mode on the subscribers target1 and target2, enter the following command:

   ```
   winstsp -ty @fsys^1.0 @target1 @target2
   ```

2. To install software package object fsys^1.0 in undoable mode with auto-accept on the subscribers target1 and target2, enter the following command:

   ```
   winstsp -uy -a @fsys^1.0 @target1 @target2
   ```

3. To install software package object fsys^1.0 in repair mode on the subscribers target1 and target2, enter the following command:

   ```
   winstsp -m r @fsys^1.0 @target1 @target2
   ```

4. To install software package object fsys^1.0 in preview mode (without actually installing it) on the subscribers target1 and target2, enter the following command:

   ```
   winstsp -p @fsys^1.0 @target1 @target2
   ```

5. To install a delta file of changes to base software package fsys^1.0 on the subscribers target1 and target2, enter the following command:

   ```
   winstsp -d @fsys^1.0 @fsyschange^1 @target1 @target2
   ```

6. To install software package object fsys^1.0 on the subscribers target1, target2, and target3, allowing installation to proceed on targets that do not fail cm status checks, enter the following command:

   ```
   winstsp -I @fsys^1.0 @target1 @target2 @target3
   ```

   The inclusion of the –I argument has the effect that even if one of the targets fails, the installation can proceed on the other targets that do not fail the checks. Without this argument, a validation failure for one target causes the command to fail on all targets.

7. To install software package DevicePkg.1 on the resource group DeviceGroup, enter the following command:

   ```
   winstsp @SoftwarePackage:DevicePkg.1 @ResourceGroup:DeviceGroup
   ```

8. To install software package UserPkg.1 on the resource group UserGroup, enter the following command:

   ```
   winstsp @SoftwarePackage:UserPkg.1 @ResourceGroup:UserGroup
   ```

See Also

- **wremovsp**
- **waccptsp**
- **wundosp**
- **wcommtsp**
- **wldsp**
- **wmdist** in the *Tivoli Management Framework: Reference Manual*
- **trace level** in the *Tivoli Management Framework: Reference Manual* (if a repeater is configured in your network)
- **wwebgw** for distributions to resource groups

wldsp

Loads a software package on a selected group of target subscribers.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage:**]*spobj_name*

wldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [@[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage:**]*spobj_name*

wldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage**:]*spobj_name* [*subscribers...*]

Description

Options

–d *spobj_name*
: The name and version of the base package to which a delta install operation is to be applied. If this argument is specified, the software package includes changes to be applied to an existing package, rather than a replacement.

–f
: Specifies that the load operation must be forced regardless of the state of the package on the repeater.

–i
: Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–I
: Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing a load operation and the software package is already loaded on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–l *mdist2_token=value*
: Specifies the distribution options, as follows:

  label
  : Specifies a description string for the distribution. The default value is the string *"name.version* (**load**)*"* where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

depot_image_dir
Directory on the depot where the product images are to be stored. The contents of this directory can be copied to a CD ROM or to a file server, which can then be used as the source for an installation. See “winstsp” on page 186

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

–T *subscribers_file*

Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name

Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...

The names of the target systems or profile manager where the software package is to be loaded. If no target systems are specified, the software package is distributed to all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wldsp** command returns one of the following:

0 Indicates that **wldsp** started successfully.

-1 Indicates that **wldsp** failed due to an error.

Examples

1. To load software package @mypackage on the profile manager myprofile, enter the following command:

   ```
   wldsp @mypackage @myprofile
   ```

2. To load a delta software package @mychanges, which is to be applied to the base package @mypackage, on the repeater depot mygateway, enter the following command:

   ```
   wldsp -d @mypackage @mychanges @mygateway
   ```

See Also

- **wuldsp**
- **wrpt** in the *Tivoli Management Framework: Reference Manual*
- **wmdist** in the *Tivoli Management Framework: Reference Manual*
- **wdepot** in the *Tivoli Management Framework: Reference Manual*

wmapsigsp

Returns information about signatures used to identify software packages.

Syntax

wmapsigsp -**a**

wmapsigsp -**n** *sp_name*

wmapsigsp -**p** [-**v**]

Description

Information about signatures is available only if the integration with Inventory has previously been enabled using the **wswdmgr** and **wsetinvswd** commands.

Options

-a Displays all signatures.

-n *sp_name*
Displays all signatures for the specified software package.

-p Physically removes from the database all signatures which have been previously marked as deleted.

-v Returns name and size of the deleted signatures. This option can only be specified in conjunction with the **-p** option

Authorization

senior, or super

Return Values

The **wmapsigsp** command returns one of the following:

0 Indicates that **wmapsigsp** started successfully.

-1 Indicates that **wmapsigsp** failed due to an error.

Examples

1. To display all signatures, enter the following command:

   ```
   wmapsigsp -a
   ```

2. To display all signatures for software package my_package^1.0, enter the following command:

   ```
   wmapsigsp -n my_package^1.0
   ```

3. To delete all signatures and display their name and size, enter the following command:

   ```
   wmapsigsp -p -v
   ```

   The following output is displayed:

   ```
   Mapping for signature 'chkorg2452.exe (754)' and software package
   'test^1' has been purged.
   ```

See Also

- **wsetinvswd** in the *User's Guide for Inventory*
- **wswdmgr**

wmsgbrowse

Allows you to browse and manage the Software Distribution message queue. This command is to be used for problem determination only.

Syntax

wmsgbrowse -s

wmsgbrowse -a [**-r** *format*] [**-e** |**-d** [**-f**]]

wmsgbrowse *filter*...[**-r** *format*] [**-e** |**-d** [**-f**]]

Description

Returns general information, such as state and number of distribution messages present in the queue, total queue size and other relevant information. Allows you to browse the queue and retrieve messages based on their distribution ID, name, target, or logger, to trace reports within messages, and to delete a defined report or a whole message. Messages to be delivered are stored in the following directory: $BINDIR/../swdis/work/messages. Browsing and editing operations might interfere with Software Distribution normal workflow, as the queue is locked while reading or writing, so these operations should be as short as possible.

Following are descriptions of the available loggers:

standard
Sends data to the five standard listeners (file, DB, Tivoli Event Console, notice, mail).

gateway
Sends data to the standard listeners, with the exception of the DB listener. This logger is usually created on the server to handle data sent by a repeater, for example when a cancel operation is required by the user, or when a failure occurs due to a full disk on the repeater.

bulk_data
Sends information available in string format to a file, for example, the standard output error received when executing a program.

delete_cm_entries
Deletes all rows related to a specific endpoint from the SD_INST table in the Inventory database. This logger is used generally when the catalog on the endpoint is missing or has been deleted, and as a result, no software package is installed on the endpoint.

delete_cmstatus_entry
Manages the SD_INST table in connection with a specific endpoint.

add_cmstatus_entry
Manages the SD_INST table in connection with a specific endpoint.

edit_cmstatus_entry
Manages the SD_INST table in connection with a specific endpoint.

external
Sends a set of data to external listeners, such as Activity Planner.

mdist2_db
Sends error messages to the MDist 2 database.

inventory
Updates the cm_status table after an Inventory scan, if the Inventory plug-in is installed.

web_sync_db
Synchronizes the database after a web operation.

publish
Sends the result of a publish or withdraw web operation to a log file.

Options

-s Returns the list of loggers (log file, Tivoli Event Console, mail, and so on) and filter keywords.

-a Returns all undelivered messages still present in the Software Distribution queue. If no filter keyword is specified, some general queue statistics are written to standard output, containing number of messages, value of configuration parameters concerning the Software Distribution message queue, and so on.

-r *format*
Allows you to format the output of the command. You can specify the sequence of fields displayed in the output by entering the related filter keywords between double quotation marks and percentage symbols. Allowed formatting characters are \n, to enter a carriage return, and \t, to enter a tab space.

-e Returns all messages delivered by Software Distribution.

-d Deletes reports matching the filter criteria and displays the names of deleted reports to standard output.

-f Forces deletion of specified reports without requiring confirmation. This option can only be specified in conjunction with the **-d** option.

filter Specifies the filter to be applied to the command output. At least one of the following must be specified:

-i *distribution_id*
Filters messages to be displayed based on their MDist 2 distribution ID. Only messages with a distribution ID are returned.

-t *target_name*
Filters messages to be displayed based on the endpoint name.

-n *logger_name*
Filters messages to be displayed based on the logger name.

Authorization

admin, senior, or super

Return Values

The **wmsgbrowse** command returns one of the following:

0 Indicates that **wmsgbrowse** started successfully.

-1 Indicates that **wmsgbrowse** failed due to an error.

Examples

1. To display all undelivered messages still present in the Software Distribution message queue, enter the following command:

```
wmsgbrowse -a
```

which returns the following output:

```
1608652338.45 standard lab15053-aix
1608652338.45 external
 inventory
1608652338.45 standard lab16001-nt
1608652338.45 external
1608652338.43 standard lab15213-XP
 delete_cmstatus_entry lab15213-XP
 inventory
 inventory
 inventory
1608652338.43 external
```

2. To display a list of all available loggers and filter keywords, enter the following command:

```
wmsgbrowse -s
```

which returns the following output:

```
Available loggers:
inventory, standard, bulk_data, delete_cmstatus_entries, delete_cmstatus_entry,
add_cmstatus_entry, edit_cmstatus_entry, external,
mdist2_db, publish, gatewayweb.
Filter keywords:
distribution_id, logger_name, endpoint_name, operation_type,
operation_mode, endpoint_id, endpoint_guid, spo_oid,
base_name, base_version,base_oid, origin_user, exec_time,
message, exit_codes_msg, state, name, version,spname,
pathname, exception, listeners, primary.
```

3. To format the command output in order to display the distribution ID and operation type for endpoint target1, enter the following command:

```
wmsgbrowse -t target1 -r "\n Distribution ID:%distribution_id%
\n Operation Type:%operation_type%"
```

which returns the following output:

```
Distribution ID:1608652338
Operation Type: install
```

See Also

None.

wmvspobj

Moves one or more software packages from the lost-n-found collection to another collection.

Syntax

wmvspobj *sp_name* [*sp_name*]... [**–s** *src_host_name*] [**–h** *log_host_name*] *target_collection*

Description

This command moves the specified software package or packages from the lost-n-found collection to the specified collection. A software package resides in the lost-n-found collection if its source host, log host or profile manager was removed from the Tivoli environment. You can specify multiple software packages on the same command line. Since this command is a bash script that must be run in the bash environment, precede the command with the string **sh**.

Options

sp_name
: Name of one or more software package objects registered in the Tivoli Name Registry that are to be moved from lost-n-found. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

-s *src_host_name*
: Specifies the name of the new source host for the one or more software package objects to be moved from lost-n-found. If this attribute is not specified, the name of the previous source host (before the object was moved to the lost-n-found collection) is used by default. If the new source host you specify is not valid, or if you do not specify a source host and the previous source host does not exist, the source host value is not set.

–h *log_host_name*
: Specifies the name of the new log host for the one or more software package objects to be moved from the lost-n-found. If this attribute is not specified, the name of the previous log host (before the object was moved to the lost-n-found collection) is used by default. If the new log host you specify is not valid, or if you do not specify a log host and the previous log host does not exist, the log host value is not set.

target_collection
: Profile manager of the target collection. You can specify the name of a profile manager or policy region. Do not precede the name with the @ character.

Authorization

admin, senior, or super

Return Values

The **wmvspobj** command returns one of the following:

0 Indicates that **wmvspobj** started successfully.

-1 Indicates that **wmvspobj** failed due to an error.

Examples

To move the fsys^1.0 software package from the lost-n-found collection to the myprofile profile manager on a Windows platform, enter the following command:

```
sh wmvspobj fsys^1.0 myprofile
```

See Also

None.

wremovsp

Removes a software package from selected target systems.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wremovsp [[-**R**{ *y* | *n*}] [**–D** *variable=value...*] [**–f** | **–I** | –**i** [–**v**]] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–u** *y* | *n* | *o* | *u* [**–a**]] [**–l** *mdist2_token=value...*]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wremovsp [[-**R**{ *y* | *n*}] [**–D** *variable=value...*] [**–f** | **–I** | –**i** [–**v**]] [**–t** *y* | *n* | *o*] [**–u** *y* | *n* | *o* | *u* [**–a**]] [**–l** *mdist2_token=value...*]] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

Description

Options

–R *y* | *n*

Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–D *variable=value*

Defines the value of a variable used in the software package, to add or override existing variables. If you are removing a software package that has already been installed, you can define or override only those variables that were not solved during the previous install operation. When specifying multiple variables, repeat the **–D** argument before each *variable=value*. Note that these variables can be resolved only on the endpoint.

–f Specifies forcing the operation, regardless of the state of the target system.

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–v Specifies that verbose logging is to be used. If verbose logging is not used, any failure in the distribution is indicated by a short message, for example:

```
ep1 - Failed dependency check
```

If verbose logging is used, a full description of the reason for failure is logged.

Use of the verbose logging option causes a significant increase in the size of the log file. Therefore, this option must be used with caution. This argument is only available if the **–i** argument is specified.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing a remove operation and the software package has already been removed on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–t *y | n | o*
Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y | n | o | r*
Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see "Commit Operation" on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options.

–u *y | n | o | u*
Specifies the undoable option: **n** (no, which is the default), **y** (yes), **o** (preferably), or **u** (undoable-in-transactional).

–a Specifies that the operation should be automatically accepted.

–l *mdist2_token=value*
Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**remove**)*"* where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software

Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

`escalate_msg_n=@filename`

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-X {none | first | middle | last | both}
Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*
Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last

distribution has completed. If the user does not respond to the dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–T *subscribers _file*
Name of a file containing a list of the subscribers for the operation. Subscribers should be specified one per line, using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...
The names of the target systems or profile manager from which to remove the software package. If no target systems are specified, the software package is removed from all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization
admin, senior, or super

Return Values
The **wremovsp** command returns one of the following:

0 Indicates that **wremovsp** started successfully.

-1 Indicates that **wremovsp** failed due to an error.

Examples
1. To remove software package object fsys^1.0 in transactional mode from the subscribers target1 and target2, enter the following command:
   ```
   wremovsp -ty @fsys^1.0 @target1 @target2
   ```
2. To remove software package object fsys^1.0 in undoable mode with auto-accept from the subscribers target1 and target2, enter the following command:
   ```
   wremovsp -uy -a @fsys^1.0 @target1 @target2
   ```
3. To remove software package object fsys^1.0 from the subscribers target1 and target2 only if they have been successfully checked, enter the following command:
   ```
   wremovsp -I @fsys^1.0 @target1 @target2
   ```

See Also

- **winstsp**
- **waccptsp**
- **wundosp**
- **wcommtsp**
- **wrpt** in the *Tivoli Management Framework: Reference Manual*
- **wmdist** in the *Tivoli Management Framework: Reference Manual*

wsdvers

Gets the package or versioning type of the specified software package or a list of later versions of the package.

Syntax

wsdvers {**–n** | **–v** | **–t**} *spobj_name* ...

Description

Depending of the argument specified, this command retrieves one of the following for specified software packages:

- The package type
- The versioning type
- A list of later versions of the package

Options

– n Specifies that the command should return a sorted list of later versions of the specified software packages.

– v Specifies that the command should return the versioning type of the specified software packages. Possible versioning types are **NONE** and **SWD**.

– t Specifies that the command should return the package type of the specified software packages. Possible package types are **PATCH** and **REFRESH**.

spobj_name **...**
The registered name and version of a software package for which information is to be returned. Multiple packages can be specified. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, super, or user

Return Values

The **wsdvers** command returns one of the following:

0 Indicates that **wsdvers** started successfully.

-1 Indicates that **wsdvers** failed due to an error.

Examples

1. To retrieve the list of all later versions of the software package object fsys_test.1.0, enter the following:

   ```
   wsdvers -n @fsys_test.1.0
   ```

2. To retrieve the package types of packages pkg.1.0, pkg.1.2, and pkg.2.0, enter the following:

   ```
   wsdvers -t @pkg.1.0 @pkg.1.2 @pkg.2.0
   ```

See Also

None.

wsetsnsp

Specifies the list of software packages to be nested in a primary software package.

Syntax

wsetsnsp [**–p** *spobj_name* | **–r**] **–n** *spobj_name... spobj_name*

Description

This command specifies a list of software packages that are to be nested in a primary software package. By default, the primary software package is processed after all its nested software packages. The command can also be used to remove nested packages, but not to remove the primary package.

For information about the format in which the names of software packages should be specified, see “Software Package Name and Version” on page 2.

Options

–p *spobj_name*
: The name of the nested software package in the primary package after which the newly nested package or packages will be inserted when creating the list. If this option is not specified, the nested package or packages are inserted at the beginning of the list of nested software packages. You can also use this option to specify that the primary software package is to be processed first, as described in example 6. If the list of software packages is already existing, you cannot use the **-p** option to modify the sequence of the software packages. In this case, you must remove the package and insert it again in the desired position.

–r
: Indicates that the nested software package, which is specified using the **–n** argument, is to be removed.

–n *spobj_name*
: The registered name of one or more software packages to be nested in the primary software package. Specify **–n** before each listed software package name.

spobj_name
: The name of the primary software package in which the software package or packages specified in *spobj_name* are to be nested.

Authorization

admin, senior, super, or user

Return Values

The **wsetsnsp** command returns one of the following:

0
: Indicates that **wsetsnsp** started successfully.

-1
: Indicates that **wsetsnsp** failed due to an error.

Examples

1. To retrieve the list of nested software packages in the primary software package pk, enter the following command:

   ```
   wgetsnsp @pk^1.0
   ```

   The result of this command is the following list:

```
pk1^1.0
pk2^1.0
pk^1.0
```

2. To add nested software packages pk3 and pk4 after software package pk2, enter the following command:

   ```
   wsetsnsp -p pk2^1.0 -n pk3^1.0 -n pk4^1.0 @pk^1.0
   ```

3. To check the results, enter the following command again:

   ```
   wgetsnsp @pk^1.0
   ```

   which provides the following output:

   ```
   pk1^1.0
   pk2^1.0
   pk3^1.0
   pk4^1.0
   ```

4. To remove the nested software package pk4, enter the following command:

   ```
   wsetsnsp -r -n pk4^1.0 @pk^1.0
   ```

5. To check the results, enter the following command again:

   ```
   wgetsnsp @pk^1.0
   ```

   which provides the following output:

   ```
   pk1^1.0
   pk2^1.0
   pk3^1.0
   pk^1.0
   ```

6. To process the primary software package pk1 first and process the nested software packages pk2 and pk3 after pk1, enter the following command:

   ```
   wsetsnsp -p pk1^1.0 -n pk2^1.0 -n pk3^1.0 @pk1^1.0
   ```

See Also

- **wgetsnsp**
- “Nesting Software Packages” on page 16.

wsetspat

Sets the attributes for a software package object.

Syntax

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** all | check] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** all | force] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** all | ignore] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** all | preview] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** src | check] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** src | force] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** src | ignore] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** src | preview] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** repair | check] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** repair | force] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** repair | ignore] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

wsetspat{[**–b** *move_removing_host*] [**–c** *y* | *n* | *o*] [**–h** *src_host*] [**–l** *lenient*] [**–m** repair | preview] [**–M** *default_cmop_mode*] [**–o** *default_cmop*] [**–s** *check_no_src_host*] [**–u** *undo*] [**–v** *web_view_mode*] [**–w** *staging_area*]} *spobj_name*

Description

Options

–b *move_removing_host*
Sets the value of the move_removing_host attribute, which moves the software package to the lost-n-found collection if the log host or source host of the software package is removed. Possible values are true (the default) or false.

–c *y | n | o*
Specifies the committable attribute, which indicates if you can execute a committable operation on the object. Possible values are:

y The installation and the remove operations that you perform on this software package object must be transactional.

n The installation and the remove operations that you perform on this software package object cannot be performed in transactional mode.

o The installation and the remove operations that you perform on this software package object can be either transactional or not transactional.

–h *src_host*
Specifies the source host where the files in the software package block are obtained and where the software package block is stored. The source host can be any of the available managed nodes.

–l *lenient*
Sets the lenient_distribution attribute. Possible values are:

true Allows distributions and removals from the command line to any endpoint or profile manager, even if that endpoint or profile manager is not currently a subscriber to the profile manager of the software package (or to any of its subscribing profile managers).

false Allows distributions and removals from the command line to only endpoints or profile managers that are currently subscribers to the profile manager of the software package (or to any of its subscribing profile managers).

–m *default_svr_mode*
Specifies the server mode that is used for a default distribution. A default distribution operation is performed when you select **Distribute** from the profile manager icon or the profile manager menu, or when you drag a software package icon to a subscriber icon. You can optionally choose one value from each of the following lists. If more than one value is specified, they must be separated by the pipe (|) character and enclosed in quotation marks, for example:

```
–m "all|check"
```

Specify one of the following options, which are mutually exclusive:

1. **all**, which installs all the objects in the package
2. **src** (source), which installs only those files on the source that have a modification time later than the time of the last successful installation

3. **repair**, which installs only those source system objects that have changed on the target system or that are not present on the target system, to make the target system objects consistent with the source system objects

and one of the following options, which are mutually exclusive:

1. **check**, which checks the operation only, without submitting it
2. **force**, which forces the operation
3. **ignore**, which executes the operation on only successfully checked target systems and ignores the target systems for which the check is unsuccessful
4. **preview**, which lists all the actions to be executed on the specified subscribers

–M *default_cmop_mode*

Specifies the mode for change management operations that is used for a default distribution. A default distribution operation is performed when you select **Distribute** from the profile manager icon or the profile manager menu, or when you drag a software package icon to a subscriber icon. You can optionally choose one value from the following list. If more than one value is specified, they must be separated by the pipe character (|) and enclosed in quotation marks, for example:

```
–M "transactional|auto_commit"
```

which sets both the values **transactional** and **auto_commit** as the default mode for change management operations. Possible values are as follows:

- **transactional**
- **not_transactional**
- **prefer_not_transactional**
- **auto_commit**
- **undoable**
- **prefer_undo**
- **auto_accept**
- **undoable_in_transactional**
- **prefer_not_reboot**
- **during_reboot**
- **auto_reboot**
- **force**

See Chapter 2, “Performing Change Management Operations,” on page 143 for more information about these modes.

–o *default_cmop*

Specifies the change management operation that is used for a default distribution. A default distribution operation is performed when you select **Distribute** from the profile manager icon or the profile manager menu, or when you drag a software package icon to a subscriber icon. Possible values are as follows:

- **install**
- **remove**
- **undo**
- **accept**
- **commit**

See “Types of Change Management Operations” on page 144 for more information about these operations.

–s *check_no_src_host*
Sets the value of the check_no_src_host attribute, which verifies the existence of all files on the source host as defined in the software package. Possible values are true or false.

–u *undo*
Specifies whether you can execute an undoable operation on the object. Possible values are:

y The installation and the remove operations that you perform on this software package object must be undoable.

n The installation and the remove operations that you perform on this software package object cannot be performed in undoable mode.

o The installation and the remove operations that you perform on this software package object can be either undoable or not undoable.

–v *web_view_mode*
Specifies the value of the web_view_mode attribute, which specifies the access permissions for the package when using the Web Interface feature. Possible values are:
- **hidden**. This is the default value.
- **subscriber**
- **public**

–w *staging_area*
Specifies the value of the staging_area attribute, which is the working directory of the distribution server.

spobj_name
The name and version of the software package whose attributes are set by this operation. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization
admin, senior, or super

Return Values
The **wsetspat** command returns one of the following:

0 Indicates that **wsetspat** started successfully.

-1 Indicates that **wsetspat** failed due to an error.

Examples
1. For an object called fsys_test^1.0, to set the lenient distribution attribute to **true**, the undoable attribute to **n**, and the source path to d:\testdir\fsys.spb, enter the following command:

   ```
   wsetspat -l true -u n d:\testdir\fsys.spb @fsys_test^1.0
   ```

2. For an object called fsys_test^1.0, to set the change management operation mode to **undoable** and **auto-accept**, and the source path to d:\testdir\fsys.spb, enter the following command:

   ```
   wsetspat -M "undoable|auto_accept" d:\testdir\fsys.spb @fsys_test^1.0
   ```

See Also
wgetspat

wsetspgs

Sets the program information associated with a software package object.

Syntax

wsetspgs {[**–a** *after_prog_path*] [**–b** *before_prog_path*] [**–i** *after_input_path*] [**–j** *before_input_path*]] [**–p** *before_prog_env*] [**–P** *after_prog_env*] [**–s** *before_skip_non_zero*] [**–u** *after_as_uid*] [**–U** *before_as_uid*] [**-l** *lcf_before_prog_path*] [**-c** *lcf_before_prog_args*] [**-t** *lcf_before_prog_timeout*] [**-L** *lcf_after_prog_path*] [**-C** *lcf_after_prog_args*] [**-T** *lcf_after_prog_timeout*]} *spobj_name*

Description

The program information associated with a software package object that you can set with this command is listed below.

Options

–a *after_prog_path*
: Sets a value for the `after_prog_path` attribute, which is the path of the program to be run on the source host after the build is completed.

–b *before_prog_path*
: Sets a value of the `before_prog_path` attribute, which is the path of the program to be run on the source host before the build is begun.

–i *after_input_path*
: Sets a value for the `after_input_path` attribute, which specifies the input parameters for the `after_prog_path` attribute. This value, although set, has no meaning unless the **–a** (after_prog_path) attribute is defined.

–j *before_input_path*
: Sets a value for the `before_input_path` attribute, which specifies the input parameters for the `before_prog_path` attribute. This value, although set, has no meaning unless the **–b** (before_prog_path) attribute is defined.

–p *before_prog_env*
: Sets the list of environment variables for the before program.

–P *after_prog_env*
: Sets the list of environment variables for the after program.

–s *before_skip_non_zero*
: Sets the `before_skip_non_zero` attribute. Specify either **true** or **false**, where **true** indicates that distribution is skipped if before_prog_path exits with a non-zero exit code, and **false** indicates that it is not skipped.

–u *after_as_uid*
: Sets the `after_as_uid` attribute, which sets the UNIX user ID under which to run after_prog_path. Specify an integer value.

–U *before_as_uid*
: Sets the `before_as_uid` attribute, which sets the UNIX user ID under which to run before_prog_path. Specify an integer value.

-l *lcf_before_prog_path*
: Sets a value for the `lcf_before_prog_path` attribute, which is the pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run. To cancel a setting defined for this option, enter a blank space between quotes.

-c *lcf_before_prog_args*
Sets a value for the `lcf_before_prog_args` attribute, which is one or more arguments passed to the program in addition to the default ones. Separate arguments with a blank space. You can also use variables to specify arguments.

-t *lcf_before_prog_timeout*
Sets a value for the `lcf_before_prog_timeout` attribute, which is The time, expressed in seconds or the value -1, to wait for the completion of the before program. See "Format of the execute_user_program Stanza" on page 118 for detailed information.

-L *lcf_after_prog_path*
Sets a value for the `lcf_after_prog_path` attribute, which is the pathname to the program to be run. You can also use a variable, such as *product_dir*, to express part or the whole pathname. You must use a system variable, or list the variables you use in the swdis.var file. The program must already be present on the target system on which it is to run. To cancel a setting defined for this option, enter a blank space between quotes.

-C *lcf_after_prog_args*
Sets a value for the `lcf_after_prog_args` attribute, which is one or more arguments passed to the program in addition to the default ones. Separate arguments with a blank space. You can also use variables to specify arguments.

-T *lcf_after_prog_timeout*
Sets a value for the `lcf_after_prog_timeout` attribute, which is The time, expressed in seconds or the value -1, to wait for the completion of the after program. See "Format of the execute_user_program Stanza" on page 118 for detailed information.

spobj_name
The name and version of the software package as an absolute path or a registered name. For information about the format of the name and version string, see "Software Package Name and Version" on page 2.

Authorization

admin, senior, or super

Return Values

The **wsetspgs** command returns one of the following:

0 Indicates that **wsetspgs** started successfully.

-1 Indicates that **wsetspgs** failed due to an error.

Examples

1. To set the value cmdname for the after_prog_path attribute, which runs after the software package object fsys_test^1.0 is applied, enter the following command:

   ```
   wsetspgs -a cmdname @fsys_test^1.0
   ```

2. To cancel the setting previously defined for the -l option for software package object fsys_test^1.0, enter the following command:

   ```
   wsetspgs -l " " @fsys_test^1.0
   ```

3. To define a custom variable for a before program which runs before the software package object fsys_test^1.0 is applied, enter the following command:

   ```
   wsetspgs -c "var1 var2" @fsys_test^1.0
   ```

4. To specify a program to be run within five minutes after the software package object fsys_test^1.0 is applied, enter the following command:

```
wsetspgs -L "/tmp/myscript" -T 300 @fsys_test^1.0
```

See Also

wgetspgs, **wdswdvar**

wsetspop

Sets the options for a software package object.

Syntax

wsetspop –g *log_gid spobj_name*

wsetspop –h *log_host spobj_name*

wsetspop –j *log_mode spobj_name*

wsetspop –L *log_path spobj_name*

wsetspop –m *mail_id spobj_name*

wsetspop –P *post_notice spobj_name*

wsetspop –r *no_chk_on_rm spobj_name*

wsetspop –u *log_uid spobj_name*

Description

The options you can set with this command are listed in the Arguments section.

Options

–g *log_gid*
: Sets the log_gid attribute, which is the UNIX group ID of the log file. Specify an integer value.

–h *log_host*
: Sets the log_host attribute, which specifies the label of the managed node where the log file is generated. Specify the host name.

–j *log_mode*
: Sets the log_mode attribute, which is the UNIX file mode of the generated log file. Specify any valid octal value.

–L *log_path*
: Sets the log_path attribute, which is the UNIX path name and file name of the generated log file. Specify the UNIX path name and file name as an absolute value.

–m *mail_id*
: Sets the mail_id attribute, which is the address for notification. Specify the name of a user e-mail ID. To specify multiple e-mail IDs, separate each ID with a comma. (See the Example section of this command for information about deleting an existing e-mail ID notification.)

–P *post_notice*
: Sets the post_notice attribute, which specifies whether to send a notice. Possible values are true or false.

-r *no_chk_on_rm*
: Specifies whether a check is performed on the whole Tivoli Management region when the specified software package is deleted to verify if it is nested in other software packages. The default value is **true**. In this case, no check is performed, and an error message is displayed when you try to install a software package nesting the deleted software package. If you set

the attribute to **false**, a check is performed. If the deleted software package was nested in any other software packages, it is removed.

–u *log_uid*
Sets the log_uid attribute, which is the UNIX user ID of the log file. Specify an integer value.

spobj_name
The name and version of the software package whose options are set by this operation. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

Authorization

admin, senior, or super

Return Values

The **wsetspop** command returns one of the following:

0 Indicates that **wsetspop** started successfully.

-1 Indicates that **wsetspop** failed due to an error.

Examples

1. To delete the mail address notification previously set for the object package^1, enter the following command:

   ```
   wsetspop -m " " @package^1
   ```

2. To enable the post_notice function, enter the following command:

   ```
   wsetspop -P true @package^1
   ```

See Also

wgetspop

wsetsps

Records the presence on a target system of an application that was installed independently of Software Distribution.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wsetsps [**-i | -f | -I**] [**-R** {**y |n**}] [**-v** *versioning_type*] [**-t** *package_type*] [**-l** *mdist2_token=value...*] ...**-s** *spname.spver... targets...*

wsetsps [**-i | -f | -I**] [**-R** {**y |n**}] [**-v** *versioning_type*] [**-t** *package_type*] [**-l** *mdist2_token=value...*] ...**-s** *spname.spver...* **-T** *targets_file...*

wsetsps [**-i | -f | -I**] [**-R** {**y |n**}] [**-v** *versioning_type*] [**-t** *package_type*] [**-l** *mdist2_token=value...*] ...**-S** *pkgfile... targets...*

wsetsps [**-i | -f | -I**] [**-R** {**y |n**}] [**-v** *versioning_type*] [**-t** *package_type*] [**-l** *mdist2_token=value...*] ...**-S** *pkgfile...***-T** *targets_file ...*

Description

Using this command, you can add applications that were installed independently of Software Distribution to the Software Distribution catalog on the endpoint. The specified software package is assigned a state of IC-D-, indicating that it is installed and ″discovered.″

There are limitations to the change management operations that can be used for a discovered software package. Only the following operations are available:

- Remove software package (not in Transactional mode). The related entry is removed from the Software Distribution catalog on the endpoint, but the application is not uninstalled.
- Force install software package

Options

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing an install operation and the software package is already installed on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–f Specifies forcing the operation, regardless of the state of the package on the target system.

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–R *y | n*

Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed

on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

-v *versioning_type*
Specifies whether to use version checking when adding the package to the catalog. Possible values are SWD (use version checking) and None (no version checking). The default is SWD.

–t *package_type*
Specifies the type of package and controls the checks made for other versions of the same package. Possible values are REFRESH and PATCH. The default is REFRESH. See “Software Package Version Checking” on page 4 for details of how version checks are made.

–l *mdist2_token=value*
Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string **setsps (setsps)**.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater waits for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater waits for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-S *pkg_file*
Specifies a file containing names of software packages to be registered with Software Distribution.

-s *spname.spver*
Name and version of the installed software package. See “Software Package Name and Version” on page 2.

You can specify multiple packages.

–T *targets_file*
Name of a file containing a list of the targets for the operation. Targets can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify targets separately on the command line (see the description of the *targets* attribute for this command). If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the operation log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

targets...
The names of the target systems or profile manager to be synchronized.

You can specify subscribers individually, using this argument, specify a file, using the **–T** argument, or both. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the operation log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wsetsps** command returns one of the following:

0 Indicates that **wsetsps** started successfully.

-1 Indicates that **wsetsps** failed due to an error.

Examples

To discover the refresh the software packages contained in the mypkgs1 file and software package test.2.1 on endpoints endpt1 and endpt2, with version checking, enter the following command:

```
wsetsps -v SWD -t REFRESH -S mypkgs1 -s test1.0 -s test.2.1 @endpt1 @endpt2
```

See Also

wdsetsps

wspmvdata

Moves data between source hosts, that is Tivoli managed nodes with Software Distribution installed, and endpoints, and between one endpoint and more endpoints.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wspmvdata –s *origin_list* **–t** *destination_list* [**–P sp | tp:** *path*] [**–r** *spre | spost | tpre | tpost:script*] [**–S** *sw_package*[*:status*]] [**–c**] [**–D** variable=value]... [**–I**] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–R**] [**-B**] [**-F**] *file*

wspmvdata –d *destination_list* [**–P tp:** *path*] [**–r** *tpre | tpost:script*] [**–S** *sw_package*[*:status*]] [**–I**] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–R**] [**-B**] *file*

wspmvdata –s *origin_list* **–t** *destination_list* [**–P sp | tp:** *path*] [**–r** *spre | spost | tpre | tpost:script*] [**–S** *sw_package*[*:status*]] [**–c**] [**–D** variable=value]... [**–I**] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [**-X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–R**] [**-B**] [**-F**] [**-G**] *file*

wspmvdata –A

wspmvdata –p *profile_manager*

wspmvdata –h *managed_node*

Description

This command is used to perform the following data-updating tasks:

- Sending data from an origin system, including endpoints, to multiple destination systems.
- Retrieving data from multiple endpoints to update values on a source host, that is a Tivoli managed node with Software Distribution installed.
- Deleting data on multiple systems.
- Ability to send, retrieve, or delete multiple files by using wildcards or matching indicators, in addition to Software Distribution standard variables.
- Running pre- and post-transfer tasks on both the origin and destination systems.
- Specifying software dependencies.
- Applying the selected operation to all subdirectories in the specified path using the recursive option.
- Send files with similar names to different endpoints. The file to be sent is identified by the endpoint label, therefore, each endpoint receives only the file named with its label. For more information, see “Sending Multiple Files” on page 242.

In the data moving architecture, data is moved between source hosts and endpoints and between one endpoint and multiple endpoints. A source host is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed. The source host corresponds to the origin system when send operations are performed, with the exception of send operations from one endpoint to multiple endpoints, where the origin system is an endpoint. During a retrieve operation, on the other hand, the source host is the destination system.

All data moving operations use the same software package object, DataMovingRequest.1. This object contains certain standard information to be used by all data moving operations, including logging options. This object is either created automatically at installation time or by using the **–A** or **–p** *profile_manager* mutually exclusive options. If neither of these operations is performed, the object is created automatically in the first profile manager that belongs to a region having SoftwarePackage as managed resource when the first data moving operation is performed.

Information about operations performed using the DataMovingRequests.1 is not stored in the Inventory database.

Data moving operations are logged in a file named **DataMovingRequests.1.log**. By default, this file is written in the working_dir defined using the **working_dir** key with the **wswdcfg** command.

You can split this file using the **split_dm_log** option in the **wswdcfg** command. For more information on this command, see "wswdcfg" on page 246.

If you enable the **split_dm_log** option, the data moving log is split into a separate file for each data moving operation. The resulting files are named according to the following standard: `DataMovingRequest.`*`DIST_ID`*`.log`

where

DIST_ID is the last portion of the MDist 2 distribution ID.

For more information about the DataMovingRequests.1 object and configuring the Data Moving service, see ″Configuring the Data Moving Service″ in the *User's Guide for Software Distribution*.

Options

–s *origin_list*

Identifies the system or systems where the data that is to be moved originates. When sending data, the origin system must be a source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed, or an endpoint. When retrieving data, the origin list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these.

Components of the list must be separated by commas but the list must not end with a comma. No spaces are allowed. Each endpoint name must be preceded by @. For example,

```
–s @test1,@test2,@pm1,file1
```

If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the **DataMovingRequests.1.log** file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

–d Specifies that the data movement is a delete operation. Do not specify an origin list if you use this argument.

Note: When this option is specified, empty directories are removed, and a warning message is inserted in the log file.

–t *destination_list*

Identifies the system or systems to which data is to be transferred or where data is to be deleted. When retrieving data, the destination system must be a source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed. When sending or deleting data, the destination list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these.

Components of the list must be separated by commas but the list must not end with a comma. No spaces are allowed. Each endpoint name must be preceded by @. For example,

```
–t @ep1,@ep2,@pm1,file1
```

This argument is always required.

If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the **DataMovingRequests.1.log** file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

–P Specifies the location of the file, as follows:

sp:*origin_path*

Specifies the location of the file on the origin system or systems.

tp:*dest_path*

Specifies the location on the destination system or systems, to which the file is to be copied.

Note: If you are performing a retrieve operation, the destination directory on the system is not created with this command, so it must be created beforehand. During the retrieve operation the program creates a new sub-directory under the specified destination directory using the following naming convention:
`endpointname_distributionID_timestamp`.

See “File Paths” on page 238.

–r *spre | spost | tpre | tpost:script*

Specifies a script to run, before or after data movement on the origin or destination system, as follows:

spre:*src_prescript*

Specifies a script to run on the origin system of the data file, before the data is transmitted. When sending data, the origin system must be a source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed, or an endpoint, when data is sent from one endpoint to one or more endpoints. When retrieving data, the origin list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these. Where the **–s** option specifies a list of endpoints, the script runs on each endpoint.

spost:*src_postscript*

Specifies a script to run on the origin system of the data file, before the data is transmitted. When sending data, the origin system must be a source host, that is a Tivoli managed node, functioning as a

gateway or a repeater, where Software Distribution is installed, or an endpoint, when data is sent from one endpoint to one or more endpoints. When retrieving data, the origin list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these. Where the **–s** option specifies a list of endpoints, the script runs on each endpoint.

tpre:*targ_prescript*
Specifies a script to run on the destination system, before the data is transmitted. When retrieving data, the destination system must be a source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed, which afterwards redirects the data to the destination systems. When sending or deleting data, the destination list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these. Where the **–t** option specifies a list of endpoints, the script runs on each endpoint.

tpost:*targ_postscript*
Specifies a script to run on the destination system, before the data is transmitted. When retrieving data, the destination system must be a source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed, which afterwards redirects the data to the destination systems. When sending or deleting data, the destination list can include multiple Tivoli endpoints, files that store a list of endpoints, profile managers, or a combination of these. Where the **–t** option specifies a list of endpoints, the script runs on each endpoint.

–S *sw_package:state*
Specifies a software dependency that must be met on the destination systems for the operation to proceed. Only one dependency can be defined with a single use of the **–S** argument. To define more dependencies, you must include the argument multiple times.

For each dependency, you specify a valid software package and one of the following states:

I The package must be in the IC, ICU or I--D state.

R The package must be in the RC or RCU state, or it must never have been installed.

The default state is **I**. If you do not include a state, the default is assumed.

The condition specified using this argument is mapped to the dependency attribute in the log file, as follows:

–S mypkg.1.0:**I** becomes **$(installed_software) ==** ″**mypkg.1.0**″

–S mypkg.1.0:**R** becomes **$(installed_software) !=** ″**mypkg.1.0**″

For more information about dependency checking, see “Dependency” on page 9.

There are two stages in the software dependency check. The first check is to the Inventory before transmission. If this check returns a value of false, the transmission is ended. Otherwise, a check is made on each target system. If any of the targets fail to meet the requirement, the transmission

is ended for all targets, unless the **–I** argument is specified in the command. In this case, the transmission is sent only to targets that pass the check.

–c Specifies that codepage translation is required.

–D *variable=value*
Defines the value of a variable that is to be added or is to override existing variables. When specifying multiple variables, repeat the **–D** attribute before each *variable=value*. Note that these variables can be resolved only on the endpoint.

–h *variable=value*
Specifies a Tivoli managed node. If no source host is specified, the default host is used.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that meet the software dependency requirements that are specified by the **–S** argument. For example, if you are performing a delete operation and the software package has already been deleted on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–l *mdist2_token=value*
Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"filename (operation) "* where *filename* indicates the file name, and *operation* is one of the following:
- **send**
- **retrieve**
- **delete**

depending on the operation submitted.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command for each repeater and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here. The default value is 30 minutes.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command for each repeater (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command for each repeater (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here. When retrieve and send from endpoint to endpoint operations are performed, the software package is built on the endpoint. As this operation can require a longer amount of time than the default timeout value allows for, set a higher timeout value.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet performed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not performed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
Indicates whether the operation on mobile targets is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the

end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-X {none|first|middle|last|both}
Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*
Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a

shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last distribution has completed. If the user does not respond to the dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–p *profile_manager*
Specifies the profile manager in which the DataMovingRequests.1 software distribution object is to be created.

Note: Before specifying a profile manager name, make sure that the profile manager belongs to a region having SoftwarePackage as managed resource.

DataMovingRequests.1 is the software distribution object used for all data moving operations. It includes general information for data moving operations, for example, the name and location of the log file.

–A Triggers the creation of the DataMovingRequests.1 software distribution object in a profile manager that belongs to a region having SoftwarePackage as managed resource

-R Specifies that the selected operation will be applied to all subdirectories in the source and target directories. This option is not supported with matching indicators.

-B Use this option to specify that the whole data moving operation must be considered as failed if the post-script on the source host fails. This option is available only for send and retrieve operations.

-F Use this option to specify that the send operation must proceed even when one or more files to be sent to endpoints are not present on the source host. This option applies to the case in which you are sending multiple files using the $(ep_label) variable to specify which files are to be sent. In this case, the $(ep_label) variable allows you to send files with similar names to different endpoints by substituting the label of the endpoint, for example 'data.*endpoint_name*.txt' where *endpoint_name* is the label of the endpoint. The $(ep_label) variable can be used only to send files from managed nodes to endpoints, and not from endpoints to other endpoints. For more information, see "Sending Multiple Files" on page 242.

-G Use this option to modify the behavior of the command when retrieving one file from the endpoints. If you specify this option, the file is saved on the destination system according to the following naming convention: *name_endpoint_timestamp_distribution_id.extension*. If this option is not specified, the default behavior applies and the retrieved file is saved with its original name to a directory on the destination system named according to the following convention: *endpoint_distribution_id_timestamp*. For more information, see "File Paths" on page 238.

Note: This parameter can be used only when retrieving a single file.

file Specifies the name of the file to be moved. The file name can be fully qualified or relative to the paths specified in the origin and destination lists. If the file name is fully qualified, the specified path defines the

location of the file on the origin and destination systems. See “File Paths.” This option also allows you to send files with similar names to different endpoints by substituting the label of the endpoint, for example ’data.*endpoint_name*.txt’ where *endpoint_name* is the label of the endpoint. For more information, see “Sending Multiple Files” on page 242.

Note: Hard links and symbolic links are not supported. Hard links are turned to files and lose the link to the original file, while symbolic links are ignored.

File Paths: The CLI allows definition of any or all of the following:
- An origin path
- A destination path
- A qualified file name

The qualified file name is appended to the origin and destination paths to obtain the full path to the file on the origin and destination systems.

Note: Due to the differences in default drive between the source host, where the default drive is the drive where Software Distribution is installed, and Windows 2000 endpoints where the default drive is defined in a system variable, it is advisable to include the drive in the definition of fully qualified paths.

The origin and destination path values are optional and the file name may be unqualified. The examples that follow show how the file location is resolved depending on the presence or absence of these values.

```
wspmvdata -s @lab15124 -P sp:/usr/sd/ -t @lab67135-w98,
@lab15180-2000 /source/data.txt
```

On the origin system, the file is: /usr/sd/source/data.txt.
On the destination system, the file is: /source/data.txt

```
wspmvdata -s @lab15124 -P sp:/usr/sd/source -t @lab67135-w98,
@lab15180-2000 data.txt
```

On the origin system, the file is: /usr/sd/source/data.txt.
On the destination system, the file is: /<*default dir*>/data.txt

In the last example, no destination path is specified and the file name is unqualified, so a default path is used for the destination location. The default path for Tivoli managed nodes is the current working directory of the SH process implementation ($DBDIR). The default path for endpoints is the <prod_dir> directory, which can be set in the swdis.ini file.

If the target directory is not preceded by a backslash, it is created in the default directory.

```
wspmvdata -s @lab15124 -P sp:/usr/sd/source -P tp:dest -t @lab67135-w98,
@lab15180-2000 data.txt
```

On the origin system, the file is: /usr/sd/source/data.txt.
On the destination system, the file is: /<*default dir*>/dest/data.txt

If the backslash is inserted, the target directory dest is created at root level.

```
wspmvdata -s @lab15124 -P sp:/usr/sd/source -P tp:/dest -t @lab67135-w98,
@lab15180-2000 data.txt
```

On the origin system, the file is: /usr/sd/source/data.txt.
On the destination system, the file is: /dest/data.txt

When performing a retrieve operation, a new sub-directory is created under the specified destination directory using the following naming convention: endpointname_distributionID_timestamp. A single directory is created on the source host for each endpoint, as described in the following example:

```
wspmvdata -t @lab21543mn -s @lab21459,@lab21635,@lab21857
-P sp:/usr/sd/source -P tp:/usr/sd/target data.txt
```

This ensures that each retrieved file is stored in a unique directory on the source host.

On the origin system, the file is: /usr/sd/source/data.txt.

On the destination system, the file for endpoint lab21459 is: /usr/sd/target/lab21459_1506362350.267_20050421112728/data.txt.

On the destination system, the file for endpoint lab21635 is: /usr/sd/target/lab21635_1384061647.853_20050421112752/data.txt.

On the destination system, the file for endpoint lab21857 is /usr/sd/target//lab21857_1956072719.249_20050421112803/data.txt.

If you are retrieving only one file from each endpoint, you can choose to save the file on the destination system with the following naming convention: file_name_endpoint_name_timestamp_distribution_id.file_extension, as described in the following example:

```
wspmvdata -t @lab21543mn -s @lab21459,@lab21635,@lab21857
-P sp:/usr/sd/source -P tp:/usr/sd/target -G data.txt
```

On the origin system, the file is: /usr/sd/source/data.txt.

On the destination system, the file for endpoint lab21459 is: /usr/sd/target/data_lab21459_20050421110213_1506362350.267.txt.

On the destination system, the file for endpoint lab21635 is: /usr/sd/target/data_lab21635_20050421112413_1685244375.497.txt

On the destination system, the file for endpoint lab21857 is: /usr/sd/target/data_lab21857_20050421111421_1375294728.468.txt

To perform this operation, use the **-G** option.

Using Wildcards and Matching Indicators: With the **wspmvdata** command, you can use the Software Distribution standard variables, or wild cards and matching indicators to specify a file name, as described below:

- The * wild card selects all files in the specified path.
- The '*.extension' structure selects those files having the same extension.
- The $(MAX) or $(MIN) variables select one file in a set of files by defining, respectively, the highest or lowest value in the set.
- The $(ep_label) variable retrieves or deletes files with similar names stored on different endpoints by substituting the label of the endpoint, for example 'data.*endpoint_name*.txt' where *endpoint_name* is the label of the endpoint. It can also be used to send multiple files with similar names to different endpoints by substituting the label of the endpoint. In this case, each endpoint receives only

the file named with its label. For more information on this usage of the $(ep_label) variable, see “Sending Multiple Files” on page 242.

The **-R** option is not supported with matching indicators. For more information about Software Distribution variables, see “Using Variables” on page 5.

On UNIX systems, enter wild cards and matching indicators between single quotation marks, or precede them with a backslash, as described in the following examples:

```
wspmvdata -s @lab78040 -t @endpt1, @endpt2, @endpt3 '*.*'
wspmvdata -t @lab78040 -s @endpt1, @endpt2, @endpt3 sales.\$\(ep_label\).txt
```

For more information about using the wild card and the matching indicators, refer to the *User's Guide for Software Distribution*.

Scripts for Pre- and Post-processing: Using the **–r** argument, you can specify scripts for pre- and post-processing on the origin and destination systems. For example, you can include .exe, .com, or .pl programs you have written to perform tasks before and after moving data.

Note: If you specify a script created in a language that is not native to the operating system installed on the origin or destination system, you must specify the path to the application that runs the script on the origin or destination system, as described in the following example:

```
wspmvdata -s @lab133049-w2k -P sp:/wtd_tmp/source -P tp:/wtd_tmp/target
-t @lab133148-w2003 -r spre:"c:/tools/applications/perl/perl.exe
/wtd_tmp/target/script1.pl"
-r tpost:/wtd_tmp/target/test.exe data.txt"
```

In this case, the origin pre-script script1.pl is to be performed on a Windows origin system, therefore the path to the perl executable must be specified with the **-r spre** option.

These scripts define pre- and post-processing tasks on the origin and destination systems between which you want to move data. Up to four scripts can be invoked.

The following list shows the sequence of scripts for send operations:

1. Origin pre-processing script on the origin system.
2. Destination pre-processing script on each endpoint.
3. Destination post-processing script on each endpoint.
4. Origin post-processing script on the origin system.

The following list shows the sequence of scripts for retrieve operations:

1. Destination pre-processing script on each endpoint.
2. Origin pre-processing script on the origin system.
3. Destination post-processing script on each endpoint.
4. Origin post-processing script on the origin system.

A delete operation does not have an origin. The destination pre- and post-processing scripts run on the endpoints.

You can also move data from one endpoint to multiple endpoints. The sequence in which the scripts run in a send operation from endpoint to endpoint is as follows:

1. The pre-processing script runs on the origin system, which is the endpoint that was specified in the command.
2. A post-processing script runs on the origin system, which is the endpoint that was specified in the command.
3. A pre-processing script runs on each destination system. The destination systems for a send operation are endpoints.
4. A post-processing script runs on each destination system. The destination systems for a send operation are endpoints.

In all pre- and post-processing scripts, there is a set of predefined parameters. The following list shows the parameters and the values assigned to each at run time.

Parameter 1 Operation Type
Send, Retrieve, Delete

Parameter 2 Location Type
EP_SCRIPT, SH_SCRIPT

Parameter 3 Timing Type
PRE_SCRIPT, POST_SCRIPT

Parameter 4 Data File
Fully qualified file name. When using this parameter in a post-processing script at destination during a recursive retrieve operation, the value assigned to the parameter is the destination directory only, not the file name, as multiple files are being retrieved.

Parameter 5 Endpoint Label
Unique endpoint identifier. This parameter is only available for the post-processing script on the source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed.

Parameter 6 Endpoint Result
Result of the operation on the endpoint. Possible results are **0** (success) and **1** (failure). This parameter is only available for the post-processing script on the source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed.

Note: The **Endpoint Result** parameter allows you to condition the execution of the post-processing script on the source host to the result of the operation on the endpoint, so that, for example, the script is not run if the operation on the endpoint has not been successful.

Note: If you are writing a post-processing script for use on Windows platforms, you must include code to deal with any errors caused by the file being locked.

This situation can occur when an identical file, in name and content, already exists on the target system and is locked at the distribution time. In such a case, the data moving operation does not fail with ″file locked″, because it does not attempt to replace the file, since there are no changes.

As the operation has not failed, the post-processing script will run and must be able to deal with a locked file.

Example: The following command includes the script merge.sh as a post-processing script on the target system:

```
wspmvdata -t @lab15124 -s @lab67135-w98,@lab15180-2000
 -r tpost:/usr/sd/scripts/merge.sh /usr/sd/source/data.txt
```

The destination system for this command is a source host and the source list includes two endpoints. The purpose of the merge.sh script is to create a single file on the source host system by merging the files that have been retrieved from the endpoints. The merge.sh script is performed as a post-processing script on the source host after the files have been retrieved from the specified endpoints.

```
#!/bin/sh
#=================================================
CM_OPERATION=$1
LOCATION=$2
PRE_POST=$3
DATA_FILE=$4
print "CM Operation:" $CM_OPERATION > /usr/sd/scripts/merge.out;
print "Location:" $LOCATION >> /usr/sd/scripts/merge.out;
print "Pre-post:" $PRE_POST >> /usr/sd/scripts/merge.out;
print "File Name" $DATA_FILE >>/usr/sd/scripts/merge.out;
print "======================================" >>/usr/sd/scripts/merge.file;
print "=FILE merged: $DATA_FILE at: `date`="          >>/usr/sd/scripts/merge.file;
print "======================================" >>/usr/sd/scripts/merge.file;
cat $DATA_FILE >> /usr/sd/scripts/merge.file;
print "Error level is:" $? >> /usr/sd/scripts/merge.out;
exit $?
```

When the merge.sh script runs, the fixed parameters are set as follows:

$1 Retrieve

$2 SH_SCRIPT

$3 POST_SCRIPT

$4 /usr/sd/source/*<endpoint name>_<distributionID>_<timestamp>*

Note: A single directory is created on the source host for each endpoint. This ensures that each retrieved file is stored in a unique directory on the source host.

The script writes these values and any errors to an output file and appends the contents of the data file to the file /usr/sd/scripts/merge.file.

Sending Multiple Files: When you need to send several different files with similar names to different endpoints and each endpoint must receive only a specific file, you can use the $(ep_label) variable in the source file name. The $(ep_label) variable replaces the label of the endpoint.

The $(ep_label) variable can be used only to send files from managed nodes to endpoints, and not from endpoints to other endpoints.

The $(ep_label) variable is then resolved on each endpoint and the file named with the endpoint label is installed on the corresponding endpoint.

When you perform a send operation using this variable, an internal software package is created in the *product_dir* on the source host, that is a Tivoli managed node, functioning as a gateway or a repeater, where Software Distribution is installed. This software package contains all the files to be sent to the endpoints and a condition for each file which specifies on which endpoint each file must be installed. The software package is then sent to the target endpoints where the variable is resolved and the files installed.

You can specify the maximum size for the software package by setting the **dms_send_max_spb_size** key with the **wswdcfg** command. The default value for this key is 10,000 kilobytes. You can set this value to any integer equal to or lower than two gigabytes, which is the maximum size for a software package. The value defined on the Tivoli server is applied to the entire region, irrespective of the values defined on the source hosts, if any. For more information on the **wswdcfg** command, see "wswdcfg" on page 246. Note that an amount of space at least equal to the value you specify must be available in the *product_dir* on the source host for the package to be created.

To calculate the precise value for the **dms_send_max_spb_size** key, you need to consider the total size of the files to be sent plus 2 kilobytes for each endpoint.

If you are working with interconnected regions, you must perform the following operations when sending multiple files:

- On the source host append the region name preceded by a pound (#) sign to the endpoint name of the files to be sent to endpoints outside the Tivoli region where the source host is located.
- From the command line, append the region name preceded by a pound (#) sign to the target endpoint name when specifying the **-t** option. This procedure applies only to endpoints with duplicate labels.

This behavior allows you to manage endpoints with duplicate labels within interconnected regions.

The following command sends files `data.ep1#sales-region.txt` to endpoint `ep1#sales-region`, file `data.ep1#resources-region.txt` to endpoint `ep1#resources-region`, and file `data.lab132782-ep.txt` to endpoint `lab132782-ep`, registered to the same region where the source host is located.

```
wspmvdata -s @yoursourcehost -t @ep1#sales-region, @ep1#resources-region,
@lab132782-ep -P sp:c:\source\ -P tp:c:\target data.$(ep_label).txt
```

When specifiyng the distribution list the pound (#) sign and region name must be specified only when the endpoint label is duplicate between one or more endpoints. Note that files `data.ep1#sales-region.txt`, `data.ep1#resources-region.txt`, and file `data.lab132782-ep.txt` must be present on the source host, otherwise the operation is not performed because the **-F** option has not been specified.

To determine the region to which the specified endpoint belongs, type the following two commands on the Tivoli server:

```
eid=`wlookup -r Endpoint endpoint_name | awk -F'.' '{print $1}'`
ep_region=`wlsconn | grep $eid | awk '{print $2}'`
```

where

endpoint_name
: is the name of the endpoint whose region name you are trying to determine.

The results of the commands are returned to standard output. If the output returned is empty, the endpoint belongs to the region where the command was launched. If you have a large number of endpoints, you can insert this command in a script file. On Windows systems, these commands must be run in a bash shell. For more information on the **wlookup** and **wlsconn** commands, refer to *Tivoli Management Framework: Reference Manual*.

In the DataMovingRequests.1.log file, the information concerning the distributions to interconnected regions is logged according to the following criteria:

- The names of the origin and destination files are specified with the $(ep_label) variable.
- The name of the endpoint which received the distribution is logged before the **Distribution ID:** keyword. Use this value to determine which endpoint received the distribution.
- The region name preceded by a pound (#) sign is appended to the names of the endpoints outside the Tivoli region where the source host is located.

Authorization

admin, senior, or super

Return Values

The **wspmvdata** command returns one of the following:

0 Indicates that **wspmvdata** started successfully.

-1 Indicates that **wspmvdata** failed due to an error.

Examples

1. The following command sends file /data/file1 from the origin system sh1 to a list of destination systems. Code translation is required and a post transmission script, epprocess.sh, is to be executed on the destination systems.

   ```
   wspmvdata -s @sh1 -t @ep1,@ep2,@ep3 -r
   tpost:/scripts/epprocess.sh -c /data/file1
   ```

2. The following command runs a pre-processing script called export_file.sh on the pi003-ept, pi006-ept endpoints to extract data. The extracted data is saved in a file called trans. The trans file is stored in the /sales directory on each endpoint. Afterwards the trans files are retrieved from the endpoints and stored on the source host system, in a sub-directory within the /data/sales destination directory:

   ```
   wspmvdata -s @pi003-ept,@pi006-ept -t @centoff -r tpost:/tmp/import_file.sh
   -r spre:/tmp/export_file.sh -P sp:/sales -P tp:/data/sales trans
   ```

   Note: The destination directory (/data/sales) on the system is not created with this command, so it must be created beforehand.

   When the operation is completed the trans file is stored on the source host system under the following paths:

   ```
   data/sales/pi003-ept_14614660071043934511_20030130144803/trans
   data/sales/pi006-ept_14614660071043934511_20030130144803/trans
   ```

3. The following command sends file file1 from the /data directory on the origin system sh1 to a list of destination systems. The file is to be transferred to a location on the destination systems that is represented by the variable $staging_dir.

   ```
   wspmvdata -s @sh1 -P sp:/data -t @ep1,@ep2,@ep3 -P tp:$staging_dir file1
   ```

4. The following command retrieves the file file1 from the directory represented by the variable $temp on each of the endpoints specified in the origin list and transfers the retrieved files to the destination system sh1, where each is saved with a unique name in the directory represented by the variable $temp. The command specifies a pre-transmission task to export the file on each origin system and a post-transmission task to import the retrieved files on the destination system.

   ```
   wspmvdata -s @ep1,@ep2,@ep3 -t @sh1 -r tpost:/scripts/import.sh
   -r spre:/scripts/export.sh $temp/file1
   ```

5. The following command deletes the file called test stored in the /temp directory on the b1, b2 systems if the software SW_Package version 2 is in removed state or has never been installed.

   ```
   wspmvdata -t @b1,@b2 -S @SW_Package^2:R -d /temp/test
   ```

6. The following command sends file plist from the origin system centoff to a list of destination systems and runs a post -transfer task on the destination systems:

   ```
   wspmvdata -s @centoff -t @pi003-ept, @pi006-ept -r tpost:/tmp/importpl.sh
   -P sp:/price -P tp:/data/sales plist
   ```

7. The following command runs a pre-process task on the origin systems pi003-ept and pi006-ept, retrieves file trans from the origin systems, sends it to the destination system centoff, and runs a post-transfer task on the destination system:

   ```
   wspmvdata -s @pi003-ept,@pi006-ept -t @centoff -r tpost:\tmp\importtrans.sh
    -r spre:\tmp\exporttran.sh -P sp:/sales -P tp:/data/sales trans
   ```

8. The following command selects and moves the file on the source directory on the centoff system c:/tmp with prefix *sales.data* and suffix *transactions.txt* and with the highest value; that is the most recent date within the set to the target systems b1, b2, b3:

   ```
   wspmvdata -s @centoff -t @b1,@b2,@b3 -P sp:c:/tmp
   -P tp:/tmp sales.data.$(MAX).transactions.txt
   ```

   To enter the same command on a UNIX system, enter the string with the wild card between single quotation marks or precede it with a backslash, as follows:

   ```
   wspmvdata -s @centoff -t @b1,@b2,@b3 -P sp:c:/tmp
   -P tp:/tmp 'sales.data.\$\(MAX\).transactions.txt'
   ```

   or

   ```
   wspmvdata -s @centoff -t @b1,@b2,@b3 -P sp:c:/tmp
   -P tp:/tmp sales.data.\$(MAX).transactions.txt
   ```

9. The following command retrieves all files in the directory c:/tmp on the endpoints b1, b2, b3 with prefix *sales.data* and suffix *transactions* replacing the $(ep_label) variable with the actual name of the endpoint and sends them to the destination system centoff:

   ```
   wspmvdata -s @b1,@b2,@b3 -t @centoff -P sp:c:/tmp
   -P tp:/tmp sales.data.$(ep_label).transactions
   ```

10. The following command sends files located in the directory c:\temp on the source host juno to each endpoint, based on the endpoint name. Each endpoint receives only the file containing its label as part of the file name. If any of the files are not present on the source host, the **-F** option causes the operation to be performed on the remaining endpoints:

    ```
    wspmvdata -s  @juno -t @ep1,@ep2 -P sp:c:\temp -P tp:d:\temp -F
    price.$(ep_label).txt
    ```

See Also

None.

wswdcfg

Changes the managed node (with Software Distribution installed) and source host settings.

Syntax

wswdcfg [[**-r** *region_name*] - **h** *hostname*] **–s**

wswdcfg [[**-r** *region_name*] - **h** *hostname*] **–s** [*key*]

wswdcfg[[**-r** *region_name*] - **h** *hostname*] **–s** [*key*[=*value*]]

wswdcfg [[**-r** *region_name*] - **h** *hostname*] **-d** *key*

wswdcfg - **s datamoving_source_host**=*managed_node_name*

Description

Options

–r *region_name*
Specifies the region where the managed node (with Software Distribution installed) or the source host reside. If no region is specified, the current region is assumed.

–h *host name*
Specifies the host name of the managed node (with Software Distribution installed) or of the source host. If you omit this parameter, the local host name is used.

–s *key=value*
Sets a custom key and its value, or allows you to define existing variables and their values. Specifying the **wswdcfg -s** command without the *key* argument, displays all keys with the corresponding settings currently used. Specifying the **wswdcfg -s** *key* command without a value, displays the value set for the key. Specifying the key with a value, sets the key to the specified value. Default keys are:

autopack_dir	Identifies the directory name where you want to temporarily store the results of the Autopack snapshots. The default is *product_directory*\autopack.
autoscan_active	Specifies whether a scan operation is automatically performed on the targets. The results of the scan operation update the COMPUTER table. Possible values are **y** and **n**; the default value is **y**.
backup_dir	Identifies the directory where the software packages are backed up.
continue_on_invalid_targets	Specifies whether a distribution must be stopped when invalid targets are encountered. Possible values are **y** and **n**; the default value is **n**.
datamoving_source_host	Specifies the default source host. If no value is specified, the Tivoli server is used as source host.
disable_remove_not_installed	Specifies whether you support the option

to enable and disable the capability to remove software packages that have not yet been installed. Possible values are **y** and **n**; the default value is **n**. To disable the capability to remove software package that have not been installed, run the following command:

```
wswdcfg - s disable_remove_not_installed = y
```

dms_send_max_spb_size

Specifies the maximum size for a software package to be created and sent to multiple endpoints using data moving. For more information on this data moving feature, see “Sending Multiple Files” on page 242. The default value for this key is 10,000 kilobytes. You can set this value to any integer equal to or lower than two gigabytes, which is the maximum size for a software package. Note that an amount of space at least equal to the value you specify must be available in the *product_dir* on the source host for the package to be created.

ep_trace_level

Enables trace logging on the endpoint and specifies the trace level. Possible values are:

- 0 (none)
- 1 (fatal)
- 2 (error)
- 3 (warning)
- 4 (information)
- 5 (verbose)

The default value is 0.

ep_trace_size

Specifies the size of the trace file. The default value is 1 000 000 bytes.

ep_trace_override_local_settings

Specifies whether the setting defined on the source host for trace logging overrides the endpoint local setting. Possible values are **0** and **1**. The default value is **1**, which means that the endpoint local settings are overridden.

fail_if_no_targets

Specifies whether the distribution operation fails in case no targets are specified. By default, if no targets are specified, the distribution is submitted to all subscribers of the profile manager in which the software package resides. By setting this key to **y**, a warning message is displayed if no targets are specified, and the distribution is not submitted. Supported values are **y** and **n**. The default value is **n**.

how_create_ep_sections

Specifies how to create the section for an endpoint in the swdis.ini file when it does

not exist, in particular when an endpoint has been renamed. Possible values are **clone_mobile** and **clone_mobile_or_first_ep_name**. The default value is **clone_mobile**, which means that the swdis.ini section corresponding to the *endpoint label* is created by cloning the contents of the [#MOBILE] section if it exists. If the [#MOBILE] section does not exist a new product_dir is created in the swdis.ini file and a new path is created in the $(system_dir). If you specify **clone_mobile_or_first_ep_name** a new endpoint section is created by cloning the [#MOBILE] section if it exists, or otherwise by cloning the first *endpoint label* section if that exists. If neither of the above sections exist then a new *endpoint label* section is created based on a new product_dir.

import_libraries Identifies the path name of the library that contains the object to be added to the software package. The path cannot contain wild cards.

inventory_rim_name Identifies the name of the RIM object. The default name is inv_query.

message_dir_usable_quota Identifies the percentage of disk space to the message directory, to avoid corruption of the message file due to insufficient disk space during the reporting phase. The limit is customizable. The default is 100%. Each time the limit specified is reached, the same exception is caught as the thread limitator.

nm_restart_timeout Identifies the timeout value after which a check for new messages with higher priority than the current ones is performed. If the timeout is set to a negative value, no check is performed. The default timeout value is 180 seconds.

notify_ext_directly{*plan_name***}** Specifies whether the reports for the Software Distribution activities contained in an activity plan are notified to Activity Planner directly without using notification manager. You can also skip validation operations against the Inventory database for activities submitted by Software Distribution. Possible values are **y** and **n**. If you specify **y**, the Software Distribution reports for the specified plan are skipped by notification manager. If you specify **a**, the Software Distribution reports for the specified plan are skipped by notification

manager and Software Distribution activities are not validated against the Inventory database. Use the *plan_name* variable to specify the name of plan. When specifying this variable, you can use wildcards. You can specify more than one of these variables. If you use variables specifying conflicting plan names, for example `notify_ext_directly{MyPlan*}=y` and`notify_ext_directly{MyPlan*}=a`, the first statement you entered is used.

product_dir Identifies a parent directory where Software Distribution data, such as catalogs, messages, traces, and backup packages, are stored.

profile_dir Identifies the user profile directory of the user currently logged on (for example, C:\WINNT\Profiles\UserName).

report_threads_limit Identifies the maximum number of threads the Software Distribution server can use for processing reports. The default value is 60.

split_dm_log Modifies the creation parameters of the data moving log file (**DataMovingRequests.1.log**). If you set this option to **y**, the data moving log file is split into a single file for each data moving operation. The resulting files are named according to the following standard: `DataMovingRequest.`*`DIST_ID`*`.log` where *DIST_ID* is the last portion of the MDist 2 distribution ID. The files are saved in the `DataMovingRequests.dir` folder created in the working directory, defined with the **working_dir** key with the **wswdcfg** command. For more information on this command, see “wswdcfg” on page 246. If you set this option to **n**, the default behaviour applies. For more information on the data moving log file, see “wspmvdata” on page 229. The default value is **n**. Supported values are **y** and **n**.

staging_dir Identifies the working directory of the managed node.

stop_on_prog_hang Modifies the behavior of the Software Distribution engine in case of user programs running when communication with the gateway is interrupted for any reason. The default behavior causes the distribution to end with a warning. If you set this key to **n**, the user program is completed and the interrupted distribution can restart from the last valid checkpoint when communication with the gateway is

restored. If the user program hangs, though, the distribution hangs until its expiration date is reached. Supported values are **y** and **n**. The default value is **y**.

trace_level Specifies the trace level. Possible values are:
- 0 (none)
- 1 (fatal)
- 2 (error)
- 3 (warning)
- 4 (information)
- 5 (verbose)

The default value is 0.

trace_size Specifies the size of the trace file. The default value is 1 000 000 bytes.

user_file_variables Identifies the location of the swdis.var file that contains user-file variables. The default location is the product directory of the endpoint.

working_dir Identifies the working directory where the main persistent data is stored.

–d *key* Deletes the specified *key*.

Authorization

senior

Return Values

The **wswdcfg** command returns one of the following:

0 Indicates that **wswdcfg** started successfully.

-1 Indicates that **wswdcfg** failed due to an error.

Examples

1. To display the source host local settings, enter the following command:

```
D:\>wswdcfg -h mycomputer -s
```

The following example output is displayed:

```
product_dir=C:\Tivoli\bin\swdis
working_dir=C:\Tivoli\bin\swdis\work
backup_dir=C:\Tivoli\bin\swdis\backup
profile_dir=C:\Tivoli\bin\swdis\work\profiles
trace_level=5
trace_size=1000000
report_threads_limit=60
nm_restart_timeout=180
inventory_rim_name=inv_query
autopack_dir=C:\Tivoli\bin\swdis\autopack
staging_dir=Tivoli\bin\swdis\service
user_file_variables=C:\Tivoli\bin\swdis\swdis.var
autoscan_active=y
continue_on_invalid_targets=n
fail_if_no_targets=n
stop_on_prog_hang=y
import_libraries=spd,libscimp
split_dm_log=n
dms_send_max_spb_size=10000
```

2. To change the value of the trace_level key, enter the following command:

   ```
   D:\>wswdcfg -h mycomputer -s trace_level=5
   ```

   To delete the custom my_key key, enter the following command:

   ```
   wswdcfg -d my_key
   ```

3. To display the value of the inventory_rim_name key, enter the following command:

   ```
   wswdcfg -s inventory_rim_name
   ```

   The following example output is displayed:

   ```
   inv_query
   ```

4. To specify that submitted activities for all plans whose name start with MyPlan in region Region1 are validated against the Activity Planner database, enter the following command:

   ```
   wswdcfg -r Region1 -h hostname1 -s notify_ext_directly{MyPlan*}=y
   ```

See Also

None.

wswdmgr

- Enables and disables the integration between Software Distribution and Inventory, and between Software Distribution and Tivoli Enterprise Console.
- This command also allows the user to define default values for distribution options in the specified policy regions. When defining default values on interconnected Tivoli Management Regions, you can only set default values on the region where the command is launched.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wswdmgr [- **r** *region_name*] **–s**

wswdmgr [- **r** *region_name*] **–s** [*key*]

wswdmgr [- **r** *region_name*] **–s** [*key* [=*value*]]

wswdmgr [- **r** *region_name*] **-n** {start | stop}

Syntax

wswdmgr -l

wswdmgr [- **p** *policy_region*] **-c** *key=value...*

wswdmgr [- **p** *policy_region*] **-d** [*key*]

wswdmgr [- **p** *policy_region*] **-e** *key*

Description

The **wswdmgr** command partly substitutes the **wswsprim** command, and allows the user to manage default values for distribution options.

Options

–r *region_name*

Specifies the Tivoli Management region on which the operation is performed. If no region is specified, the current region is assumed.

–s *key=true | false*

Specifying the **wswdmgr -s** command without the *key* argument, displays all keys with the corresponding settings currently used. Possible values are **true** and **false**. Specifying the **wswdmgr -s** *key* command without the **true** or **false** value, displays the value set for the key. Specifying the key with the **true** or **false** value, sets the key to the specified value. Valid keys are:

is_swd_inv_enabled	Used with the Inventory **wsetinvswd** command to enable the transfer of signature data from Inventory to Software Distribution to update the sig_sp_map table. You also need to enable InventoryManager using the **wsetinvswd** command. For more information about this command, see *User's Guide for Inventory*. The default value is **true**.

is_swd_tec_enabled Sends Software Distribution events to the Tivoli Enterprise Console. The default value is **true**.

is_hdb_enabled Stores and updates the historical database information in the Inventory database. The default value is **true**.

is_cmstatus_enabled Stores and updates change management status information in the Software Distribution tables within the Inventory database. The default value is **true**.

Note: If *is_cmstatus_enabled* is set to **false**, the product forces *is_swd_inv_enabled* and *is_hdb_enabled* to **false**.

is_swd_mdist_enabled Sends information to the MDist 2 database and console. The default value is **true**.

-n {start | stop}
Starts and stops the Software Distribution message queue.

-l Displays all available keys for distribution options with the related information. Use the **-c** option for setting values for these options. The default values for the options listed below are defined in MDist:

- deadline
- execute_timeout
- notify_interval
- send_timeout

To view the default values for these options, use the **wmdist** command. For more information on this command, refer to *Tivoli Management Framework: Reference Manual*.

-p *policy_region*
Specifies the policy region to which the default values are applied. If no policy region is specified, the command is applied to all policy regions in the Tivoli Management Region.

-c *key=value*
Specifies the distribution options for the specified policy region, as follows:

label Specifies a description string for the distribution. The default value varies depending on the operation to be performed. Supported values are as follows:

"name.version (operation)"
This convention applies to the following operations:

- install
- undo
- remove
- accept
- commit
- verify
- load
- unload

operation (operation)
This convention applies to the following operations:

- wsyncsp

- wsetsps

filename **(***operation***)**

This convention applies to the following operations:

- send
- retrieve
- delete

priority

Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval

Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout

Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout

Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

disposable

Specifies if data must be removed from the repeater after distribution. Valid values are **y** (yes) or **n** (no). The default value is **n** for a software package block (built format). This attribute is not available for software packages (not-built format).

Note: For a software package block, with the exclusion of delta software package blocks, disposable and from_depot attributes cannot both be set to **y**.

deadline

Specifies the number of hours, starting from the time the distribution is submitted, after which a distribution expires. Valid values are all integers greater than zero. For example if you schedule a distribution to start at 16:00 and the operation is executed at 15:00, the deadline is computed starting from 15:00 and not at 16:00.

depot_image_dir
Directory on the depot where the product images are to be stored. The contents of this directory can be copied to a CD ROM or to a file server, which can then be used as the source for an installation. This option is used only for the load operation.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@[hostname]:absolute_path
```

The maximum supported length for the message contained in the file is 256 characters. All characters exceeding this limit are ignored without warning. If you specify a file name, the content of the file is resolved when you launch the command. If the file is modified or deleted afterwards, this modification has no effect on the message. If no hostname is specified, the Tivoli server is assumed. If you define a value for this attribute, the hidden attribute must be set to **n**.

mandatory_date
Specifies the number of hours, starting from the time the distribution starts, by which the operation must be completed on endpoint or mobile targets. The operation can be deferred up to this date. When the date is reached, the operation is automatically performed on all outstanding endpoint or mobile targets. Use this option to set the distribution as mandatory. Valid values are all integers greater than zero. A check is performed to ensure the default value for the deadline attribute is greater than the mandatory_date value.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

from_depot
Specifies that the software package to be installed resides on the repeater depot, rather than on the source host. In this case, the package was previously loaded on the gateway using the**wldsp** command. Valid values are **y** (yes) or **n** (no). The default value for a software package block (built format) is **n**. This attribute is not available for software packages (not-built format).

Note: For a software package block, with the exclusion of delta software package blocks, the following limitations apply:
- disposable attribute must be set to **n**

- from_cd attribute must be set to **n**
- from_fileserver attribute must be set to **n**

from_fileserver
Specifies that the images referenced in the software package are to be retrieved from a file server. File servers must be configured if this argument is used.

Notes:

1. You cannot specify the **from_fileserver** and **from_depot** attributes in the same command. However, you can specify the **from_fileserver** and **from_cd** attributes as alternative locations for the images.
2. Before you can use MDist 2 to install a distribution from a file server, the following file must exist on the target system: $LCF_DATDIR/remote.dir. Refer to *Tivoli Management Framework: User's Guide* for more information about the remote.dir file.
3. This option is not supported for Novell Netware endpoints.

from_cd
Specifies that the images referenced in the software package are to be retrieved from the CD. If you use this option, you are prompted to insert the CD.

Note: You cannot specify the **from_cd** and **from_depot** attributes in the same command. However, you can specify **from_cd** and **from_fileserver** as alternative locations for the software package.

escalate_date_*n*
Specifies the number of hours, starting from the time the distribution starts, after which a reminder message must be sent to mobile targets that have not yet completed the operation. Valid values are all integers greater than zero.

The *n* represents a number, 0 through 9, so that a sequence of up to 10 messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@[hostname]:absolute_path
```

The maximum supported length for the message contained in the file is 256 characters. All characters exceeding this limit are ignored without warning. If you specify a file name, the content of the file is resolved when you launch the command. If the file is modified or deleted afterwards, this modification has no effect on the

message. If no hostname is specified, the Tivoli server is assumed. If you define a value for this attribute, the hidden attribute must be set to **n**.

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it. When this option is set to **y**, the hidden attribute must be set to **n**.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for the following attributes:
- distribution_note
- mandatory_date
- enable_disconnected
- escalate_date
- escalate_msg

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this attribute to **t** enables the **retry_unicast** attribute. If the **retry_unicast** attribute is set to **t**, you cannot change **is_multicast** to **f**.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the

distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

The maximum supported length for the message contained in the file is 256 characters. All characters exceeding this limit are ignored without warning. If you specify a file name, the content of the file is resolved when you launch the command. If the file is modified or deleted afterwards, this modification has no effect on the message. If no hostname is specified, the Tivoli server is assumed.

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

-d [*key***]**
Displays all default values already set by the user for the distribution options. If a key is specified, the value set for the specified key is returned.

-e *key* Deletes the specified key and the related default value.

Authorization

senior, super or admin

When using the **-c**, **-d**, and **-e** keys, which define default values for distribution options, you need to have the authorization listed above for the specified policy region, or, if you do not specify a policy region, you need to have the authorization listed above for the Tivoli Management Region.

Return Values

The **wswdmgr** command returns one of the following:

0 Indicates that **wswdmgr** started successfully.

-1 Indicates that **wswdmgr** failed due to an error.

Examples

1. To stop sending the Software Distribution events to Tivoli Enterprise Console, enter the following command:

   ```
   wswdmgr -s is_swd_tec_enabled=false
   ```

2. To update the Inventory configuration repository with the historical database information, enter the following command:

   ```
   wswdmgr -s is_hdb_enabled=true
   ```

3. To update the Inventory configuration repository with the change management status information, enter the following command:

   ```
   wswdmgr -s is_cmstatus_enabled=true
   ```

4. To find out the current integration settings for region test-region, enter the following command:

   ```
   wswdmgr -r test-region -s
   ```

   The following output is displayed:

   ```
   is_swd_inv_enabled=true
   is_cmstatus_enabled=true
   is_swd_tec_enabled=true
   is_hdb_enabled=true
   ```

5. To stop the Software Distribution message queue for region my-region, enter the following command:

   ```
   wswdmgr -r my-region -n stop
   ```

6. To discover the value of the is_cmstatus_enabled key for the current region, enter the following command:

   ```
   wswdmgr -s is_cmstatus_enabled
   ```

7. To set the mandatory date for a distribution to two hours after the distribution starts and to force the distribution process on its targets, enter the following command:

   ```
   wswdmgr -c mandatory_date=2 -c force_mandatory=true
   ```

8. To set the distribution to expire after two hours after the command is launched, enter the following command:

   ```
   wswdmgr -p mypolicyregion -c send_timeout=200 -c deadline=2
   ```

9. To specify a description string for the distribution for software package test^1.0, enter the following command:

   ```
    wsdmgr -c label="test^1.0 (remove)"
   ```

See Also

wsetinvswd in the *User's Guide for Inventory*, **wldsp**

wswsprim

Enables and disables the historical database and change management status features of Software Distribution Historical Database. Since this command is a bash script that must be run in the bash environment, precede the command with the string **sh**.

Syntax

wswsprim {**–c** | **–d** | **–s** | **–v**}

Description

The **wswdmgr** command substitutes the **wswsprim** command. Although the **wswsprim** command is available in this release of the product, it will not be supported in future releases. Use instead the **wswdmgr** command, which replaces it.

Options

–c Enables the historical database and change management status features.

–d Disables the historical database and change management status features.

–s Enables only the change management status feature (the historical database feature is disabled after the execution of this command option).

–v Gives the status of the historical database and change management status features.

Authorization

senior or super

Return Values

The **wswsprim** command returns one of the following:

0 Indicates that **wswsprim** started successfully.

-1 Indicates that **wswsprim** failed due to an error.

Examples

1. To enable the historical database and change management status features on a Windows platform, enter the following command:

   ```
   sh wswsprim -c
   ```

2. To verify whether these features have been enabled, enter the following command:

   ```
   wswsprim -v
   ```

   The following output is displayed:

   ```
   -Historical database support: enabled
   -Change management status support: enabled
   ```

See Also

None.

wsyncsp

Synchronizes the software-package status information on the server with that for software packages on specified endpoints.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wsyncsp [**–f**] [**–l** *mdist2_token=value*]... **–T** *targets_file... targets...*

wsyncsp [**–f**] [**–l** *mdist2_token=value*]... **–T** *targets_file ...*

wsyncsp [**–f**] [**–l** *mdist2_token=value*]... *targets...*

Description

The **wsyncsp** command triggers a lenient and dataless distribution to the specified targets. It returns information about changed software packages to the server.

The information about status of software, stored on the server, can get out of step with the real situation on the endpoints, for example, when the disconnected target command **wdinstsp** is run to install a software package on an endpoint.

You can use the **–f** argument if you want the command to return information about all software, changed and unchanged. This could be used, for example, if the cm_status information in the Inventory database was lost.

Details of the command and the information returned are stored in the log file wsyncsp.log, which is stored in the working directory. Information is also stored in the logs of the individual software package objects that are affected by the command.

Options

–f Specifies information about unchanged software packages should be included.

–l *mdist2_token=value*
Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *sync (sync)*.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

hidden
Valid values are **y** (hidden) and **n** (not hidden).You can set this argument to **y** to ensure that the synchronization is not rejected at the target.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

–T *targets_file*
Name of a file containing a list of the targets for the operation. Targets can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify targets separately on the command line (see the description of the *targets* attribute for this command). If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the wsyncsp.log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

targets...
The names of the target systems or profile manager to be synchronized. You can specify subscribers individually, using this argument, specify a file, using the **–T** argument, or both. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the wsyncsp.log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wsyncsp** command returns one of the following:

0 Indicates that **wsyncsp** started successfully.

-1 Indicates that **wsyncsp** failed due to an error.

Examples

To synchronize the information on the server with the status of software packages on the endpoints ep1, ep2, and ep3, enter the following command:

```
wsyncsp @ep1 @ep2 @ep3
```

Or:

```
wsyncsp -T targetfile
```

Where the content of the file `targetfile` is @ep1 @ep2 @ep3.

See Also

None.

wuldsp

Unloads a software package from a selected group of target subscribers.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wuldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage:**]*spobj_name*

wuldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wuldsp [**–d** @[**SoftwarePackage:**] *spobj_name*] [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [@[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wuldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage:**]*spobj_name*

wuldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wuldsp [[**–i** | **–f** | **–I**] [**–l** *mdist2_token=value...*]] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

Description

Options

–d *spobj_name*
: Specifies the name and version of the base package to which a delta file is to be applied. If this argument is specified, the software package includes changes to an existing package, rather than a replacement.

–f
: Specifies that the unload must be forced regardless of the state of the package on the repeater.

–i
: Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–I
: Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing an unload operation and the software package has already been unloaded on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–l *mdist2_token=value*
: Specifies the distribution options, as follows:

  label
  : Specifies a description string for the distribution. The default value is the string *"name.version*(**unload**)*"* where *name.version* indicates the software package name and version.

priority
Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval
Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout
Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures.This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

–T *subscribers_file*
Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is

specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see "Software Package Name and Version" on page 2.

subscribers...
The names of the target systems or profile manager from which the software package is to be unloaded. If no target systems are specified, the software package is unloaded from all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wuldsp** command returns one of the following:

0 Indicates that **wuldsp** started successfully.

-1 Indicates that **wuldsp** failed due to an error.

Examples

1. To unload software package @mypackage from the profile manager myprofile, enter the following command:

   ```
   wuldsp @mypackage @myprofile
   ```

2. To unload a delta software package @mychanges, which is to be applied to the base package @mypackage, from the profile manager myprofile, enter the following command:

   ```
   wuldsp -d @mypackage @mychanges @myprofile
   ```

See Also

- **wldsp**
- **wrpt** in the *Tivoli Management Framework: Reference Manual*
- **wmdist** in the *Tivoli Management Framework: Reference Manual*
- **wdepot** in the *Tivoli Management Framework: Reference Manual*

wundosp

Undoes an installed software package.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *n* | *o*] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *n* | *o*] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] [**–T** *subscribers_file*]... @[**SoftwarePackage:**]*spobj_name*

wundosp [**–i** [**–v**] | **–I**] [-**R**{ *y* | *n*}] [**–t** *y* | *n* | *o*] [**–l** *mdist2_token=value...*] [[**-X none | last | middle**] | [-**X first | both** [**-*Y*** *max_login_allowed*] [**-W**]]] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

Description

The **wundosp** command returns the system to its state prior to the execution of the previous operation. This command is used for objects for which the previous operation was run in undoable mode.

Options

–i Verifies whether the operation can be performed without submitting it. The process generates a list of targets on which the operation fails. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target.

–v Specifies that verbose logging is to be used. If verbose logging is not used, any failure in the distribution is indicated by a short message, for example:

```
ep1 - Failed dependency check
```

If verbose logging is used, a full description of the reason for failure is logged.

Use of the verbose logging option causes a significant increase in the size of the log file. Therefore, this option must be used with caution. This argument is only available if the **–i** argument is specified.

–I Verifies whether the operation can be performed and proceeds with the operation only on target systems that pass the verification. For example, if you are performing an undo operation and the software package has already been undone on a target system, the operation does not proceed on that target system. If this argument is not included, the operation cannot proceed on any destination if all destinations do not meet the requirements.

–t *y | n | o*
Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y | n | o | r*
Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see "Commit Operation" on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options.

-X {none | first | middle | last | both}
Use this option to define a set of software packages for which user login and shutdown operations can be disabled while the distribution is taking place. If you define a package as **first**, this package is the first in a series for which you can define these options. Define the other packages in the series as **middle** and the last package as **last**. A software package defined as **last** must exist for each software package defined as **first**. If the series consists of just one package, define this package as **both**, which means the software package is both first and last in the series. The default value is **none** which means user login and shutdown operations cannot be disabled.

-Y *max_login_allowed*
Use this option to specify whether users can log on to the workstation while a distribution is taking place. This setting can be defined only for software packages defined as **first** or **both**. It applies to software packages defined as **first**, **middle**, **last**, or **both**. Supported values are **0** (no login is allowed), **-1** (an unlimited number of logins is allowed), and any positive integer. If a login is performed while the distribution is taking place, the distribution is paused until the user performs a logoff.

-W Specifies that the user cannot perform a shutdown while a distribution is taking place. If the user attempts to perform a shutdown and the timeout is set to a value other than zero using the **Timeout** key, a dialog box is displayed on the endpoint listing the allowed operations and requesting the user to select one. The user can choose between performing a restart, a logoff, or a logoff and shutdown. The restart and logoff operations are performed immediately, while the shutdown is performed after the last distribution has completed. If the user does not respond to the

dialog within the allotted time, the default action is performed. The default action is logoff and shutdown.

–l *mdist2_token=value*

Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**undosp**)*"* where *name.version* indicates the software package name and version.

priority

Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval

Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout

Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout

Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline

The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note

Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date

Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target.

Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile

endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

–R *y | n*
Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–T *subscribers_file*
Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see “Software Package Name and Version” on page 2.

subscribers...
The names of the target systems or profile manager from which to undo the software package. If no target systems are specified, the software package is undone on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wundosp** command returns one of the following:

0 Indicates that **wundosp** started successfully.

-1 Indicates that **wundosp** failed due to an error.

Examples

To undo the previous operation performed on a software package object called fsys_test^1.0, enter the following command:

```
wundosp @fsys_test^1.0
```

See Also

- The **undo** option for **winstsp** and **wremovsp**
- **wrpt** in the *Tivoli Management Framework: Reference Manual*
- **wmdist** in the *Tivoli Management Framework: Reference Manual*

wversp

Verifies an installed software package.

This command specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

Syntax

wversp [**–l** *mdist2_token=value...*] [**–T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

wversp [**–l** *mdist2_token=value...*] [**–T** *subscribers_file*] @[**SoftwarePackage:**]*spobj_name*

wversp [**–l** *mdist2_token=value...*] @[**SoftwarePackage:**]*spobj_name* [*subscribers...*]

Description

The **wversp** command verifies that the operation executed on the target object is consistent with the installed package, that is, that the files have been successfully installed on the target system.

Note: If a package contains two commands that perform opposite operations, such as adding and removing the same object, one of the operations will fail verification.

Options

–l *mdist2_token=value*

Specifies the distribution options, as follows:

label Specifies a description string for the distribution. The default value is the string *"name.version* (**verify**), where *name.version* indicates the software package name and version.

priority

Specifies the priority level, which is the order in which distributions are handled by repeaters, either **h** (highest priority), **m** (medium priority), or **l** (low priority). The default value is **m** (medium priority). The priority level is the priority also used when logging information.

notify_interval

Specifies the notification interval key, which determines how often each repeater bundles the completed results and returns them to the application and distribution manager. This attribute is initially set using the **wmdist –s** command (the default value is 30 minutes) and is expressed as a positive integer representing minutes. You can override the **wmdist –s** setting by specifying a different value here.

send_timeout

Specifies the length of time a repeater will wait for a target system to receive a block of data. This timeout is used to detect network or endpoint failures. This attribute is initially set using the **wmdist –s** command (the default value is 300 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

execute_timeout
Specifies the length of time a repeater will wait for Software Distribution to return the result of a distribution after all the data has been sent. This timeout is used to detect network, endpoint, or script failures, such as a script running an infinite loop. This attribute is initially set using the **wmdist –s** command (the default value is 600 seconds) and is expressed as a positive integer representing seconds. You can override the **wmdist –s** setting by specifying a different value here.

deadline
The date on which a distribution expires, that is, when it fails for unavailable target systems, in the format *"mm/dd/yyyy hh:mm"*.

distribution_note
Specifies a message to be associated with a software package when it is distributed to mobile targets.

You can enter a text using the format:

```
distribution_note="message text"
```

You can specify a file using the format:

```
distribution_note=@filename
```

mandatory_date
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, by which the distribution must be made to an endpoint or mobile target. Distributions to endpoints or mobile targets can be deferred up to this date. When the date is reached, the package is automatically installed on all endpoints or mobile targets that have not yet accepted it. Use this option to set the distribution as mandatory.

force_mandatory
The setting of this argument controls the way in which mandatory distributions on mobile targets are treated once the mandatory date is passed.

If you specify **y** (the default value), the mandatory distribution is automatically started as soon as the mobile user connects.

If you specify **n**, the mobile user has the choice of not starting the mandatory distribution. However, the user will not be able to perform any other operations until the mandatory distribution has been performed.

escalate_date_*n*
Specifies a date, in the format *"mm/dd/yyyy hh:mm"*, on which a reminder message must be sent to mobile targets that have not yet completed the operation.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each escalation date must have an associated message, and there must be no gaps in the sequence.

escalate_msg_*n*
Specifies a message that must be sent to mobile targets that have not completed the operation by the associated escalation date.

The *n* represents a number, 0 through 9, so that a sequence of up to ten messages can be specified. Each message must have an associated date and there must be no gaps in the sequence.

You can enter a text using the format:

```
escalate_msg_n="message text"
```

You can specify a file using the format:

```
escalate_msg_n=@filename
```

enable_disconnected
Indicates whether disconnected operations are enabled. If you specify **y**, you have the option of downloading the software package to a depot and applying it later. If you specify **n**, you must apply the software package as soon as you download it.

hidden
For mobile targets, indicates whether the operation is to be hidden. Non-hidden operations on mobile targets can be deferred. Hidden operations cannot.

Valid values are **y** (hidden) and **n** (not hidden). If you set this argument to **y**, you must not set values for mandatory date, escalation dates, or escalation messages.

roam_endpoints
Indicates whether the operation defined in the command supports roaming endpoints. Setting this argument to **y**, indicates that the distribution is to be transferred to any gateway where the mobile endpoint connects. Setting this argument to **n**, indicates that once the package is queued at a gateway it cannot be transferred to another. The default value is **n**.

wake_on_lan
Indicates whether the operation sends a wake-on-lan message to trigger rebooting of systems that are not available at distribution time. Valid values are **y** (yes) and **n** (no).

is_multicast
Enables data broadcasting to multiple repeaters. Multicast sends only one distribution from the source to a group of targets simultaneously. Use this option where there is limited network bandwidth. Valid values are **t** (true) and **f** (false). Setting this token to **t** enables the **retry_unicast** token.

retry_unicast
Retransmits the distribution independently to each endpoint that failed to receive the original multicast distribution. This option can be used only if the **is_multicast** option is set to **t**.

enable_notification
Specifies whether the user should be notified of a distribution starting on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To specify the message text, use the **user_notification** option. Valid values are **y** and **n**. The default value is **n**.

allow_defer
Specifies whether the user should be allowed to defer the distribution. A user can defer the software distribution and, at the end of the defer timeout period, subsequently reject it or defer it again. Valid values are **y** and **n**. The default value is **y**.

allow_reject
Specifies whether the user should be allowed to reject the distribution. Valid values are **y** and **n**. The default value is **y**.

default_action
Specifies the default action to be performed on the user's machine in case the user is not logged on the machine, or is not physically present. Valid values are **accept**, **reject**, and **defer**. The default value is **accept**.

default_timeout
Specifies the interval of time the notification dialog is displayed. The default is 60 seconds. When the timeout period elapses, the default action is launched if the user is logged on. If the user is not logged on, the default action is launched immediately without a timeout period.

user_notification
Specifies the text to be sent with the distribution and displayed on the user's machine. The notification dialog containing the message text is displayed only on Windows platform endpoints. To enable this function, set **enable_notification** to **y**.

You can enter text using the following format:

```
user_notification="message text"
```

You can specify a file using the following format:

```
user_notification=@/test/download/filename
```

fail_unavail
Specifies whether the distribution fails on endpoints that cannot be reached for any reason. Supported values are true and false. The default value is false.

–T *subscribers_file*
Name of a file containing a list of the subscribers for the operation. Subscribers can be specified using either relative or absolute path names. You can specify one file name, multiple file names, or none at all. Multiple names should be separated by one or more blanks. Even if a file is specified, you can also specify subscribers separately on the command line (see the description of the *subscribers* attribute for this command). If neither the **–T** attribute nor *subscribers* is used, the operation is performed on all subscribers of the profile manager in which the software package resides. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Note: If any of the specified files are empty, the operation fails.

spobj_name
Name and version of the software package object registered in the Tivoli Name Registry. For information about the format of the name and version string, see "Software Package Name and Version" on page 2.

subscribers...
The names of the target systems or profile manager where the software package is to be verified. If no target systems are specified, the software

package is verified on all subscribers of the profile manager in which the software package resides. You can specify either one target host, multiple hosts, or none at all. If one or more subscribers are not valid, the operation fails on those subscribers, and continues on the other subscribers. Information about the subscribers on which the operation did not complete is written to the Software Distribution log file. To specify that a distribution must be stopped when invalid targets are encountered, use the **continue_on_invalid_targets** key in the wswdcfg command.

Authorization

admin, senior, or super

Return Values

The **wversp** command returns one of the following:

0 Indicates that **wversp** started successfully.

-1 Indicates that **wversp** failed due to an error.

Examples

To verify the installation of the fsys^1.0 software package on the target1 and target2 endpoints, enter the following command:

```
wversp @fsys^1.0 @target1 @target2
```

See Also

None.

wwebgw

Lists information about distributions on a resource gateway. Allows a distribution on a resource gateway to be canceled.

Syntax

wwebgw [**-c** *distribution_ID*] **@Endpoint:***endpoint_name*

wwebgw [**-d** *distribution_ID*] **@Endpoint:***endpoint_name*

wwebgw [**-l** [*application_ID*]] **@Endpoint:***endpoint_name*

Description

The **wwebgw** command allows you to list the distribution IDs of outstanding jobs, list the outstanding devices for jobs with a specific distribution ID, or cancel jobs for a specific distribution ID on a resource gateway. Note that for Inventory distributions, you should use the **wcancelscan** command to cancel the distribution. Refer to the *User's Guide for Inventory* for details about this command.

Options

The following lists the subcommands of the **wwebgw** command:

–c *distribution_ID*
: Cancels all jobs that have not yet completed for the specified distribution ID on a resource gateway. Canceled jobs are deleted 6 minutes after you isssue the command.

–d *distribution_ID*
: Lists the devices that have not yet completed for jobs on a resource gateway with distribution ID distribution_id.

–l [*application_ID***]**
: Lists the distribution and application IDs for outstanding jobs on a resource gateway. If application_id is specified, only distributions for the specified application ID are listed; otherwise all distributions are listed.

@Endpoint:*endpoint_name*
: Specifies the endpoint name of the endpoint for the resource gateway.

Authorization

For the –d or –l options: **user, admin, senior, super**

For the –c option: **admin, senior, super**

Return Values

The **wwebgw** command returns one of the following:

0 Indicates that the **wwebgw** command was successful.

-1 Indicates that the **wwebgw** command failed due to an error.

Examples

To list all outstanding distributions for the resource gateway on endpoint prague, enter the following:

```
wwebgw -l @Endpoint:prague
```

To list all outstanding devices for jobs with distribution ID 1624210394.34 for the resource gateway on endpoint prague, enter the following:

```
wwebgw -d 1624210394.34 @Endpoint:prague
```

To cancel all outstanding jobs for distribution ID 1624210394.34 for the resource gateway on endpoint prague, enter the following:

```
wwebgw -c 1624210394.34 @Endpoint:prague
```

See Also

The **wcancelscan** command in the *User's Guide for Inventory*

The **winstsp** command for the valid options for distributions to resource groups.

Disconnected Target Commands

There are a number of commands that are executed locally on a disconnected system.

Most of these commands have the same functions as related server commands, for example, install, accepting, committing, and removing software packages. An important use of these commands is to test software packages on a preparation machine before you distribute them to a larger environment.

In addition, there are disconnected commands which help you to keep control of software installed on an endpoint by providing the facilities to produce a list of installed packages on the endpoint (**wdlssp**) and to bring independently installed packages under the control of Software Distribution (**wdsetsps**).

Reports concerning these operations are sent to the Tivoli server only after a connected operation is performed on the endpoint.

To use these commands on an endpoint, you must install the Software Distribution Java Endpoint Package Editor.

The Software Distribution Web Interface can be run from a disconnected target system. See the *User's Guide for Software Distribution* for more information about the Web Interface.

Note: The disconnected command line interface is not available for managing software packages on OS/400 machines.

Table 49 lists the disconnected target commands.

Table 49. Disconnected target commands

Disconnected Target Command	Purpose	See Page
wdacptsp	Accepts a software package.	282
wdcmmtsp	Commits a software package.	283
wdinstsp	Installs a software package.	284
wdlssp	Lists the installed software packages.	286
wdrmvsp	Removes an installed software package.	287
wdsetsps	Records the presence on endpoints of software packages that have been installed independently of Software Distribution.	289
wdswdvar	Manages variables used in software packages on the endpoint where the command is run.	291
wdubldsp	Converts a software package from the built format to the unbuilt format.	293
wdundosp	Undoes an installed software package.	294
wdversp	Verifies an installed software package.	296

wdacptsp

Accepts a software package. This command is run against an object for which the previous operation was performed with the undoable option. Running this command deletes any backup copies and changes the status of the package accordingly.

Syntax

wdacptsp *spname.spver*

Description

The **wdacptsp** command deletes any backup copies, so the previous operation can no longer be undone.

Options

spname.spver
: Name and version of the installed software package. See “Software Package Name and Version” on page 2.

Return Values

The **wdacptsp** command returns one of the following:

0 Indicates that **wdacptsp** started successfully.

-1 Indicates that **wdacptsp** failed due to a generic error.

Positive return value
: Indicates that **wdacptsp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To accept the fsys_test^1.0 software package, enter the following command:

```
wdacptsp fsys_test^1.0
```

See Also

The **undo** option for **wdinstp** and **wdrmvsp**

wdcmmtsp

Commits a software package.

Syntax

wdcmmtsp [-**R**{ *y* | *n*}][**–c** *y* | *n* | *o* | *r*] *spname.version*

Description

The **wdcmmtsp** command causes all updates performed in the preparation phase (in transactional mode) to take effect.

Options

–R {*y* | *n***}**

Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–c *y* | *n* | *o* | *r*

Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see “Commit Operation” on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–cr** option is not available on disconnected target systems running the Windows ME platform.

spname.version

Name and version of the installed software package. See “Software Package Name and Version” on page 2.

Return Values

The **wdcmmtsp** command returns one of the following:

0 Indicates that **wdcmmtsp** started successfully.

-1 Indicates that **wdcmmtsp** failed due to a generic error.

Positive return value

Indicates that **wdcmmtsp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To prepare a commit operation for the object fsys_test^1.0, enter the following command:

```
wdcmmtsp -c y fsys_test^1.0
```

The **y** value for the **–c** option specifies that the commit will be executed after the first user reboot.

See Also

The **transactional** option for **wdinstsp**, **wdrmvsp**, and **wdundosp**

wdinstsp

Installs a software package.

Syntax

wdinstsp [**–p**] [[**–f**] [-**R**{ *y* | *n*}] [**–D** *variable=value*]... [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–u** *y* | *n* | *o* | *u* [**–a**]] *spblock_path*

wdinstsp [**–p**] [[**–f**] [-**R**{ *y* | *n*}] [**–D** *variable=value*]... [**–t** *y* | *n* | *o*] [**–u** *y* | *n* | *o* | *u* [**–a**]] *spblock_path*

wdinstsp [**-n** *spname.ver*] [[**–f**] [-**R**{ *y* | *n*}] [**–D** *variable=value*]... [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–u** *y* | *n* | *o* | *u* [**–a**]] *spblock_path*

wdinstsp [**-n** *spname.ver*] [[**–f**] [-**R**{ *y* | *n*}] [**–D** *variable=value*]... [**–t** *y* | *n* | *o*] [**–u** *y* | *n* | *o* | *u* [**–a**]] *spblock_path*

Description

This command performs an install on the target system of the actions described in a software package.

Options

–p Specifies a preview installation only; the software package is not actually installed.

–n *spname.ver*
Specifies the name and version of the software package to be installed.

–f Specifies forcing the operation, regardless of the state of the disconnected target system.

–R *y* | *n*
Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–D *variable=value*
Defines the value of a variable used in the software package, to add or override existing variables. When specifying multiple variables, repeat the **–D** attribute before each *variable=value*. Note that these variables can be resolved only on the endpoint.

–t *y* | *n* | *o*
Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y* | *n* | *o* | *r*
Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see “Commit Operation” on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–cr** option is not available on disconnected target systems running the Windows ME platform.

The -c attribute can only be used in conjunction with the **-ty** or **-to** (transactional) options.

–u *y | n | o | u*
Specifies the undoable option: **n** (no, which is the default), **y** (yes), **o** (preferably), or **u** (undoable-in-transactional).

–a Specifies that the operation should be automatically accepted. This attribute is used in conjunction with the **–u** (undo) attribute.

spblock_path
Specifies the path to the software package block.

Return Values

The **wdinstsp** command returns one of the following:

0 Indicates that **wdinstsp** started successfully.

-1 Indicates that **wdinstsp** failed due to a generic error.

Positive return value
Indicates that **wdinstsp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To install the software package block d:\testdir\fsystem.spb with the undoable option, enter the following command:

```
wdinstsp -u y d:\testdir\fsystem.spb
```

See Also

wdrmvsp, **wdundosp**, **wdcmmtsp**, **wdacptsp**

wdlssp

Lists the software packages installed on an endpoint, including hidden packages.

Syntax

wdlssp

wdlssp -b

Description

The output of the **wdlssp** command includes a list of software package names, version and software package states.

See Chapter 2, “Performing Change Management Operations,” on page 143 for more information about software package states and the use of change management operations.

Options

-b Creates a backup copy of the catalog to the file you specified. The information stored in the epsp.cat file is retrieved up to the point where the corruption occurred. Some data in the new file might be inconsistent if the command failed to retrieve complete data from the corrupt catalog. You can then manually replace the catalog with the new file.

Return Values

The **wdlssp** command returns one of the following:

0 Indicates that **wdlssp** started successfully.

-1 Indicates that **wdlssp** failed due to a generic error.

Positive return value
Indicates that **wdlssp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

Following is an example of the output from the **wdlssp** command:

```
----------------------------
Name             : MY_APPL
Version          : 1.0
State            : ICU--
----------------------------
Name             : ADD_FILES
Version          : 1.0
State            : IC---
-----------------------------
```

See Also

wdinstsp, **wdrmvsp**, **wdcmmtsp**, **wdundosp**, **wdacptsp**

wdrmvsp

Removes an installed software package.

Syntax

wdrmvsp [**–f**] [-**R**{ *y* | *n*}][**–D** *variable=value...*] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] [**–u** *y* | *n* | *o* | *u* [**–a**]] {{**–s** | **–S**} *spname_path* | *spname.version*}

wdrmvsp [**–f**] [-**R**{ *y* | *n*}][**–D** *variable=value...*] [**–t** *y* | *n* | *o*] [**–u** *y* | *n* | *o* | *u* [–**a**]] {{**–s** | **–S**} *spname_path* | *spname.version*}

Description

Options

–f Specifies forcing the operation, regardless of the state of the target system.

–R *y* | *n*
Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–D *variable=value*
Defines the value of a variable used in the software package, to add or override existing variables. If you are removing a software package that has already been installed, you can define or override only those variables that were not solved during the previous install operation. When specifying multiple variables, repeat the **–D** argument before each *variable=value*. Note that these variables can be resolved only on the endpoint.

–t *y* | *n* | *o*
Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y* | *n* | *o* | *r*
Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see “Commit Operation” on page 148.

Note: The only option available on the UNIX platform is **–cn**.

The **–cr** option is not available on disconnected target systems running the Windows ME operating system.

The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options.

–u *y* | *n* | *o* | *u*
Specifies the undoable option: **n** (no, which is the default), **y** (yes), **o** (preferably), or **u** (undoable-in-transactional).

–a Specifies that the operation should be automatically accepted. This attribute is used in conjunction with the **–u** (undo) attribute.

–s Specifies to use the software package to perform the operation. This attribute can be used if the software package object does not exist or in conjunction with the **–f** attribute, which forces the operation.

–S Specifies to use the software package block to perform the operation. This attribute can be used if the software package object does not exist or in conjunction with the **–f** attribute, which forces the operation.

spname_path
Name of the software package path file (used in conjunction with the **–s** and **–S** attributes).

spname.version
Name and version of the installed software package. See “Software Package Name and Version” on page 2.

Return Values

The **wdrmvsp** command returns one of the following:

0 Indicates that **wdrmvsp** started successfully.

-1 Indicates that **wdrmvsp** failed due to a generic error.

Positive return value
Indicates that **wdrmvsp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To remove the software package fsys_test^1.0, enter the following command:

```
wdrmvsp fsys_test^1.0
```

See Also

wdinstsp

wdsetsps

Records the presence on a target system of an application that was installed independently of Software Distribution.

Syntax

wdsetsps [**–f**] [**–v** *versioning_type*] [**–t** *package_type*] **-T** *pkgfile...*

wdsetsps [**–f**] [**–v** *versioning_type*] [**–t** *package_type*] *spname.version*

Description

Using this command, you can bring applications, which have been independently installed, under the control of Software Distribution. When you execute this command on a disconnected target system, the specified software package is assigned a state of IC-D-, indicating that it is installed and ″discovered″.

The command includes optional version checking. If you use version checking, the software package states of previous versions of the package that are present on the endpoint are set to removed in the catalog. A message is displayed to alert you about the deletion. You can use the **–f** argument to suppress this warning.

After using this command, you can update the information on the server to include the statuses of discovered software packages by running the command **wsyncsp**.

There are limitations to the change management operations that can be used for a discovered software package. Only the following operations are available:

- Remove software package (not in Transactional mode)
- Force install software package

Options

–f
: If version checking is used, specifies that no warning message is to be issued before deleting previous versions of the software package.

–v *versioning_type*
: Specifies whether to use version checking when adding the package to the catalog. Possible values are SWD (use version checking) and None (no version checking). The default is SWD.

–t *package_type*
: Specifies the type of package and controls the versions checks to be made. Possible values are REFRESH and PATCH. The default is REFRESH. See “Software Package Version Checking” on page 4 for details of how version checks are made.

-T *pkgfile*
: Specifies a file containing names of software packages to be registered with Software Distribution.

spname.version
: Name and version of the installed software package. See “Software Package Name and Version” on page 2.

  You can specify multiple packages.

Return Values

The **wdsetsps** command returns one of the following:

0
: Indicates that **wdsetsps** started successfully.

-1 Indicates that **wdsetsps** failed due to a generic error.

Positive return value
Indicates that **wdsetsps** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To discover the refresh software package fsys_test.2.0, with version checking, enter the following command:

```
wdsetsps -v SWD -t REFRESH fsys_test.2.0
```

See Also

wsyncsp, wsetsps

wdswdvar

Manages variables used in software packages on the endpoint where the command is run.

Syntax

wdswdvar -a

wdswdvar -s *var=value*

wdswdvar -g *var*

wdswdvar -d *var*

Description

The **wdswdvar** command creates, modifies, displays, and deletes variables defined in the swdis.var file and used in software packages on the endpoint where the command is run. The variables are resolved at runtime on the endpoint machine and the resulting value is used by software packages to perform the required operations. This command helps the user edit the swdis.var file, and is mainly used in before and after programs.

The swdis.var file is created when you first use this command, if it is not already existing, and is saved to the directory specified in the swdis.ini file.

This command is downloaded to the endpoint when a software package is distributed. It is not available in the disconnected command line and is used only in before and after programs. For more information, see "Setting Up Before and After Programs on the Endpoint" on page 15.

Notes:

1. When using this command in before and after programs on Netware endpoints, insert the following statement as the first line in the program:

   ```
   search add path_to_wdswdvar.nlm_file
   ```

2. On Netware endpoints, you cannot use this command within before and after programs.

Note:

Options

–a Returns a list of all defined variables with their associated values.

–s *var=value*
Sets the value for the specified variable.

–g *var*
Returns the current value for the specified variable.

-d *var* Deletes the specified variable.

Return Values

The **wdswdvar** command returns one of the following:

0 Indicates that **wdswdvar** started successfully.

-1 Indicates that **wdswdvar** failed due to a generic error.

Examples

1. To define the variable WebSrvHomeDir so that it specifies the path to the HTTP Server, enter the following command:

   ```
   wdswdvar -s WebSrvHomeDir=/IBMHTTPServer
   ```

2. To set the home directory to /test/temp, enter the following command:

   ```
   wdswdvar -s home_dir=/test/temp
   ```

See Also

wsetspgs, **wgetspgs**

wdubldsp

Converts a software package from the built format (spb) to the not built format (sp or spd) and saves the contents of the software package to a specified directory.

Syntax

wdubldsp {**–s** *sp_path* | **–f** *spd_path*} [**–o**] *spb_path target_dir*

Description

The **wdubldsp** command converts a built software package to one of the not built formats, .sp or .spd. It saves the resultant software package to the directory you specify in the *sp_path* or *spd_path* variable and saves the contents of the package to the directory specified in the *target_dir* variable.

The command updates the stanza in the software package that indicates the location of the software package contents, so that it points to the specified directory. Relative source paths are not supported. If these paths are present, the unbuild operation might fail or behave unpredictably.

Options

–s *sp_path*
: Specifies the directory where the unbuilt software package is to be stored, if you are converting it to sp format.

–f *spd_path*
: Specifies the directory where the unbuilt software package is to be stored, if you are converting it to spdf format.

–o
: Indicates that an existing file with the same name must be overwritten.

spb_path
: Specifies the fully qualified file name of the built software package that you want to convert.

target_dir
: Specifies the directory where the contents of the software package are to be stored.

Return Values

The **wdubldsp** command returns one of the following:

0
: Indicates that **wdubldsp** started successfully.

-1
: Indicates that **wdubldsp** failed due to a generic error.

Positive return value
: Indicates that **wdubldsp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To unbuild software package block file package1.spb to software package definition format and to save the contents to the directory c:\staging, enter the following command:

```
wdubldsp -f c:\spd c:\spb\package1.spb c:\targdir
```

See Also

wconvspo, wdbldsp, wdexptsp, wdcrtsp.

wdundosp

Undoes the previous operation.

Syntax

wdundosp [-**R**{ *y* | *n*}] [**–t** *y* | *o* [**–c** *y* | *n* | *o* | *r*]] *spname.version*

wdundosp [-**R**{ *y* | *n*}] [**–t** *y* | *n* | *o*] *spname.version*

Description

The **wdundosp** command returns the system to its state prior to the execution of the previous operation. This command is used for objects for which the previous operation was run in undoable mode.

Options

–R *y* | *n*
: Specifies whether or not dependency checking should be used. This option is available only if the target is already registered in the Inventory database as a consequence of a scan or a change management operation performed on the target. The default is **–R y**; **–R n** indicates that any dependency expression defined in the SPD file should be ignored.

–t *y* | *n* | *o*
: Specifies the transactional option: **n** (no, which is the default), **y** (yes), or **o** (only if necessary).

–c *y* | *n* | *o* | *r*
: Specifies the reboot options for the commit operations: **n** (not-in-a-reboot), which is the default, **y** (in-a-reboot), **o** (in-a-reboot), **r** (auto-reboot). For more information on these options, see “Commit Operation” on page 148. The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options. The **–c** attribute can only be used in conjunction with the **–ty** or **–to** (transactional) options.

  Note:

  The only option available on the UNIX platform is **–cn**

  The **–cr** option is not available on disconnected target systems running the Windows ME platform.

spname.version
: Name and version of the installed software package. See “Software Package Name and Version” on page 2.

Return Values

The **wdundosp** command returns one of the following:

0 Indicates that **wdundosp** started successfully.

-1 Indicates that **wdundosp** failed due to a generic error.

Positive return value
: Indicates that **wdundosp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To undo the last operation performed on fsys_test^1.0, enter the following command:

```
wdundosp fsys_test^1.0
```

See Also

The **undo** option for **wdinstsp** and **wdrmvsp**

wdversp

Verifies an installed software package.

Syntax

wdversp *spname.version*

Description

The **wdversp** command verifies that the operation executed on the target object is consistent with the installed package, that is, that the files have been successfully installed on the target system.

Note: If a package contains two commands that perform opposite operations, such as adding and removing the same object, one of the operations will fail verification.

Options

spname.version
Name and version of the installed software package. See "Software Package Name and Version" on page 2.

Return Values

The **wdversp** command returns one of the following:

0 Indicates that **wdversp** started successfully.

-1 Indicates that **wdversp** failed due to a generic error.

Positive return value
Indicates that **wdversp** failed due to a specific error. For more information about return values, see Table 50 on page 297.

Examples

To verify the installation of the fsys^1.0 software package on the target system, enter the following command:

```
wdversp @fsys^1.0
```

See Also

wdinstsp, **wdacptsp**, **wdcmmtsp**, **wdundosp**

Return Values

Return values help you identify the result of the command: a return value of zero indicates that the command completed successfully, while a return value of -1, or other than zero, indicates that an error has occurred. A list of all return values other than -1 and zero is given in Table 50.

Table 50. Return Values

Exit Code	Value	Operation Flow
success_reboot_now	1	Operation successful. The machine is rebooting automatically.
success_reboot_now_reexecute	2	Operation successful. Manual reboot is required.
success_reboot_after	3	Operation successful. Manual reboot is required.
success_reboot_after_reexecute	4	Operation successful. Manual reboot is required.
success_in_a_reboot	5	Operation successful. Manual reboot is required.
success_retry	6	Only used for internal checks.
warning	7	Operation successful. Some operations returned a warning.
temporary_failure	8	A temporary error occurred. Before re-entering the command, you must correct the error.
failure	9	Operation unsuccessful. The system can not be rolled back to its previous state.
fatal_failure	10	Operation unsuccessful. The system can not be rolled back to its previous state.
failure_with_info	11	Operation unsuccessful. Additional information is displayed to standard error.

Preparation Site Commands

The following table lists the commands that can be used to prepare a software package:

Table 51. Preparation commands

Preparation Command	Purpose	See Page
autopack	Creates a software package by comparing successive snapshots of the disk contents of a target system.	299
wdbldspb	Creates a software package block from a software package.	301
wdcrtsp	Creates a software package or software package block from a software package definition (SPD) file.	302
wdexptsp	Exports a software package.	303

autopack

Creates a software package by comparing successive snapshots of the disk contents of a target system.

Syntax

autopack{**–f** [**–l** *drive_to_scan*] | **–s | –d** [**–n** *sp_name*] [**–v** *sp_version*] [**–p** *sp_file*] [**–b**]}

Description

See *User's Guide for Software Distribution* for a complete description of this tool.

Options

–f Specifies to run the first snapshot.

–l *drive_to_scan*
Specifies the drive to be scanned. This attribute can be used only in conjunction with the **–f** option. On UNIX systems, the order in which directories are scanned is as follows:

1. The directory specified with the **-l** option, if provided.
2. The directory specified in the autopack.ini file, if existing.
3. The directory specified in the $HOME environment variable, that is the directory of the user issuing the command.

–s Specifies to run the second snapshot.

–d Specifies to compare the first and second snapshots and record the differences.

–n *sp_name*
Specifies the name of the software package. The default name is autopack.

–v *sp_version*
Specifies the version of the software package. The default version is 1.0.

–p *sp_file*
Specifies the name of the software package (or software package block, if the **–b** option is specified) to be created that contains the differences between the first and second snapshots. The default name is autopack.sp.

–b Used in conjunction with the **–d** attribute, specifies that the differences file that is generated is in software package block format (built), not software package format (not-built).

Return Values

The **autopack** command returns one of the following:

0 Indicates that **autopack** started successfully.

-1 Indicates that **autopack** failed due to an error.

Examples

1. To create the first snapshot for the d: drive, enter the following command:

   ```
   autopack -f -l d:
   ```

2. To create the second snapshot, enter the following command:

   ```
   autopack -s
   ```

3. To create a software package named diff.sp that contains the differences between the two snapshots, enter the following command:

```
autopack -d -p diff.sp
```

4. To create a software package block called diff.spb that contains the differences between the two snapshots, run the **wdbldspb** command on the software package or enter the following command:

   ```
   autopack -d -b -p diff.spb
   ```

See Also

wdbldspb

wdbldspb

Builds a software package block from a software package.

Note: The size of the software package block cannot exceed two gigabytes.

Syntax

wdbldspb [**–o**] *sp_path spblock_path*

Description

Options

–o Specifies to overwrite an existing target file. If not specified and *spblock_path* already exists, an error results.

sp_path
Path of the input software package.

spblock_path
Path of the output software package block.

Return Values

The **wdbldspb** command returns one of the following:

0 Indicates that **wdbldspb** started successfully.

-1 Indicates that **wdbldspb** failed due to an error.

Examples

Starting with a software package called d:\testdir\fsystem.sp, enter the following command to build a software package block called d:\testdir\fsystem.spb:

```
wdbldspb -o d:\testdir\fsystem.sp d:\testdir\fsystem.spb
```

The **–o** option specifies that if the output file already exists, it is overwritten.

See Also

wdcrtsp

wdcrtsp

Using a software package definition (SPD) file as input, creates a software package (not-built format) or software package block (built format).

Note: If creating in the built format, the size of the software package block cannot exceed two gigabytes.

Syntax

wdcrtsp [**–s**] [**–o**] [**–f** *spfile_path*] *target_path*

Description

Options

–s Specifies to output a software package rather than a software package block.

–o Specifies to overwrite an existing target file. If not specified and *spblock_path* already exists, an error results.

–f *spfile_path*
Inputs an SPD file from the specified path.

target_path
Path of the output software package or software package block.

Return Values

The **wdcrtsp** command returns one of the following:

0 Indicates that **wdcrtsp** started successfully.

-1 Indicates that **wdcrtsp** failed due to an error.

Examples

Starting with a software package definition file called d:\testdir\fsystem.spf, enter the following command to create a software package block called d:\testdir\fsystem.spb:

```
wdcrtsp -f d:\testdir\fsystem.spf d:\testdir\fsystem.spb
```

See Also

wdbldspb, **wdexptsp**

wdexptsp

Exports a software package in software package definition (SPD) file format.

Syntax

wdexptsp [[**–o**] [**–b**] **–f** *export_file*] {{**–s** | **–S**} *spname_path* | *spname.version*}

Description

Options

–o Specifies to overwrite an existing target file. If not specified and *export_file* already exists, an error results.

–b Specifies to export the file in software package format. This attribute is used only in conjunction with the **–f** and **–S** attributes.

–f *export_file*
Specifies to export the software package to a file. If **–f** is not specified, the software package is exported to standard output.

–s Specifies that a software package is to be exported.

–S Specifies that a software package block is to be exported.

spname_path
Name of the software package path file.

spname.version
Name and version of the installed software package. See “Software Package Name and Version” on page 2.

Return Values

The **wdexptsp** command returns one of the following:

0 Indicates that **wdexptsp** started successfully.

-1 Indicates that **wdexptsp** failed due to an error.

Examples

To export the contents of software package block named d:\testdir\fsystem.spb to d:\testdir\fsystem.exp, enter the following command:

```
wdexptsp -f d:\testdir\fsystem.exp -S d:\testdir\fsystem.spb
```

See Also

wdbldspb, **wdcrtsp**

Chapter 4. Managing Policy

Tivoli policy enables you to control the default values of newly created resources (default policy) and to maintain guidelines when administrators modify or operate on resources (validation policy). Specifically, the Software Distribution default and validation policies enable you to set defaults and enforce guidelines for software package properties and operations. These policies are implemented as shell scripts or programs: UNIX scripts (such as Bourne, K, and Perl shells), awk programs, C programs, and so on.

Default policy sets the default values for software package properties. These policies are useful if you want to preset software package properties with specific values. For example, if most of your software packages will have the same source host, you could define a default policy so that every newly created software package has its source host set to that machine. You can change properties set to a value by a default policy (if you do not violate validation policy).

Similarly, *validation policy* ensures that software package properties or operations always adhere to *rules*. For example, you can create a script for a validation policy method specifying that software package names cannot contain punctuation marks or slashes. For example, if an administrator attempts to name a software package data\upgrades, the validation fails and the administrator must select another name, such as dataupgrades.

Software Distribution policy is policy-region-based. When you set a default or validation policy, that policy method generally runs on the Tivoli Management Region server in the policy region in which the software package resides. The policy applies to all software package resources in that policy region. The names of the methods and their inputs remain the same.

Note: A policy method that does not run in the policy region where the resource resides is sp_val_src_host. This method runs in the policy region where the source host resides for the software package.

Because these resources can only reside on UNIX managed nodes, the policy methods must be UNIX scripts, programs, or executables. Policies are stored in the database, so you should consider writing your policy methods in an interpreter language to save space. Executables (compiled programs) are generally larger. While an interpretive program can often be used across multiple platforms; executables often cannot. Thus, use UNIX shell scripts or programs on UNIX managed nodes. Do not use C shell scripts.

For more information on default and validation policy and policy regions, see the *Tivoli Management Framework: User's Guide*.

Default Policy Methods

Default policy methods are shell scripts or programs invoked by Software Distribution when you create a new software package. By creating scripts or programs and replacing the contents of these policy methods, you can automatically set the properties in newly created software packages.

Default Policy Methods

When Software Distribution invokes a default policy method, the name of the software package being created is passed to the method. Software Distribution expects the default policy methods to exit with the code 0, so you must write your policy methods to do so. Reserve other exit codes for hard errors, such as insufficient memory, incorrect usage, and so on.

The default policy methods available with Software Distribution are listed in the following table:

Table 52. Default policy methods

Method	Purpose
sp_def_src_host	Generates the default source host for the software package. No keywords are associated with this method. See "Examples of Default Policy Methods" on page 309 for an example of the usage of sp_def_src_host.
sp_def_properties	Generates the default properties for the software package. See Table 53 for a list of the keywords associated with this method.

Following is a list of keywords along with their valid values that are managed by sp_def_properties. For more information on these keywords, see Table 53.

Table 53. Keywords for sp_def_properties method

Keyword	Possible Values	Default Value
after_prog_env	String of *name=value* pairs	None
before_prog_env	String of *name=value* pairs	None
no_check_source_host	**y**: yes **n**: no	**y**
committable	**y**: yes **n**: no **o**: optional	**o**
lenient_distribution	**y**: yes **n**: no	**n**
log_gid	*ID value*	**-1**
log_host	*host name*	Host name of the Tivoli Management Region server. If you set this keyword to the name of a system that is not a valid managed node, the log_host keyword is set to null.
log_mode	Octal 0-777	0
log_path	*path*	Working directory of the distribution server
log_uid	*ID value*	None (**0**)
mail_id	*mail address*	None
mv_on_rm_host	**y**: yes **n**: no	**y**
no_chk_on_rm	**y**: yes **n**: no	**y**

Table 53. Keywords for sp_def_properties method (continued)

Keyword	Possible Values	Default Value
post_notice	**y**: yes **n**: no	**n**
src_after_as_uid	*ID value*	None (**0**)
src_after_input_path	*path*	None
src_after_prog_path	*path*	None
src_before_as_uid	*ID value*	None (**0**)
src_before_input_path	*path*	None
src_before_prog_path	*path*	None
src_before_skip_non_zero	**y**: yes **n**: no	**n**
stage_area	*path*	Working directory of the distribution server
undoable	**y**: yes **n**: no **o**: optional	**o**
web_view_mode	**hidden** **subscriber** **public**	**hidden**

Default Policy Methods for Software Packages

The following figures map the software package default policy methods to the Software Package Properties and Advanced Properties dialog boxes. When you set or edit a policy, you will see the default value in the indicated section of the dialog box.

Figure 21 is the Software Package Properties dialog box.

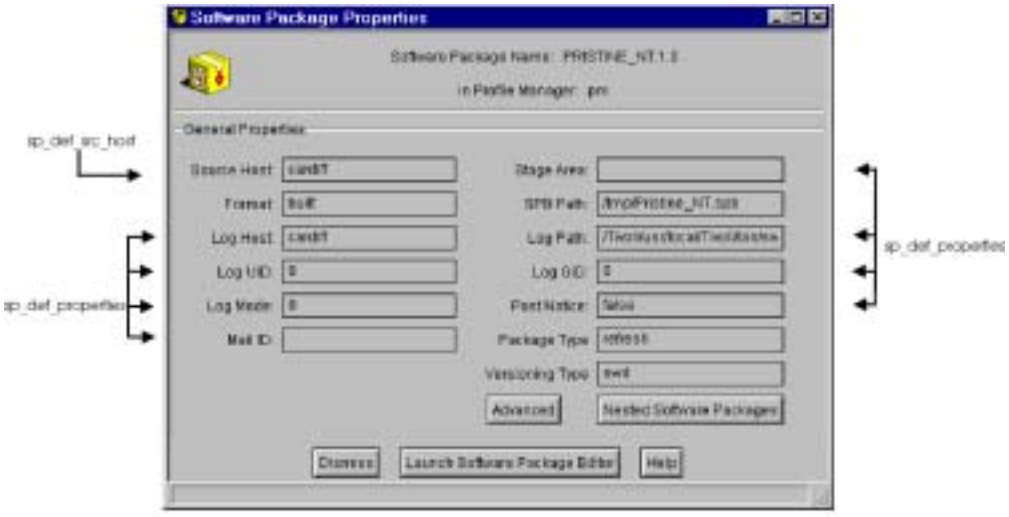


Figure 21. Software package properties (general) and default policy methods

Figure 22 is the Advanced Properties dialog box.

Figure 22. Advanced properties and default policy methods

The following example maps the default policy methods to a software package definition (SPD) file. If you choose to set software package properties using export/import, you will see the values set by a default policy method in the exported software package definition file, as shown below.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                  = "test"
  version               = "2.0"
  web_view_mode         = "hidden"
```

```
  undoable               = "o"
  committable            = "o"
  history_reset          = n
  save_default_variables = n
  creation_time          = "2000-05-23 14:54:44"
  last_modification_time = "2000-05-23 14:55:02"
  source_host_name       = "msecchi"
  stage_area             = "D:\Tivoli\bin\courier\work"
  move_removing_host     = y
  no_check_source_host   = y
  lenient_distribution   = n
  default_operation      = "install"
  server_mode            = "all"
  operation_mode         = "not_transactional"
  log_path               = "D:\Tivoli\bin\courier\ /
    work\test^2.0.log"
  log_mode               = "0"
  log_user_id            = 0
  post_notice            = n
  before_as_uid          = 0
  skip_non_zero          = n
  after_as_uid           = 0
  no_chk_on_rm           = y
  log_gid                = -1
  log_host_name          = "msecchi"
  stop_on_failure        = y
end
```

The following attributes in this example SPD file are set by means of the sp_def_properties method:

- undoable
- committable
- stage_area
- move_removing_host
- lenient_distribution
- log_path
- log_user_id
- post_notice
- before_as_uid
- skip_non_zero
- after_as_uid
- no_chk_on_rm
- log_gid
- log_host_name

In addition, the source_host_name attribute is set by means of the sp_def_src_host method. In the case of attributes with no default value, such as src_after_input_path, such attributes are not exported unless a value has been specified. For more information on the SPD file attributes, see Chapter 1, “Editing the Software Package Definition File,” on page 1.

Examples of Default Policy Methods

The following default policy method examples are UNIX Bourne shell scripts that set various software package properties. Those scripts that simply echo a value (for example, sp_def_src_host) are only responsible for setting one software package option.

To set the source host to the name ″jazz″ for all newly created software packages, create the following script for the sp_def_src_host policy method:

```
#!/bin/sh
PATH=/bin:/usr/bin
echo jazz
exit 0
```

Using the sp_def_properties policy method to set the undoable option to **y**, create the following script:

```
#!/bin/sh
PATH=/bin:/usr/bin
echo undoable=y
exit 0
```

To set multiple options at the same time, create the following script for the sp_def_properties policy method:

```
#!/bin/sh
PATH=/bin:/usr/bin
sp_name="$1"
cat <<EOF
undoable=y
committable=y
post_notice=n
lenient_distribution=n
stage_area=/staging/$sp_name
EOF
exit 0
```

Validation Policy Methods

Validation policy methods are called when software package properties are set or modified. Validation policy also ensures that an attempted software package operation is allowed. When you modify or perform an operation on a software package using export/import, the command line, or the Install Software Package dialog box, the software package validation policy methods are invoked.

Validation policy methods are shell scripts or programs that Software Distribution automatically calls when you perform any of these actions. Initially, the validation policy methods are not set. Thus, when you create, modify, or perform an operation on a software package, none of the properties are checked. By creating scripts or programs and replacing the contents of these validation policy methods, you can control changes to software package properties or the attempted software package operations.

Validation policy methods receive as input the proposed value of the software package property. They return TRUE if the input passes validation or FALSE if the input does not pass validation. For example, Software Distribution always invokes the sp_val_operation policy method to verify proposed software package operations. If an operation does not adhere to the guidelines set by this policy method, the validation fails and the method returns FALSE.

Software Distribution expects the validation policy methods to exit with the code 0 if successful, even if the input does not pass validation. Reserve other error codes for hard errors, such as insufficient memory, incorrect usage, and so on.

In general, policy methods run in the policy region of the software package. However, the sp_val_src_host and sp_val_delete_src_host policy methods run in the policy region of the source host, which may differ from the policy region of the software package.

The validation policy methods in the following table are available with Software Distribution. Because these methods are also invoked when you set or change software package properties using the command line, this table also lists the commands that invoke each validation policy method:

Table 54. Validation policy methods

Method	Purpose	Command
sp_val_delete_src_host	Validates the removal of the source host for a software package. When the source host of a software package is changed from one host to another, this method first validates the *unsetting* of the original source host. See “sp_val_delete_src_host” on page 323 for more information on the syntax of this method.	**wimpspo** **wsetspat –h**
sp_val_name	Validates the proposed name of the software package. See “sp_val_name” on page 325 for more information on the syntax of this method.	**wimpspo**
sp_val_operation	Validates the operations performed on software packages. See “sp_val_operation” on page 327 for more information on the syntax of this method.	**wconvspo** **wimpspo** **winstsp** **wremovsp** **wexptsp** **wundosp** **waccptsp** **wcommtsp** **wversp**
sp_val_properties	Validates software package properties. See “sp_val_properties” on page 329 for more information on the syntax of this method.	**wimpspo** **wsetspat** **wsetspgs** **wsetspop**
sp_val_src_host	Validates the proposed source host of the software package. See “sp_val_src_host” on page 331 for more information on the syntax of this method.	**wimpspo** **wsetspat –h**

The following figures map the software package validation policy methods to the Software Package Properties and Advanced Properties dialog boxes. When you set or edit a software package property, the validation policy method that corresponds to that property is invoked.

Figure 23 on page 312 is the Software Package Properties dialog box.

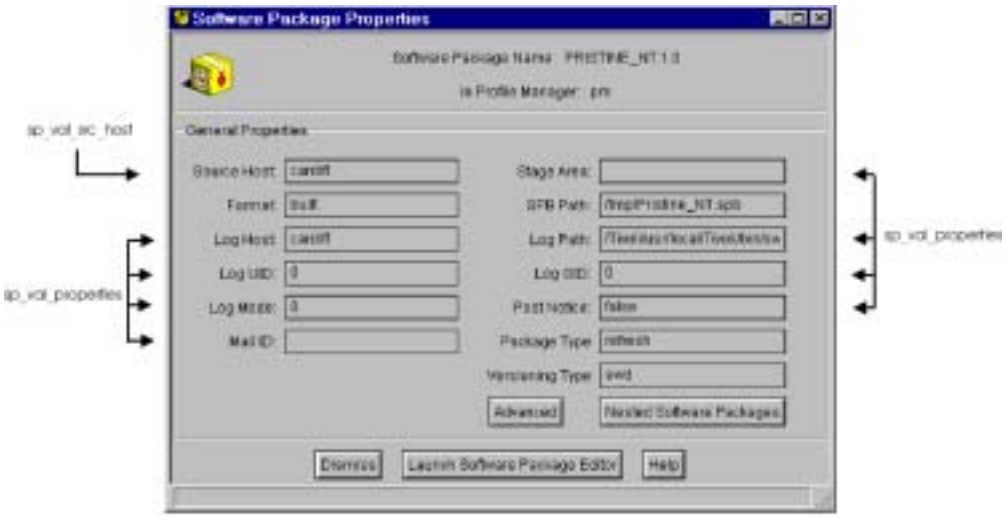


Figure 23. Software package properties (general) and validation policy methods

Figure 24 is the Advanced Properties dialog box.

Figure 24. Advanced properties and validation policy methods

The following example maps the validation policy methods to the software package definition file. The same set of properties are validated through the sp_val_properties method as were previously defined through the sp_def_properties method. The sp_val_delete_src_host and sp_val_src_host methods are validated in relation to the source_host_name keyword. The sp_val_name method refers to the software package name that will be stored in the Tivoli object database and is composed of *name^version* (unless you are importing to a software package object that already exists). The sp_val_operation method is linked to the operation being performed by the software package. If you choose to set software package properties using export/import, the validation policy method that corresponds to a changed property is invoked. The sp_val_name, the sp_val_operation, the sp_val_delete_src_host, and the sp_val_src_host policy methods are not invoked by changes to the SPD file.

```
"TIVOLI Software Package v4.3.1 - SPDF"
package
  name                    = "test"
  version                 = "2.0"
  web_view_mode           = "hidden"
  undoable                = "o"
  committable             = "o"
  history_reset           = n
  save_default_variables  = n
  creation_time           = "2000-05-23 14:54:44"
  last_modification_time  = "2000-05-23 14:55:02"
  source_host_name        = "msecchi"
  stage_area              = "D:\Tivoli\bin\courier\work"
  move_removing_host      = y
  no_check_source_host    = y
  lenient_distribution    = n
  default_operation       = "install"
  server_mode             = "all"
  operation_mode          = "not_transactional"
  log_path                = "D:\Tivoli\bin\courier\ /
    work\test^2.0.log"
  log_mode                = "0"
  log_user_id             = 0
  post_notice             = n
  before_as_uid           = 0
  skip_non_zero           = n
  after_as_uid            = 0
  no_chk_on_rm            = y
  log_gid                 = -1
  log_host_name           = "msecchi"
  stop_on_failure         = y
end
```

The following examples provide shell scripts and programs for various validation policy methods.

To ensure that an attempted operation (except for a commit operation) is valid anytime except for Monday through Friday, from 8 a.m. to 5 p.m., create the following script to set the sp_val_operation policy method. This script is invoked at the start of a Software Distribution change management operation.

The script indicates validation failure if the current day and time is within this 8 a.m. to 5 p.m. time period.

When this program is called, the name of the software package is passed as the first argument to the program. The type of operation is passed as the second argument.

```
#!/bin/sh
export PATH

FP_NAME=$1
FP_OP=$2
DAY=`date +%w`
HOUR=`date +%H`

# Allow a commit operation
# The operation must be specified in lowercase
if [ $FP_OP = "commit" ]; then
   echo TRUE
   exit 0
fi

#If it's Sunday or Saturday, return SUCCESS
if [ $DAY = 6 ] || [ $DAY = 0 ] ; then
```

```
   echo TRUE
   exit 0
fi

# If the day is Monday-Friday, ensure that the hour is not
#   between 0800 (8am) and 2000 (8pm)
if [ $HOUR -ge 08 ] && [ $HOUR -lt 20 ]; then
   echo FALSE
   exit 0
fi
echo TRUE
exit 0
```

To validate the same options on the UNIX operating system, create the following program:

```
#!/usr/bin/sh
  while read IN; do
  if [ ! "$IN" = "" ];then
   if [ "$IN" = "committable=y" -o "$IN" = "undoable=y" ]; then
      echo FALSE; exit 0
   fi
  fi
  done
echo TRUE
exit 0
```

In addition, the following Perl script validates the lenient_distribution, rmv_on_rm_host, log_path, and log_host software package options:

```
#!/etc/Tivoli/bin/perl

# SP_VAL_PROPERTIES: Validate Software Package Options
# usage: sp_val_properties sp_name < options

# The following associative array defines the keywords
# of interest and their required values:
%check = (
  'lenient_distribution', 'y',
  'rmv_on_rm_host', 'y',
  'log_path',  'fuji:/home/dist/dist_log',
  'log_host_name', 'host_name'
);

# Check usage
if ( @ARGV ne 1 ) {
  die("usage: sp_val_properties sp_name < options\n");
} else {
  ($fp_name) = @ARGV;  # (unused)
}

$status = 'TRUE';

# Iterate over each line on standard input.
# Split each line into keyword and value around the "=".
# If the keyword is present in the check array (as an index),
# make sure the value matches the required one as defined
# in the array.  Otherwise, set the status and exit the
# while loop.
while ( <STDIN> ) {
  chop;   # discard trailing new line
  ($keyword, $value) = split('=', $_, 2);
  if (defined $check{$keyword} && $value ne
  $check{$keyword} ) {
   $status = 'FALSE';
   last;
```

```
}
}
print "$status\n";
exit 0;
```

Policy Objects

You specify the resource type of a new policy validation object with the SoftwarePackage attribute. The default and validation policy methods that govern SoftwarePackage resources are defined in a policy default object and a policy validation object. Both objects for the SoftwarePackage resource are called BasicSoftwarePackage. A policy object is a set of policy methods for a specific resource class. Each resource type has two policy objects that define its default and validation policy methods.

Each BasicSoftwarePackage policy object and the contained policy methods are provided with Software Distribution. You can, however, create additional policy objects for the SoftwarePackage resource. Multiple policy objects enable you to define different policies that are enforced in different policy regions.

For example, suppose you have two policy regions called Data and Software. To create policies that govern the software package operations and properties in each policy region, you can create separate policy objects, such as DataSPpolicy and SoftwareSPpolicy. After you define the policies in each policy object and link the policy objects to the policy regions, any newly created software packages would adhere to the guidelines you defined.

To define a new policy object and its policy methods, you must perform the following procedures:

1. Create a new policy object.
2. Replace the contents of the new policy object methods.
3. Assign the new policy object to the policy region in which the software packages will reside or to the policy region where the software package source host resides.

The following sections provide detailed instructions on each of these procedures for the SoftwarePackage resource. See the *Tivoli Management Framework: User's Guide* for detailed information about checking policy in a policy region.

Creating a New Policy Object

To define different policies for multiple policy regions, you must create new policy objects. If you do not define policy for a policy region, Software Distribution uses the BasicSoftwarePackage policy object and its policy methods by default.

*Table 55 shows the context and role required for this task.

Table 55. Roles for creating policy objects

Activity	Context	Required Role
Create a new policy object	Tivoli Management Regions	senior or super

You must use the command line to create a new policy object.

Enter the following **wcrtpol** command to create a software package policy default object:

```
wcrtpol –d SoftwarePackage DataSPpolicy
BasicSoftwarePackage
```

where:

–d Creates a policy default object.

SoftwarePackage
Specifies the resource type of the new policy default object.

DataSPpolicy
Specifies the name of the new software package policy default object.

To create a software package policy validation object, enter the following command:

```
wcrtpol –v SoftwarePackage DataSPpolicy
BasicSoftwarePackage
```

where:

–v Creates a policy validation object.

SoftwarePackage
Specifies the resource type of the new policy validation object.

DataSPpolicy
Specifies the name of the new software package policy validation object.

After you create a new policy object, you can view the existing policy methods to validate software package properties or operations for a particular policy region using the following commands:

wlspolm
Lists the policy methods for the specified resource. You can list the default or validation policy methods with this command.

wgetpolm
Retrieves the contents of the specified default or validation policy method.

For more information on the **wcrtpol**, **wlspolm**, and **wgetpolm** commands, see the *Tivoli Management Framework: Reference Manual*.

Replacing the Contents of a Policy Method

To define different policies from those inherited by the parent policy object, you must create a script or program and replace the existing policy method with it.

Note: Policy methods run on managed nodes and thus must be UNIX scripts, programs. Use UNIX scripts or programs on UNIX managed nodes. Do not use C shell scripts.

Table 56 shows the context and role required for this task.

Table 56. Roles for replacing policy methods

Activity	Context	Required Role
Replace the content of a policy method	Tivoli Management Regions	super and policy

You must use the command line to replace the content of a policy method.

Enter the following **wputpolm** command to replace the contents of the sp_def_src_host policy method with the contents of the Data_def_file.sh script:

```
wputpolm –d SoftwarePackage DataSPpolicy sp_def_src_host \
< Data_def_file.sh
```

where:

–d Specifies that the method is a policy default method.

SoftwarePackage
Specifies SoftwarePackage as the managed resource for which the policy is defined.

DataSPpolicy
Specifies the DataSPpolicy policy object that contains the default policy method being replaced.

sp_def_src_host
Replaces the contents of the sp_def_src_host default policy method.

< Data_def_file.sh
Redirects the Data_def_file.sh script to the command. The contents of this file replace the existing contents of the sp_def_src_host policy method.

To replace the contents of the sp_val_src_host policy method with the contents of the Data_val_file.sh script, enter the following command:

```
wputpolm –v SoftwarePackage DataSPpolicy sp_val_src_host
 < Data_val_file.sh
```

where:

–v Specifies that the method is a policy validation method.

SoftwarePackage
Specifies SoftwarePackage as the managed resource for which the policy is defined.

DataSPpolicy
Specifies the DataSPpolicy policy object that contains the validation policy method being replaced.

sp_val_src_host
Replaces the contents of the sp_val_src_host validation policy method.

< Data_val_file.sh
Redirects the Data_val_file.sh script to the command. The contents of this file replace the existing contents of the sp_val_src_host policy method. The Data_val_file.sh file must reside in the directory from which you call the **wputpolm** command.

For more information about the **wputpolm** command, see the *Tivoli Management Framework: Reference Manual.*

Assigning Policy to a Policy Region

To change the default policy for a policy region, you must assign policy to the policy region after you have created a new policy object and replaced policy methods.

Table 57 shows the context and role required for this task:

Table 57. Roles for assigning policy

Activity	Context	Required Role
Assign policy to a policy region	Tivoli Management Regions	policy and either senior or super

You can use the desktop or command line to assign policy to a policy region. See the *Tivoli Management Framework: User's Guide* for instructions on using the desktop.

To use the **wsetpr** command to change the default policy in the Data policy region to those methods defined in the DataSPpolicy policy object, enter the following command:

```
wsetpr –d DataSPpolicy SoftwarePackage @PolicyRegion:Data
```

where:

–d DataSPpolicy
Changes the default policy to that defined in the DataSPpolicy object.

SoftwarePackage
Specifies the SoftwarePackage resource type for which the policy is defined.

@PolicyRegion:Data
Specifies the Data policy region for which to assign the policy.

To use the command line to change the validation policy in the Data policy region to those methods defined in the DataSPpolicy policy object, enter the following command:

```
wsetpr –v DataSPpolicy SoftwarePackage @PolicyRegion:Data
```

where:

–v DataSPpolicy
Changes the validation policy to that defined in the DataSPpolicy object.

SoftwarePackage
Specifies the SoftwarePackage resource type for which the policy is defined.

@PolicyRegion:Data
Specifies the Data policy region for which to assign the policy.

For more information about the **wsetpr** command, see the *Tivoli Management Framework: Reference Manual*.

Example: Setting a Default Policy Method

The following example provides the complete command line solution of how to set the software package policy default for the log_path and log_host_name keywords, and how to create and assign policy to a new software package policy object.

1. Create a new default policy object for the SoftwarePackage class by entering the following command:

   ```
   wcrtpol -d SoftwarePackage SP_default BasicSoftwarePackage
   ```

2. Set the new default policy object as the default in the current policy region:

```
wsetdfpol -d SoftwarePackage SP_default
```

3. List the policy default methods for the SoftwarePackage class by entering the following command:

   ```
   wlspolm –d SoftwarePackage
   ```

   where:

   –d Lists the policy default methods for the SoftwarePackage resource type.

   SoftwarePackage
   Specifies the resource whose policy methods are to listed.

   The following default policies are returned:

   ```
   sp_def_properties
   sp_def_src_host
   ```

   The sp_def_properties policy method is used to set software package options (keywords).

4. List the policy default objects that exist for the SoftwarePackage class. The Tivoli Management Framework supports multiple policy default and validation objects so that, for example, you can have one set of policy objects in policy region X and a different set in policy region Y. Use the following command to list the policy default objects:

   ```
   wlspol –d SoftwarePackage
   ```

   where:

   –d Lists the policy default objects for the SoftwarePackage resource type. (To list the policy validation objects, use the **–v** argument.)

   SoftwarePackage
   Specifies the resource whose policy objects are to listed.

   The following policy default objects are returned:

   ```
   BasicSoftwarePackage
   SP_default
   ```

   This command returns only the BasicSoftwarePackage object if you have not created additional policy default objects.

5. Create a script that sets the log_path keyword. The following script, called /tmp/options.sh, accomplishes this and sets the log_host_name keyword.

   ```
   #!/bin/sh
   cat <<EOF
   log_path=/usr/local/log_file
   log_host_name=venere
   EOF
   exit 0
   ```

6. Replace the contents of the sp_def_properties policy method with the new script using the following command:

   ```
   wputpolm –d SoftwarePackage SP_default
   sp_def_properties </tmp/options.sh
   ```

   where:

   –d Specifies that the method is a default policy.

   SoftwarePackage
   Specifies SoftwarePackage as the resource for which the policy is set.

   SP_default
   Specifies SP_default as he policy object for which the policy is set.

   sp_def_properties
   Specifies the sp_def_properties policy method whose contents are to be replaced.

</tmp/options.sh
Redirects the /tmp/options.sh script to the command. This command reads its input from standard input.

7. Extract the current contents of the sp_def_properties policy default method to make sure that another administrator has not modified it.

   ```
   wgetpolm –d SoftwarePackage SP_default sp_def_properties
   ```

   where:

   –d Lists the contents of the sp_def_properties policy default method.

   SoftwarePackage
   Specifies the SoftwarePackage resource whose policy is to be returned.

   SP_default
   Specifies the SP_default policy object whose policy is to be returned.

   sp_def_properties
   Specifies the policy method whose contents are to be returned.

   The contents of this policy method are sent to standard output by default. If the previous command does not return anything, the policy method is not set.

8. Associate the new policy method in the SP_default policy object with the Source policy region:

   ```
   wsetpr –d SP_default SoftwarePackage
   @PolicyRegion:Source
   ```

   where:
   –d SP_default
   Changes the default policy to that defined in the SP_default object.
   SoftwarePackage
   Specifies the SoftwarePackage resource type for which the policy is defined.
   @PolicyRegion:**Source**
   Specifies to change the policy for the Source policy region.

   After setting the policy, every software package created in policy regions whose default policy for the SoftwarePackage resource type is set to SP_default will have the log_path and log_host_name keywords set as specified in the sample script.

Policy Methods

The following default and validation policy methods enable you to control resources and to maintain guidelines when resources are modified or operated on. These methods can differ from policy region to policy region. That is, one policy region could have one set of validation policy methods and another policy region could have another.

Note: The authorization roles listed for each policy method include the roles required to perform the action that calls each policy method, not the role required to create or edit a policy method. For the authorization roles required to create or edit policy, see the roles listed for the **wcrtpol**, **wgetpolm**, and **wputpolm** commands.

sp_def_properties

Generates default properties for a software package.

Syntax
sp_def_properties

Resource
SoftwarePackage

Description
This method generates software package options and sends them to standard output. These options include logging, file permission, and general distribution. When you set the values of this method, list each on a separate line in the form *keyword=value*.

Authorization
senior or super

Return Values
The **sp_def_properties** method returns one of the following

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following displays:

```
Attempt to use the default policy for properties
  (default policy method 'sp_def_properties') for software package
  'package_name^package_version' in policy region 'region_name'
  failed.
```

You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also
sp_val_properties

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_def_src_host

Generates the default source host for a software package.

Syntax
sp_def_src_host

Resource
SoftwarePackage

Description
This method generates the source host for a software package and outputs it to standard output. When you set the value of this method, you must specify a valid managed node name.

Authorization
senior or super

Return Values
The **sp_def_src_host** method returns one of the following

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following displays:

```
Attempt to use the default policy for source host
  (default policy method 'sp_def_src_host') for software package
  'package_name^package_version' in policy region 'region_name'
  failed.
```

You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also
sp_val_src_host and **sp_val_delete_src_host**

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_val_delete_src_host

Validates the removal of the source host for a software package.

Syntax

sp_val_delete_src_host *sp_name src_host*

Resource

SoftwarePackage

Description

This method validates that an administrator can change (thereby removing) the source host, as specified by the *src_host* argument, of an existing software package, as specified by the *sp_name* argument. It is called when the source host for a software package is changed. Use this method, for example, to enforce a restriction that a certain software package must have a specific host as its source host.

This method runs within the policy region of the source host and applies only to packages defined in its profile managers. To validate successfully, sp_val_delete_src_host must return an exit status of 0 and write TRUE to standard output.

Options

sp_name
: The name of the software package whose source host is changed (removed).

src_host
: The name of the managed node that is currently the source host for the software package.

Authorization

admin, senior, or super

Results

The **sp_val_delete_src_host** method writes the following strings to standard output:

TRUE
: Successful validation.

FALSE
: Failed validation.

Return Values

The **sp_val_delete_src_host** method returns one of the following:

E_OK =0
: Successful completion.

E_USAGE =1
: The method encountered an illegal option, argument, or parameter.

E_FAIL =2
: The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following displays:

```
Attempt to use policy method 'sp_val_delete_src_host'
for software package 'package_name^package_version' in policy
region 'region_name'failed with exit code 'N'.
```

where *N* is the exit code returned when the script completed. You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also

sp_def_src_host and **sp_val_src_host**

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_val_name

Validates the proposed name of a software package.

Syntax

sp_val_name *sp_name*

Resource

SoftwarePackage

Description

This method validates the name given to the software package by the *sp_name* argument. The sp_val_name method is invoked only when the software package is created. Note that this method does not set the default name.

This method must have an exit status of 0 and write TRUE to standard output. Otherwise, validation is considered unsuccessful.

Options

sp_name
The name of the software package to create.

Authorization

senior or super

Results

The **sp_val_name** method writes these strings to standard output:

TRUE
Successful validation.

FALSE
Failed validation.

Return Values

The **sp_val_name** method returns one of the following:

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following is displayed:

```
Attempt to use policy method 'sp_val_name' for
software package 'package_name^package_version' in the policy
region 'region_name' failed with exit code 'N'.
```

where *N* is the exit code returned when the script completed. You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_val_operation

Validates the software package operations.

Syntax

sp_val_operation *sp_name sp_operation*

Resource

SoftwarePackage

Description

This method validates a software package operation, such as distribute, distribute and commit, commit, preview, or removal. It is called when a software package operation is attempted. The list of target systems on which the operation will be performed is available to this method through standard input. If validation is not successful, the operation will not occur.

This method must have an exit status of 0 and write TRUE to standard output. Otherwise, validation is considered unsuccessful.

Options

sp_name

The name of the software package that is the subject of the operation being validated.

sp_operation

The software package operation, modified by the distribution type, which determines what kind of distribution takes place (see the **winstsp** command or the Install Software Package dialog box in the Software Package Editor GUI). The possible software package operations are:

- send
- retrieve
- export
- remove
- undo
- accept
- commit
- convert (in scripts, refer to this operation as **import**)
- install

  Options include:
  - install:ALL (for the **all** option of server mode)
  - install:ANY (for the **repair** option of server mode)
  - install:SRC (for the **source** option of server mode)

Authorization

admin, senior, or super

Results

The **sp_val_operation** method writes these strings to standard output:

TRUE

Successful validation.

FALSE

Failed validation.

Return Values

The **sp_val_operation** method returns one of the following

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following is displayed:

```
Attempt to use policy method 'sp_val_operation' for
software package 'package_name^package_version' in the policy
region 'region_name' failed with exit code 'N'.
```

where *N* is the exit code returned when the script completed. You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also

winstsp

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_val_properties

Validates software package properties.

Syntax

sp_val_properties *sp_name*

Resource

SoftwarePackage

Description

This method validates the options that are set for the software package.

The list of options is available to the method through standard input, each listed on a line by itself in the format *sp_keyword=value*. See Table 53 on page 306, as well as Chapter 1, "Editing the Software Package Definition File," on page 1 for a description of the possible keywords.

Use this policy method, for example, to ensure that log files are created on certain managed nodes. You can also prevent distributions to certain target systems or ensure that software packages are not removed using the sp_val_properties method.

This method must have an exit status of 0 and write TRUE to standard output. Otherwise, validation is considered unsuccessful.

Options

sp_name
The name of the software package whose options are validated.

Authorization

senior or super

Results

The **sp_val_properties** method writes the following strings to standard output:

TRUE
Successful validation.

FALSE
Failed validation.

Return Values

The **sp_val_properties** method returns one of the following

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following is displayed:

```
Attempt to use policy method 'sp_val_properties' for
software package 'package_name^package_version' in the policy
region 'region_name' failed with exit code 'N'.
```

where *N* is the exit code returned when the script completed. You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

sp_val_src_host

Validates the proposed source host of a software package.

Syntax

sp_val_src_host *sp_name src_host*

Resource

SoftwarePackage

Description

This method validates the new source host specified by the *src_host* argument for the software package specified by *sp_name*. When a software package source host is changed or set, this method is invoked.

Use this method, for example, to enforce a restriction that a certain software package must have a specific host as its source host.

This method runs within the policy region of the source host and applies only to packages defined in its profile managers. To validate successfully, sp_val_src_host must return an exit status of 0 and write TRUE to standard output.

Options

sp_name
The software package name whose source host was set or changed.

src_host
The name of the managed node being validated.

Authorization

admin, senior, or super

Results

The **sp_val_src_host** method writes the following strings to standard output:

TRUE
Successful validation.

FALSE
Failed validation.

Return Values

The **sp_val_src_host** method returns one of the following

E_OK =0
Successful completion.

E_USAGE =1
The method encountered an illegal option, argument, or parameter.

E_FAIL =2
The method failed due to an error.

A script that completes successfully must exit with a code of either 0 or $E_OK. Otherwise, an exception condition occurs and a message similar to the following is displayed:

```
Attempt to use policy method 'sp_val_src_host' for
software package 'package_name^package_version' in the policy
region 'region_name' failed with exit code 'N'.
```

where *N* is the exit code returned when the script completed. You can use predefined exit codes or any numeric value between 0 and 126. Undefined values, strings, or numeric values that are negative or greater than 126 cause unpredictable behavior and are not recommended.

See Also

sp_def_src_host, **sp_val_delete_src_host**

See Chapter 1, “Editing the Software Package Definition File,” on page 1 for more information on the software package definition format.

Chapter 5. Checking Object Consistency

Software Distribution creates software package objects, which have relationships with other objects such as ManagedNode, PolicyRegion, and ProfileManager. Object consistency is vital to Software Distribution functionality, and in order for Software Distribution to function correctly, the relationships among objects must be maintained. This chapter describes how this consistency is automatically or manually enforced.

Object consistency is checked on software packages by the remove_host operation, which is invoked when a managed node is deleted.

In addition to this operation, an administrator can run the **wchkdb** command to check the Tivoli object database for software package object consistency. This command invokes the check_db and fix_db operations, which check for inconsistencies and fix any that are detected.

When an object is found to be in an inconsistent state, it is moved to the lost-n-found collection. “Moving Objects between Collections” on page 334 provides instructions on moving objects from the lost-n-found collection so that they can be repaired and used.

This default behavior can be changed by using the **wsetspat** command with the **-b** *value* option, which sets the value of the move_removing_host attribute. Possible values are true or false. See “wsetspat” on page 215 for more information on this command.

If move_removing_host is set to false, you must verify and repair problems in the Tivoli resource database using the **wchkdb -u** command. See “The wchkdb Command” on page 334 and the *Tivoli Management Framework: Reference Manual*for more information on this command.

The remove_host Operation

The Tivoli Management Framework automatically invokes the remove_host operation when an administrator deletes a managed node. If the deleted managed node is the source host or log host for a software package, this operation moves the affected software package from the profile manager in which it resides to the lost-n-found collection. It also posts a notice to the ″Software Distribution″ notice group describing the action.

For example, suppose the managed node jupiter is specified as the log host for the DemoSW^1.0 software package. If an administrator deletes jupiter from the Tivoli management region, the remove_host operation detects that the log host specified for the DemoSW^1.0 software package is not valid. This operation then moves the DemoSW^1.0 software package to the lost-n-found collection and logs a notice to the ″Software Distribution″ notice group.

To distribute a software package that was moved to the lost-n-found collection, you must first move the software package back to a valid profile manager. See the following section for instructions on moving objects from the lost-n-found collection.

The wchkdb Command

The **wchkdb** command detects, reports, and corrects inconsistencies in the Tivoli Management Framework environment. Refer to the *Tivoli Management Framework: Reference Manual*for a complete description of this command. In particular, the **wchkdb** command checks the following Software Distribution objects:

- The software package source host to ensure that it is a valid Tivoli object reference; the source host is recorded in the software package as a ManagedNode object reference.
- The managed node on which the Software Distribution log file resides, as indicated by the log_host keyword, to ensure that it is the name of a valid managed node; this managed node is recorded in the software package definition.

If inconsistencies are present when an administrator issues the **wchkdb –u** command, notices are logged to the ″Software Distribution″ and ″Diagnostics″ notice groups. (The **wchkdb –u** command updates the Tivoli object database and fixes discrepancies.)

In addition, software packages with any of these problems are moved from the profile manager in which they reside to the lost-n-found collection. If an object is moved to the lost-n-found collection, a notice to this effect is logged to the ″Software Distribution″ and ″Diagnostics″ notice groups. You can list the contents of the lost-n-found collection using the **wls /lost-n-found** command. This command lists the contents of the collection for each Tivoli management region.

Moving Objects between Collections

The remove_host operation automatically moves software package objects from their original collection (profile manager) to the lost-n-found collection. The objects that reside in lost-n-found reference a deleted software package or managed node. Because you cannot distribute or modify a software package from the lost-n-found collection, you must first move it to a valid profile manager. The **wmvspobj** command enables you to do this. Use this command to regain access to the software package after you correct the problems.

See “wmvspobj” on page 203 for a complete description of the **wmvspobj** command.

Chapter 6. Migrating File Packages to Software Packages

File packages and software packages are supported by TME 10 Software Distribution, Version 3.6.*x* and Software Distribution, Version 4.x, respectively. Both package types can be used in an environment where both products are installed. You can also migrate existing file packages to software packages that can be processed by Software Distribution, Version 4.x.

This chapter assists you in the migration task by providing:

- A table that contains mappings between the Tivoli Software Distribution, Version 3.6 file package keywords and the Software Distribution, Version 4.*x* software package stanzas and attributes (see Table 58 on page 336).
- A table that contains mappings between the Tivoli Software Distribution, Version 3.6 file package commands and the Software Distribution, Version 4.*x* software package commands (see Table 59 on page 339).
- The **wfptosp** migration command (see "wfptosp" on page 346).

Migration Environments

To migrate file package objects, definition files, or blocks to software package objects, definition files, or blocks, you must copy the source files of the file packages that you want to migrate to the Tivoli Software Distribution, Version 4.x server. File package blocks can only be migrated on the following systems:

- Windows
- HP-UX
- Solaris
- AIX

To migrate AutoPack objects or .pak files you must have a Software Distribution server, and the migration must be carried out in a Windows NT environment. The distribution of an AutoPack software package that was migrated from an AutoPack object or .pak file creates the migr_autopack directory on the target system where you are distributing the AutoPack software package. This directory is necessary to install and remove the AutoPack software package. To perform the migration, you can have one of the following environments:

- A Tivoli management region with Tivoli Management Framework, Version 3.7 and Tivoli Software Distribution, Version 4.x installed.

  Using this environment, from the Tivoli management region that has Software Distribution, Version 4.*x* installed, migrate the file package definition files to the software package objects or the software package definition files running the **wfptosp** command from a Software Distribution, Version 4.*x* server. See "Using the Migration Command" on page 345 for a detailed explanation on how to perform the migration steps.
- A Tivoli management region with Tivoli Management Framework, Version 3.6.x and Tivoli Software Distribution, Version 3.6.*x* installed, and a Tivoli management region with Tivoli Management Framework, Version 3.7, Tivoli Software Distribution, Version 3.6, and Software Distribution, Version 4.*x* installed. These Tivoli management regions must be interconnected.

  Using this environment, from a workstation that has Tivoli Software Distribution, Version 3.6.*x* and the Software Distribution, Version 4.*x* server installed, run the **wfptosp** command to migrate:
  - A file package object to a software package object or a software package definition file.

  - An AutoPack object or a .pak file to a software package block
- A Tivoli management region with Tivoli Management Framework, Version 3.7, Tivoli Software Distribution,Version 3.6.*x*, and Software Distribution, Version 4.*x* installed.

  Using this environment you can run the migration performing the following steps:
  1. Ensure that you have a policy region that manages the file packages and the software packages resources.
  2. Perform the migration running the **wfptosp** command from a workstation that has Tivoli Software Distribution, Version 3.6.*x* and the Software Distribution, Version 4.*x* server installed. See "Using the Migration Command" on page 345 for a detailed explanation on how to use the migration command.

  Note: This is one of the possible environments to perform the AutoPack object or .pak file migration.

Mapping File Package Keywords to Software Package Stanzas and Attributes

Table 58 maps the Tivoli Software Distribution, Version 3.6 file package keywords to the corresponding Software Distribution, Version 4.*x* software package stanzas and attributes. Since the structure of a file package is different from the structure of a software package, there is no simple mapping between keywords, but a file package keyword usually maps to an attribute of a stanza. A stanza is a collection of attributes that characterizes objects and actions. The differences between the file package structure and the software package structure do not always allow a direct mapping from keywords to stanzas-attributes, therefore there are some keywords, such as the destination name of a file system object or the program name, that must follow rules. See "Migrating the Destination Name of a File System Object" on page 340 and "Migrating File Package Programs" on page 342 to understand how to migrate the destination name and the source name of a file system object.

Note: In the following table, *os* stands for operating system. For example: *os*_commit_input_path could be: `os2_commit_input_path` or `nt_commit_input_path`.

Table 58. Mappings between keywords and stanzas/attributes

Tivoli Software Distribution, Version 3.6 File Package Keywords	Tivoli Software Distribution, Version 4.x Software Package Stanzas/Attributes
append_log	ignored (append mode always set)
backup_fmt	N/A
create_dirs	file_system_object.create_dirs
default_dest	file_system_object.destination
default_dir_mode default_file_mode	file_system_object.fat_attributes file_system_object.netware_attributes file_system_object.ntfs_attributes file_system_object.unix_attributes
default_mtime	N/A
descend_dirs	file_system_object.descend_dirs
do_checksum	file_system_object.compute_crc
do_compress	file_system_object.compression_method

Table 58. Mappings between keywords and stanzas/attributes (continued)

Tivoli Software Distribution, Version 3.6 File Package Keywords	Tivoli Software Distribution, Version 4.x Software Package Stanzas/Attributes
dos_platform_prefix	N / A
file_cksums	file_system_object.verify_crc
follow_links	link.follow_links
install_progs	file_system_object.destination
keep_paths	file_system_object.destination
list_path	package.log_object_list.location
log_file	package.log_path
log_file_gid	package.log_gid
log_file_mode	package.log_mode
log_file_uid	package.log_user_id
log_host	package.log_host_name
mail_id	package.mail_id
nested_first	**Note:** If it is set to yes, the primary software package is the first that is distributed. If it is set to no, the nested software packages are the first that are distributed depending on the order in which they are specified in the primary file package.
no_overwrite	add_object.replace_if_existing
nt_platform_prefix	file_system_object.destination
nw_bindery	N / A (implemented by Light Client Framework (LCF) security)
nw_broadcast_message	N / A (Tivoli class not yet implemented)
nw_broadcast_mode	N / A (Tivoli class not yet implemented)
nw_context	N / A (implemented by LCF security)
nw_force_disconnect	N / A
nw_on_error_prog_path	execute_user_program.cleanup.name **Note:** The program always runs, not only in case of error.
nw_platform_prefix	file_system_object.destination
nw_tree	N / A (implemented by LCF security)
os_platform_prefix (See the note on page 336.)	file_system_object.destination
*os*_after_input_from_src *os*_before_input_from_src	execute_user_program.during_install. corequisite_files.file.name
*os*_after_input_path os_before_input_path	execute_user_program.during_install.arguments
*os*_after_option	restart.during_install
*os*_after_prog_from_src *os*_before_prog_from_src	execute_user_program.during_install. corequisite_files.file.name
*os*_after_prog_path *os*_before_prog_path	execute_user_program.during_install.path

Table 58. Mappings between keywords and stanzas/attributes (continued)

Tivoli Software Distribution, Version 3.6 File Package Keywords	Tivoli Software Distribution, Version 4.x Software Package Stanzas/Attributes
*os*_after_removal_input_from_src	execute_user_program.during_remove. corequisite_files.file.name
*os*_after_removal_input_path	execute_user_program.during_remove.arguments
*os*_after_removal_option	restart.during_remove
*os*_after_removal_prog_ from_src	execute_user_program.during_remove. corequisite_files.file.name
*os*_after_removal_prog_path	execute_user_program.during_remove.path
*os*_before_skip_non_zero	execute_user_program.during_install.exit_ codes.success
*os*_commit_input_from_src	execute_user_program.during_commit. corequisite_files.file.name
*os*_commit_input_path	execute_user_program.during_commit.arguments
*os*_commit_option	The **wcommtsp -c** command allows you to do a reboot
*os*_commit_prog_from_src	execute_user_program.during_commit. corequisite_files.file.name
*os*_commit_prog_path	execute_user_program.during_commit.path
*os*_on_error_input_from_src	package.add_directory package.add_file
*os*_on_error_input_path	execute_user_program.during_cleanup.arguments **Note:** The program always runs, not only in case of error.
*os*_on_error_option	N / A
*os*_on_error_prog_from_src	package.add_directory package.add_file
*os*_on_error_prog_path	execute_user_program.during_cleanup.arguments **Note:** The program always runs, not only in case of error.
*os*_removal_input_from_src	execute_user_program.during_remove. corequisite_files.file.name
*os*_removal_input_path	execute_user_program.during_remove.arguments
*os*_removal_option	restart.during_remove
*os*_removal_prog_from_src	execute_user_program.during_remove. corequisite_files.file.name
*os*_removal_prog_path	execute_user_program.during_remove.path
post_notice	package.post_notice
postproc	N / A (not implemented on PC managed nodes and endpoints)
preproc	N / A (not implemented on PC managed nodes and endpoints)
prog_env	package.before_prog_env package.after_prog_env execute_user_program.environment
progs_timeout	execute_user_program.timeout
rm_empty_dirs	file_system_object.remove_ empty_dirs

Table 58. Mappings between keywords and stanzas/attributes (continued)

Tivoli Software Distribution, Version 3.6 File Package Keywords	Tivoli Software Distribution, Version 4.x Software Package Stanzas/Attributes
rm_extraneous	file_system_object.remove_ extraneous
skip_older_src	add_object.replace_if_newer
src_after_as_uid	package.after_as_uid
src_after_input_path	package.after_imput_path
src_after_prog_path	package.after_program_path
src_before_as_uid	package.before_as_uid
src_before_input_path	package.before_input_path
src_before_prog_path	package.before_program_path
src_before_skip_non_zero	N / A
src_relpath	file_system_object.location execute_user_program.during_operation.path execute_user_program.during_operation. corequisite_files.file.name
stderr_size	execute_user_program.during_operation. max_stderr_size
stop_on_error	package.stop_on_failure
unix_default_dir_gid	file_system_object.unix_group_id
unix_default_dir_uid	file_system_object.unix_user_id
unix_default_file_gid	file_system_object.unix_group_id
unix_default_file_uid	file_system_object.unix_user_id
unix_platform_prefix	file_system_object.destination
win95_optional_dist	N / A
win95_optional_dist_timeout	N / A
win_optional_dist	N / A
win_optional_dist_timeout	N / A

Mapping File Package and Software Package Commands

Table 59 maps the Tivoli Software Distribution, Version 3.6 file package commands to the corresponding Tivoli Software Distribution, Version 4.x software package commands.

Table 59. Comparison of commands

Tivoli Software Distribution, Version 3.6 Command	Tivoli Software Distribution, Version 4.x Command
N / A	**waccptsp**
wdistfp, **wrestart**	**wcommtsp**
N / A	**wconvspo**
wexprtfp	**wexpspo**
wgetfpattr	**wgetspat**
wgetfpattr	**wgetspgs**
wgetfpattr	**wgetspop**

Table 59. Comparison of commands (continued)

Tivoli Software Distribution, Version 3.6 Command	Tivoli Software Distribution, Version 4.x Command
wimprtfp, **wcrtfpblock**	**wimpspo**
wdinstfp, **wdinstfpblock**	**winstsp**
N / A	**wldsp**
wmvapobj, **wmfpobj**	**wmvspobj**
wrmfp, **wrmfpblock**	**wremovsp**
wsetfpattr	**wsetspat**
wsetfpprgs	**wsetspgs**
wsetfpopts	**wsetspop**
N / A	**wundosp**
N / A	**wuldsp**
N / A	**wversp**
wcpfpblock	N / A
wsetfpcontents	N / A

Migrating the $fpname String

In the Tivoli Software Distribution, Version 3.6.*x* environment the $fpname string is used to identify the name of the file package. The $fpname string is migrated to the Software Distribution, Version 4.*x* environment by using the *fpname* variable defined in the default_variables stanza. The value that the *fpname* variable assumes in the software package is

```
<fpname>="<package.name^package.version>"
```

When you migrate a file package definition file, the default value for *fpname* is SP_^1.0. When you migrate a file package object, the default value for *fpname* is SP_$fpname^1.0. For example, if the name of the file package object is test and you are migrating to a software package definition file, the default value of the *fpname* variable is *fpname*="SP_test^1.0 and the software package definition file will have the following structure:

```
...
   package
        name="test"
        title="Migrated file package"
        version="1.0"
...
  default variables
        fpname="SP_test^1.0"
         ...
```

Migrating the Destination Name of a File System Object

In a file package, the destination path name of a file system object (such as file, directory, or link) depends on the program name and on the following keywords:

- *xxx*_platform_prefix
- default_dest
- keep_paths

In a software package, you must assign a destination name to each file system object and, except for the resolution of variables, Software Distribution, Version 4.*x*

does not perform any checks on file names. For this reason, Software Distribution, Version 4.*x* uses the following procedure to migrate each part of the destination name:

1. The *xxx*_platform_prefix keyword, which represents the first element of the destination path name, is resolved according to the destination platform. It is used to specify platform-specific parts of the destination directory, such as drive letters. Tivoli Software Distribution maps *xxx*_platform_prefix to the *$(target_dir_$(os_name))* variable and then converts the *$(os_name)* variable to the appropriate value for each platform. For example, in a Windows NT environment, nt_platform_prefix is mapped to target_dir_Windows_NT or, if nt_platform_prefix has the value c:\foo_nt, when the software package is distributed to Windows NT workstations, all the files are stored in the c:\foo_nt directory. The following table maps the value that the Tivoli Software Distribution, Version 3.6 *xxx*_platform_prefix keyword assumes with the corresponding value of the Tivoli Software Distribution *os_name* variable.

Table 60. Operating system keywords

Operating System	**os_platform_prefix**	**os_name**
AIX 4.3	unix_platform_prefix	AIX
HP 11	unix_platform_prefix	HP-UX
Solaris Operating Environment 2.6	unix_platform_prefix	SunOS
Solaris Operating Environment 2.7	unix_platform_prefix	SunOS
OS/2 4.0	os2_platform_prefix	OS/2
Windows 95 (or later)	win95_platform_prefix	Windows_95
Windows NT 4.0 (SP6)	nt_platform_prefix	Windows_NT

2. The default_dest keyword is not platform-dependent. It is used if you are distributing a file package to different types of target systems and want the file package to reside in the same place on all the target systems. In a software package, you specify it after the value that indicates the *xxx*_platform_prefix keywords. If the default_dest keyword assumes the value “bar” in the file package, the destination directory for all the files indicated in nt_platform_prefix is:

   ```
   c:\foo_nt\bar
   ```

   This example indicates the link between nt_platform_prefix and default_dest.

   Note: In the files and directories section of a file package, the **–d** option overrides the value for both the *xxx*_platform_prefix and the default_dest keywords if it is used to specify the destination name of a file package.

3. The remaining part of the destination path name is the keep_paths keyword. It depends on the name of each file system object. If the keep_paths keyword is specified in the file package, the full path of the file system object is concatenated to the destination name. If keep_paths is not specified, the last component of the full path name only is used to build the destination name. For example, if keep_paths = y, and you want to install the c:\mydir\myfile.txt file on Windows NT, the destination name should be c:\foo_nt\bar\mydir\myfile.txt. Otherwise, the destination name should be c:\foo_nt\bar\myfile.txt.

Migrating File Package Programs

A file package can contain programs that must run during operations, such as install or remove. A program is defined by several keywords, such as unix_removal_input_path or unix_removal_prog_path, as shown in Table 58 on page 336. Although these keywords are specified with their full names in a file package, in this chapter they are referred to as, for example, input_path or prog_path, removing the platform and the type of operation from their name. The keywords of a program are the following:

prog_path Indicates the full path of the program that must run in the file package.

prog_from_src Indicates whether the program specified resides on the Software Distribution server or locally on the target system.

input_path Indicates the full path of a file to be passed as the second argument to the program specified by the related keyword.

input_from_src
Indicates whether the input file resides on the Software Distribution server or locally on the target system.

option Indicates whether to reboot the target system or restart the operating system.

skip_non_zero
Indicates whether to skip the distribution to a target system if the program specified by the related keyword exits with a non-zero exit code.

as_uid Indicates the UID under which to run the program specified by the related keyword. This keyword is valid for UNIX platforms only.

It is possible to specify multiple files in the prog_path and input_path keywords by using the comma as a separator. These keywords generate different program objects when they are migrated. In the examples in this chapter, only one file is specified at a time.

In a software package, you use the user_program stanza to specify the properties of a program, and place it in the execute_user_program stanza to run the programs inside a software package. Programs cannot reside on the Software Distribution server. They must reside on the local target system only. If you set prog_from_src or input_from_src, you must be sure that the files specified in prog_path or input_path are copied in the target system by making them corequisites of the program.

For the skip_non_zero attribute, you can take advantage of the exit codes customizable program and associate the range from 0x0000 to 0xFFFF with the successful status.

The option keyword is no longer a property of the command software package program. It is migrated using the restart stanza, which the **wfptosp** command inserts after the execute_user_program stanza. Table 61 on page 343 shows how to migrate the option attribute using the restart stanza.

Table 61. Migrating the option keywords

File Package Keyword	Software Package Stanza	Stanza Attributes
*os*_after_option	restart	during_install=after during_remove=none during_undo=none
*os*_after_removal_option	restart	during_install=none during_remove=after during_undo=none
*os*_removal_option	restart	during_install=none during_remove=immediately during_undo=none

After you migrate the keywords, you must decide how to handle the execution of a program in a software package. In a file package a program runs during one of the following operations:

before The program runs before the file package is distributed.

after The program runs after the file package is distributed.

removal
The program runs before removing a file package.

after_removal
The program runs after removing a file package.

commit
The program runs during a file package commit operation.

on_error
The program runs if an error stops the distribution of a file package.

In a software package, the previous operations are migrated using the mappings shown in Table 62.

Table 62. Operation mappings

File Package Operation	Software Package Operation
Before After	Install
Removal After Removal	Remove
Commit	Commit
On_error	Cleanup **Note:** The program always runs, not only in case of error.

In a software package the execute_user_program stanza controls the running of a program and you cannot specify when (before or after) the program runs during a phase. When the program runs depends on the position where execute_user_program is specified in the software package. For example, programs specified at the beginning of a file package run before the programs that are specified at the end of a software package. Programs specified at the end of the software package run as the last actions.

Migrating File Package Programs

In the Tivoli Software Distribution, Version 3.6 product, the src_before_prog_path and src_after_prog_path keywords are migrated to the before_program_path and to the after_program_path attibutes, respectively. These attributes specify the path of the program to be run on the Software Distribution server before or after the build is completed. For this reason, these programs must be stored on the Software Distribution server before you perform the migration. Because the before_prog_path and the after_prog_path can require input programs that are indicated with the before_input_path and the after_input_path attributes, you must store these programs on the Software Distribution server also.

Using the Migration Command

This section describes how to use the **wfptosp** command.

wfptosp

This command performs the migrations shown in Table 63.

Table 63. Migration input and output

Input	Output
File package object	Software package object
	Software package definition file
File package definition file	Software package object
	Software package definition file
File package block	Software package block
AutoPack object or .pak file	Software package block

You must run the **wfptosp** command from the $BINDIR/TME/SWDIS/MIGRATION directory. In a Windows NT environment, you must run the bash shell before running the **wfptosp** command.

Syntax

To migrate from a file package object to a software package object:

wfptosp –c *profile_manager* **–h** *src_host* [**–n** *spobj_name*] [**–d**] *file_package_object*

To migrate from a file package object to a software package definition file:

wfptosp –h *src_host* [**–n** *spobj_name*] **–f** *spd_path* [**–d**] *file_package_object*

To migrate from a file package definition file to a software package object:

wfptosp –c *profile_manager* **–h** *src_host* [**–n** *spobj_name*] *file_package_definition_file*

To migrate from a file package definition file to a software package definition file:

wfptosp –h *src_host* [**–n** *spobj_name*] **–f** *spd_path file_package_definition_file*

To migrate from a file package block to a software package block:

wfptosp -c *profile_manager* **–h** *src_host* [**–n** *spobj_name*] **–p** *spb_path file_package_block*

To migrate from an AutoPack object or a .pak file to a software package block:

wfptosp -c *profile_manager* **–h** *src_host* [**–n** *spobj_name*] **–p** *spb_path* **–a** autopack

Description

This command migrates:

- A file package object or a file package definition file to a software package object. By default, if you are migrating a file package object, the value of the name attribute inside the software package object will be the same as the specified file package object, but prefixed with SP_. The software package object is defined in the profile manager specified in the **–c** argument using the Software Distribution server defined in the **–h** argument. If you are migrating a file package definition file, the value of the name attribute inside the software

package object will be SP_. The file package definition file is migrated to the profile manager specified in the **–c** argument, using the Software Distribution server defined in the **–h** argument.

- A file package object or a file package definition file to a software package definition file. By default, if you are migrating a file package object, the value of the name attribute inside the software package definition file will be the same as the specified file package object, but prefixed with SP_. If you are migrating a file package definition file, the value of the name attribute inside the software package definition file will be SP_. You must specify the name of the software package definition file using the **–f** argument. The software package definition file is created on the Software Distribution server on which you are running the **wfptosp** command.
- A file package block to a software package block. By default, if you are migrating a file package block, the value of the name attribute inside the software package block will be the same as the specified file package block, but prefixed with SP_. The software package block is defined in the profile manager specified in the **–c** argument, using the Software Distribution server defined in the **–h** argument. The Software Distribution server defined in the **–h** argument must be the same machine from which the **wfptosp** command is run when the file package block migration is performed.

  Note: The migration from a file package block to a software package block must be performed on the same platform family as was used to create it. For example, if the file package block was created on a UNIX platform, such as Sun or HP, it must be migrated on the same platform.

  When you perform a file package block migration, all the files contained in the file package block are temporarily stored in the tmp directory on the Software Distribution server. Therefore, you need enough space in that directory to contain all the files stored in the Software Distribution server directory as well as the file package block.

  You cannot migrate file package blocks generated from file packages that have other file packages nested in them.
- An AutoPack object or a .pak file to a software package block. By default, if you are migrating an AutoPack object, the value of the name attribute inside the software package block will be the same as the specified AutoPack object, but prefixed with SP_. The software package block is defined in the profile manager specified in the **–c** argument using the Software Distribution server defined in the **–h** argument. If you are migrating a .pak file, the value of the name attribute inside the software package block will be SP_. The .pak file is migrated to the profile manager specified in the **–c** argument, using the Software Distribution server defined in the **–h** argument. The Software Distribution server defined in the **–h** argument must be the same machine from which the **wfptosp** command is run when the AutoPack migration is performed.

  Note: When you perform an AutoPack migration, all the files contained in the AutoPack are temporarily stored in the tmp directory on the Software Distribution server. Therefore, you need enough space in the tmp directory to contain all the AutoPack files.

This command does not modify the input file.

Options

–c *profile_manager*
Specifies the name of the profile manager where you want to define the software package.

–h *src_host*
Specifies the Software Distribution server where the files in the software package are obtained and where the software package is stored. The Software Distribution server can be any of the available managed nodes where the Software Distribution server component is installed. For additional information on how Software Distribution uses this argument refer to the “Specifying the Software Distribution server in the wfptosp Command” on page 351 section.

–n *spobj_name*
Specifies the value of the name attribute inside the software package.

The software package name can optionally include the version specified in the form *@sp_name^version*. If not specified, the value of the version attribute in the software package is the same as the file package.

If the file package or the AutoPack file does not contain the version, the default is 1.0. Specify this argument if you are migrating from a file package definition file, otherwise the name of the software package will be SP_^1.0.

If the file package or the AutoPack object name contains more than one caret (^), for example, `name1^name2^1.0`, or if you have a caret followed by a non-numeric character, for example, `name1^name2`, specify the **–n** argument followed by the software package name. Otherwise, you will receive an error message.

If the file name contains a period, insert a caret (^) or a period followed by the number that indicates the version of the file package. For example, if the file package name is visio_pro.visio_filepackage_50 the software package name is visio_pro.visio_filepackage_50^1.0 or visio_pro.visio_filepackage_50.1.0. For more details, see “Software Package Version Checking” on page 4.

Note: On the UNIX and NetWare platforms, the caret (^) symbol between the name and version is required. On the Windows NT and OS/2 platforms, specify either *spname^^version* or *“spname^version”*. In the Solaris sh shell environment, specify *“spname^version”*.

–f *spd_path*
Specifies the absolute path in which to store the software package definition file. Do not specify this attribute if you are migrating a software package object.

–d Deletes the file package object if the software package is created successfully.

-p *spb_path*
Specifies the absolute path where to store the software package block.

-a *autopack*
Specifies the name of the AutoPack object or the absolute path of the .pak file to migrate.

file_package
You can specify the following types of file packages:

file_ package_object
Specifies the name of the file package object to migrate.

file_ package_definition_file
Specifies the absolute path of the file package definition file to migrate.

file_package_block
Specifies the absolute path of the file package block to migrate.

Authorization

admin

Return Values

The **wfptosp** command returns one of the following:

0 Indicates that **wfptosp** started successfully.

-1 indicates that **wfptosp** failed due to an error.

Examples

1. To migrate the Notes file package object to the SP_notes^1.0 software package object and define it in the prf_mgr profile manager, enter the following command:

   ```
   wfptosp -c @prf_mgr -h src_host Notes
   ```

2. To migrate the Notes file package object to the migrnotes_sp^1 software package object and define it in the prf_mgr profile manager, enter the following command:

   ```
   wfptosp -c @prf_mgr -h src_host -n @migrnotes_sp ^1 Notes
   ```

3. To migrate the Notes file package object to the notes.spd software package definition file, enter the following command:

   ```
   wfptosp -h src_host -f c:\mysoftpack\notes.spd Notes
   ```

4. To migrate the notes.fpd file package definition file to the Migrates_sp^1.0 software package object and define it in the prf_mgr profile manager, enter the following command:

   ```
   wfptosp -c @prf_mgr -h src_host -n Migrates_sp^1.0
   c:\myfilepack\notes.fpd
   ```

5. To migrate the notes.fpd file package definition file to the notes.spd software package definition file, enter the following command:

   ```
   wfptosp -h src_host -f c:\mysoftwarepack\notes.spd
   c:\myfilepack\notes.fpd
   ```

6. To migrate the notes.fpb file package block to the Migrates_sp^1.0 software package block and define it in the prf_mgr profile manager, enter the following command:

   ```
   wfptosp -c @prf_mgr -h src_host -n Migrates_sp^1.0
   -p c:\mysoftwarepackageblock\notes.spb
   c:\myfilepackageblock\notes.fpb
   ```

7. To migrate the WinZip AutoPack object to the WinZip_sp^1.0 software package block and define it in the prf_mgr profile manager, enter the following command:

   ```
   wfptosp -c @prf_mgr -h src_host -n WinZip_sp^1.0
   -p c:\mysoftwarepackageblock\WinZip.spb
   -a WinZip
   ```

8. To migrate the WinZip.pak file to the WinZip_sp^1.0 software package block and define it in the prf_mgr profile manager, enter the following:

```
wfptosp -c @prf_mgr -h src_host -n WinZip_sp^1.0 -p
c:\mysoftwarepackageblock\WinZip.spb -a c:\temp\WinZip.pak
```

See Also

None.

Specifying the Software Distribution server in the wfptosp Command

When you migrate a file package to a software package, you must specify the Software Distribution server using the **–h** argument. The Software Distribution server is a managed node with the Tivoli Software Distribution, Version 4.*x*server installed. When you perform the migration, all the files contained in the file packages that you want to migrate must be stored on the Software Distribution server. If so, the software package is correctly migrated and no user intervention is required. Tivoli strongly recommends that you copy the required files before performing the migration. If the Software Distribution server that you specify using the **–h** argument does not contain the required files, the software package is created by the **wfptosp** command and the following occurs:

- The following warning message is displayed.

  ```
  Warning: Source files/directories not found
  on host <source_host_name>'
  ```

- By default, a file stanza is created because the migration command treats the last token of the path name as a file and not as a directory.

For example, if the files of the file package definition file are stored in the `c:\wtd_suite\nt_testsuite\test` directory and test is a directory and not a file, the software package definition file will contain an add directory stanza as follows:

```
add directory
    ....
    location= c:\wtd_suite\
    name="nt_testsuite"
    ....
```

and a file stanza as follows:

```
file
    ....
    name="test"
    destination="test"
    ....
```

In this case, you must correct the software package definition file, by removing the file stanza and modifying the add directory stanza as follows:

```
add directory
    ....
    location= c:\wtd_suite\nt_testsuite
    name="test
    ....
```

which represents the correct contents of the file package definition file.

Migrating Nested File Packages to Nested Software Packages

Software Distribution migrates nested file packages to nested software packages using the following rules:

- Nested file packages are migrated to nested software packages assuming that the nested file packages have already been migrated using the standard naming convention, SP_name^1.0. See “Using the Migration Command” on page 345 for more details on the naming convention used to migrate a file package. If you have not already migrated the nested file packages to software packages, the following message is displayed when you migrate the primary file package:

  ```
  The software package object SP_name^1.0 does not exist.
  ```

  However, the primary file package is successfully migrated.

If you have already migrated a file package to a software package definition file, you can edit the primary software package definition file and specify the names of the nested software package definition files with the name attribute.

- The primary file package is migrated to the primary software package.
- Nested file packages are migrated to nested software packages in the same order in which they are specified in the primary file package.

Appendix A. Built-in Variables

The variables shown in Table 64 are provided by Software Distribution. Only these variables can be used without defaults.

Table 64. Built-in variables

Variable Name	Operating Systems	Description
all_users_shell_ programs	All supported Windows platforms	Specifies the All Users Programs folder, for example, on Windows 2000 machines: `C:\Documents and Settings\All Users\ Start Menu\Programs`
all_users_shell_desktop	All supported Windows platforms	Specifies the All Users Desktop folder, for example, on Windows 2000 machines: `C:\Documents and Settings\All Users\ Desktop`
all_users_shell_start_ menu	All supported Windows platforms	Specifies the All Users Start Menu folder, for example, on Windows 2000 machines: `C:\Documents and Settings\All Users\ Start Menu`
all_users_shell_startup	All supported Windows platforms	Specifies the All Users Startup folder, for example, on Windows 2000 machines: `C:\Documents and Settings\All Users\ Start Menu\Programs\Startup`
cdrom_*xx*	All	Specifies the letter of the cd drive or volume with a CD-ROM inserted. The value for *xx* is a two-digit number ranging from 01 through 26. For example, cdrom_*01* refers to the first available CD-ROM drive, while cdrom_*05* refers to the fifth available CD-ROM drive.
computer_name	All	Specifies the computer name of the target system.
common_files	All supported Windows platforms	Specifies the directory where applications can place common, shared information, for example: `C:\Program Files\Common Files`
ep_label	All	Specifies the name of an endpoint. **Note:** This variable cannot be resolved in a disconnected environment. It must not be used in packages that are to be available to Web Interface or any other disconnected environment.
freedrive_*xx*	All supported Windows platforms, OS/2	Specifies an available drive. The value for *xx* is a two-digit number ranging from 01 to 26. For example, freedrive_01 refers to the first available drive, while freedrive_05 refers to the fifth available drive.

Table 64. Built-in variables (continued)

Variable Name	Operating Systems	Description
home_path	All supported Windows platforms, UNIX	Specifies the home path of the user currently logged on, if any.
hostname	All	Specifies the TCP/IP host name of the target system.
operation_name	All	Specifies the name of the current operation. This variable can be used only as an argument to a user program. The possible values are the following: • install • remove • undo • accept • commit • verify
operation_phase	All	Specifies the phase of the current operation. This variable can be used only as an argument to a user program. The possible values are the following: • prepare • commit • backup • rollback • prepare_cleanup • commit_cleanup • rollback_cleanup • commit_before_reboot • commit_after_reboot
operation_result	All	Specifies the result of the last operation performed. This variable can be used only as an argument to a user program. The possible values are the following: • success • temporary_failure • failure • fatal_failure
os_family	All	Specifies characteristics that are common to all the operating system platforms in a family. For example, the following statement could be used to customize the names of the UNIX targets of an installation: `target_dir.UNIX=/target` For the possible values of *os_family*, see Table 65 on page 356.

Table 64. Built-in variables (continued)

Variable Name	Operating Systems	Description
os_name	All	Specifies the operating system name. For the possible values of *os_name*, see Table 65 on page 356. **Note:** When specifying the OS/2 operating system, the name must be included in single quotes, for example: `condition="$(os_name) =='OS/2'"`
os_release	All	Specifies the operating system release. For the possible values of *os_release*, see Table 65 on page 356.
os_version	All	Specifies the operating system version. For the possible values of *os_version*, see Table 65 on page 356.
os_architecture	All	Specifies the operating system architecture.
os2_desktop	OS/2	Specifies the path of the active OS/2 desktop.
product_dir	All	Specifies the Software Distribution installation directory. It is defined in the configuration file (swdis.ini), and you should edit the *product_dir* variable in this file if you need to modify the installation directory. The file location varies from system to system.
program_files	All supported Windows platforms	Specifies the directory where program files are stored (for example, C:\Program Files).
system16_dir	All supported Windows platforms	Specifies the Windows 16-bit system directory.
system_dir	All supported Windows platforms	Specifies the Windows system directory.
system_drive	All supported Windows platforms, OS/2, NetWare	Specifies the system drive of the target system.
system_root	All supported Windows platforms, OS/2	Specifies the system root of the target system.
temp_dir	All	Specifies the temporary directory. In a connected environment, the value of *temp_dir* is determined by the settings on the Tivoli endpoint.
user_domain	Windows 2000	Specifies the Windows user domain.
user_name	Windows 2000	Specifies the name of the user currently logged on.

Table 64. Built-in variables (continued)

Variable Name	Operating Systems	Description
user_profile_dir	All supported Windows platforms	Specifies the user profile directory of the user currently logged on (for example, C:\WINNT\Profiles*UserName*).
user_shell_desktop	All supported Windows platforms	Specifies the shell desktop directory of the user currently logged on.
user_shell_programs	All supported Windows platforms	Specifies the shell programs directory of the user currently logged on.
user_shell_start_menu	All supported Windows platforms	Specifies the shell start menu directory of the user currently logged on.
user_shell_startup	All supported Windows platforms	Specifies the shell startup folder of the user currently logged on.

Table 65 shows the possible values for the *os_family*, *os_name*, *os_release*, and *os_version* variables, depending on the operating system. The list of values for a particular platform (except for NetWare) can also be generated by running the **uname** command.

Table 65. Values of operating system variables

Operating System Name	os_name	os_family	os_release	os_version	os_architecture
AIX 4.3.3	AIX	UNIX	3	4	RISC
AIX 5.1	AIX	UNIX	1	5	RISC
AIX 5.2	AIX	UNIX	2	5	RISC
HP 11	HP-UX	UNIX	B.11.00	A	9000_785
HP 11 i	HP-UX	UNIX	B.11.11	U	9000_785
NetWare Server 5	NETWARE	NETWARE	0	5	IX86
NetWare Server 6	NETWARE	NETWARE	6	5	IX86
Solaris 2.7	SunOS	UNIX	5.7	Generic	SUN4U
Solaris 2.8	SunOS	UNIX	5.8	Generic	SUN4U
Solaris 2.9	SunOS	UNIX	5.9	Generic_112233_01	SUN4U
OS/2 4.0[1]	OS/2	PC	2	2.40	IX86
OS/2 4.5[1]	OS/2	PC	2	2.45	IX86
OS 400 4.3	OS 400	OS 400	V4R3M0	V4R3M0	AS400
OS 400 4.4	OS 400	OS 400	V4R4M0	V4R4M0	AS400
OS 400 4.5	OS 400	OS 400	V4R5M0	V4R5M0	AS400
OS 400 5.1	OS 400	OS 400	V5R1M0	V5R1M0	AS400
OS 400 5.2	OS 400	OS 400	V5R2M0	V5R2M0	AS400
Windows 2000 (SP2)	Windows_NT	PC	5.0	2195+ Service_Pack_2	IX86

Table 65. Values of operating system variables (continued)

Operating System Name	os_name	os_family	os_release	os_version	os_architecture
Windows XP	Windows_NT	PC	5.1	2600	IX86
Windows 2003	Windows_NT	PC	5.2	3790	IX86
Linux Intel	Linux	UNIX	[2]	[2]	IX86
Linux OS390	Linux	UNIX	[2]	[2]	S390
Linux PPC	Linux	UNIX	[2]	[2]	PPC

Notes:

1. When specifying the OS/2 operating system, the value for *os_name* must be included in single quotes, for example:

   ```
   condition="$(os_name) =='OS/2'"
   ```

2. This value varies depending on the operating system you have installed.

Appendix B. Support information

This section describes the following options for obtaining support for IBM products:

Searching knowledge bases

If you have a problem with your IBM software, you want it resolved quickly. Begin by searching the available knowledge bases to determine whether the resolution to your problem is already documented.

Search the information center on your local system or network

IBM provides extensive documentation that can be installed on your local computer or on an intranet server. You can use the search function of this information center to query conceptual information, instructions for completing tasks, reference information, and support documents.

Search the Internet

If you cannot find an answer to your question in the information center, search the Internet for the latest, most complete information that might help you resolve your problem. To search multiple Internet resources for your product, expand the product folder in the navigation frame to the left and select **Web search**. From this topic, you can search a variety of resources including:

- IBM technotes
- IBM downloads
- IBM Redbooks
- IBM developerWorks
- Forums and newsgroups
- Google

Obtaining fixes

A product fix might be available to resolve your problem. You can determine what fixes are available for your IBM software product by checking the product support Web site:

1. Go to the IBM Software Support Web site (http://www.ibm.com/software/support).
2. Under **Products A - Z**, select your product name. This opens a product-specific support site.
3. Under **Self help**, follow the link to **All Updates**, where you will find a list of fixes, fix packs, and other service updates for your product. For tips on refining your search, click **Search tips**.
4. Click the name of a fix to read the description and optionally download the fix.

To receive weekly e-mail notifications about fixes and other news about IBM products, follow these steps:

1. From the support page for any IBM product, click **My support** in the upper-right corner of the page.
2. If you have already registered, skip to the next step. If you have not registered, click register in the upper-right corner of the support page to establish your user ID and password.
3. Sign in to **My support**.
4. On the My support page, click **Edit profiles** in the left navigation pane, and scroll to **Select Mail Preferences**. Select a product family and check the appropriate boxes for the type of information you want.
5. Click **Submit**.
6. For e-mail notification for other products, repeat Steps 4 and 5.

For more information about types of fixes, see the *Software Support Handbook* (http://techsupport.services.ibm.com/guides/handbook.html).

Contacting IBM Software Support

IBM Software Support provides assistance with product defects.

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus, and Rational products, as well as DB2 and WebSphere products that run on Windows or UNIX operating systems), enroll in Passport Advantage in one of the following ways:
  - **Online**: Go to the Passport Advantage Web page (http://www.lotus.com/services/passport.nsf/WebDocs/ Passport_Advantage_Home) and click **How to Enroll**
  - **By phone**: For the phone number to call in your country, go to the IBM Software Support Web site (http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.
- For IBM eServer software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web page (http://www.ibm.com/servers/eserver/techsupport.html).

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States or, from other countries, go to the contacts page of the IBM Software Support Handbook on the Web (http://techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region for phone numbers of people who provide support for your location.

Follow the steps in this topic to contact IBM Software Support:

1. Determine the business impact of your problem.
2. Describe your problem and gather background information.
3. Submit your problem to IBM Software Support.

Determine the business impact of your problem

When you report a problem to IBM, you are asked to supply a severity level. Therefore, you need to understand and assess the business impact of the problem you are reporting. Use the following criteria:

Severity 1	**Critical** business impact: You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.
Severity 2	**Significant** business impact: The program is usable but is severely limited.
Severity 3	**Some** business impact: The program is usable with less significant features (not critical to operations) unavailable.
Severity 4	**Minimal** business impact: The problem causes little impact on operations, or a reasonable circumvention to the problem has been implemented.

Describe your problem and gather background information

When explaining a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- What software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can the problem be re-created? If so, what steps led to the failure?
- Have any changes been made to the system? (For example, hardware, operating system, networking software, and so on.)
- Are you currently using a workaround for this problem? If so, please be prepared to explain it when you report the problem.

Submit your problem to IBM Software Support

You can submit your problem in one of two ways:

- **Online**: Go to the ″Submit and track problems″ page on the IBM Software Support site (http://www.ibm.com/software/support/probsub.html). Enter your information into the appropriate problem submission tool.
- **By phone**: For the phone number to call in your country, go to the contacts page of the IBM Software Support Handbook on the Web (techsupport.services.ibm.com/guides/contacts.html) and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround for you to implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the IBM product support Web pages daily, so that other users who experience the same problem can benefit from the same resolutions.

For more information about problem resolution, see Searching knowledge bases and Obtaining fixes.

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents.You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy form, the photographs and color illustrations might not appear.

Trademarks

IBM, the IBM logo, developerWorks, eServer, iSeries, Lotus, Passport Advantage, pSeries, Rational, Redbooks, Tivoli, the Tivoli logo, Tivoli Enterprise, Tivoli Enterprise Console, NetView, Wake on LAN, AIX, AS/400, DB2, OS/2, OS/400, WebSphere, zSeries are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

Lotus, and Lotus Notes, are trademarks of International Business Machines Corporation and Lotus Development Corporation in the United States, other countries, or both.

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Intel is a trademark or registered trademark of Intel Corporation in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Glossary

A

accept operation. An operation that deletes the backup software package so that the previous operation cannot be restored.

admin role. See authorization role.

authorization role. In a Tivoli environment, a role assigned to Tivoli administrators to enable them to perform their assigned systems management tasks. A role may be granted over the entire Tivoli management region or over a specific set of resources, such as those contained in a policy region. Examples of authorization roles include super, senior, admin, and user. See also role.

AutoPack. In Tivoli Software Distribution, Version 4, a tool that enables a Tivoli administrator to create a software package. AutoPack produces the software package by (a) taking snapshots of the drive and system configuration before and after the installation of an application on a PC and (b) capturing the differences between these snapshots in the software package.

AutoPack file. In Tivoli Software Distribution, Version 3, an installable image that is used to distribute shrink-wrapped applications to multiple PC targets. The file contains PC software application files and directories, information on how to distribute these files and directories, and any system configuration changes needed by the application. A Tivoli administrator must associate an AutoPack file with an AutoPack profile.

B

BAROC. See Basic Recorder of Objects in C (BAROC)

Basic Recorder of Objects in C (BAROC). In the event server of the Tivoli Enterprise Console product, the internal representation of the defined event classes.

base package. The name and version of a software package that is installed on a system.

byte-level differencing. The process of detecting the differences, or the delta, between the software package to be installed and the base software package, and creating a delta software package. See also delta install.

C

collection. In a Tivoli environment, a container that provides a single view of related resources.

commit operation. (1) In Tivoli Software Distribution, Version 4, a change management operation that causes all the updates prepared in the preparation phase to take effect. (2) In Tivoli Software Distribution, Version 3, an operation performed by a configuration program on target managed nodes after a file package distribution. This function enables a Tivoli administrator to distribute a file package to multiple targets over time and then make the distributed information effective on all targets at the same time.

commit phase. In Tivoli Software Distribution, Version 4, the phase of transactional mode operation in which previously prepared actions are committed, causing all of the updates to take effect. See also preparation phase

configuration repository. In a Tivoli environment, a RIM repository that contains information that is collected or generated and stored by inventory scans and software distributions.

D

default policy. In a Tivoli environment, a set of resource property values that are assigned to a resource when the resource is created.

delta install. The process of creating the software package that contains only the delta between the base software package and the software package to be installed. By creating and distributing a delta software package, network traffic is reduced.

depot. See repeater depot.

device. Any non-client, non-server part of a network managed by Tivoli software, including, but not limited to, Palm devices, handheld PCs, cable set-top boxes, and other pervasive devices.

differencing phase. In Tivoli Software Distribution, Version 4, the process by which AutoPack examines and compares before and after snapshots and, for each difference found, generates the related action and adds it to a software package.

disconnected target command. In Tivoli Software Distribution, Version 4, a command that is run from the command line of a target system that is not connected to the Tivoli management region server.

discovered software package. In Tivoli Software Distribution, Version 4.1, an application that was installed on an endpoint independently of Tivoli Software Distribution and that was later added to the endpoint catalog and given a status of “installed and

discovered," using a Tivoli Software Distribution disconnected target command.

discovery. A means of synchronizing the software-package status information on the server with that for software packages on specified endpoints.

Distribution Status console. An MDist 2 interface provided by Tivoli Management Framework that enables administrators to monitor and control distributions across a network. See also MDist 2.

domain. A user-defined subdivision that is based on groups and hierarchies of components in subsystems. Multiple domains can exist and be nested within or overlap other domains.

E

endpoint. In a Tivoli environment, the agent that is the ultimate recipient for any type of Tivoli operation. See Tivoli management agent.

endpoint list. In a Tivoli environment, a list of all endpoints in a Tivoli management region with their assigned gateways. See endpoint manager.

endpoint manager. In a Tivoli environment, a service that runs on the Tivoli management region server, assigns endpoints to gateways, and maintains the endpoint list.

endpoint method. In a Tivoli environment, a method that runs on an endpoint as the result of a request from another managed resource. Results of the method are forwarded to the gateway, and then to the calling managed resource.

event. Any significant change in the state of a system resource, network resource, or network application. An event can be generated for a problem, for the resolution of a problem, or for the successful completion of a task.

event adapter. In a Tivoli environment, software that converts events into a format that the Tivoli Enterprise Console product can use and forwards the events to the event server.

event class. A classification for an event that indicates the type of information that the event adapter can send to the event server. See also Basic Recorder of Objects in C (BAROC).

event console. In the Tivoli Enterprise Console product, a graphical user interface that enables system administrators to view and respond to dispatched events from the event server.

event database. A RIM database that contains event-related information that is collected or generated by the Tivoli Enterprise Console product.

event filter. (1) In a Tivoli environment, rules that determine which events are sent from an event adapter or displayed on an event console. Also used to determine which events a specific correlation rule will apply to. (2) In Tivoli NetView, a logical expression of criteria that determines which events are forwarded to the application program that registers the event filter with the event sieve agent. (3) In the Tivoli Enterprise Console product, the criteria that must be met by an event before a rule action is executed.

event group. In the Tivoli Enterprise Console product, a set of events that meet certain criteria defined by event group filters, which include constraints that are expressions that define the filter conditions. Event console operators can monitor event groups that are relevant to their specific areas of responsibility. See also event filter.

event server. A server program that processes events.

F

file package. In Tivoli Software Distribution, Version 3, a profile. The file package describes which files and directories to distribute and how to distribute them.

file package block. In Tivoli Software Distribution, Version 3, a static file containing (a) the file package definition, (b) the file package attributes, (c) the source files and directories, and (d) the configuration programs of the file package.

file package definition. In Tivoli Software Distribution, Version 3, an ASCII file that identifies the contents and characteristics of a file package.

fpblock. See file package block.

G

gateway. Software that provides services between the endpoints and the rest of the Tivoli environment.

gateway method. A method that runs on behalf of an endpoint on the gateway to which the endpoint is assigned. The results of the method are forwarded to the managed resource that requested that the method be run.

J

job. A resource consisting of a task and its preconfigured parameters. Among other things, the parameters specify the targets on which the job is to run.

L

lcfd. The Tivoli service that is used by an endpoint to communicate with a gateway. Contrast with oserv.

lenient distribution. The process of distributing software packages to endpoints, managed nodes, or profile managers that are not current subscribers of the profile manager to which the software packages belong.

M

managed node. In a Tivoli environment, a computer system on which Tivoli Management Framework is installed. Contrast with endpoint.

managed resource. In a Tivoli environment, a database object that represents a resource and is governed by policies. See also policy and resource.

MDist. A multiplexed distribution service provided by Tivoli Management Framework that enables efficient transfer of data to multiple targets. Another multiplexed distribution service, MDist 2, provides additional management features. See also MDist 2.

MDist 2. A multiplexed distribution service provided by Tivoli Management Framework that enables efficient transfer of data to multiple targets. Administrators can monitor and control a distribution throughout its life cycle. Another multiplexed distribution service, MDist, lacks these management features. See also Distribution Status Console.

multiplexed distribution. The mechanism used by Tivoli Enterprise applications to transfer data to multiple targets. Tivoli Management Framework provides two multiplexed distribution services, MDist and MDist 2. See also MDist and MDist 2.

N

name registry. See Tivoli name registry.

notice. In a Tivoli environment, a message generated by a systems management operation that contains information about an event or the status of an application. Notices are stored in notice groups. See also notice group.

notice group. In a Tivoli environment, an application- or operation-specific container that stores and displays notices that pertain to specific Tivoli functions. The Tivoli bulletin board comprises notice groups. A Tivoli administrator can subscribe to one or more notice groups. The administrator's bulletin board contains only the notices that reside in a notice group to which the administrator is subscribed. See also "notice."

notification manager. A software distribution service that handles message queues.

O

OID. An object identifier.

object reference. In a Tivoli environment, the object identifier (OID) that is given to an object during its creation.

object request broker (ORB). In object-oriented programming, software that serves as an intermediary by transparently enabling objects to exchange requests and responses.

ORB. See object request broker.

oserv. The Tivoli service that is used as the object request broker (ORB). This service runs on the Tivoli management region server and each managed node. Contrast with lcfd.

P

PDA. See personal data assistant.

personal data assistant (PDA).. A handheld device that is used for personal organization tasks (such as calendaring, note-taking, and recording telephone and fax numbers), and networking functions such as e-mail and synchronization.

policy. In a Tivoli environment, a set of rules that are applied to managed resources.

policy domain. A grouping of policy users with one or more policy sets, which manage data or storage resources for the users. The users can be client nodes or agents on managed hosts.

policy region. A group of managed resources that share one or more common policies and which model the management or organizational structure of a network computing environment. Administrators use policy regions to group similar resources, to define access to the resources, to control the resources, and to associate rules for governing the resources.

preparation machine. (1) In Tivoli Software Distribution, Version 4, any Windows 95 (or later), Windows NT, or OS/2 system on which the Tivoli Software Distribution Java Endpoint Package Editor is installed. (2) In Tivoli Software Distribution, Version 3, any Windows system on which the AutoPack Control Center is installed.

preparation phase. In Tivoli Software Distribution, Version 4, the phase of transactional mode operation in which each action in a software package prepares the conditions for the successful execution of an install or remove operation. If the preparation phase fails, the target system is returned to its original, stable state. See also commit phase.

Pristine tool. A self-contained application used for preparing and storing operating system images and configuration information to be installed on pristine systems.

profile. In a Tivoli environment, a container for application-specific information about a particular type of resource. A Tivoli application specifies the template for its profiles, which includes information about the resources that can be managed by that Tivoli application.

profile manager. In a Tivoli environment, a container for profiles that links the profiles to a set of resources, called subscribers. Tivoli administrators use profile managers to organize and distribute profiles. A profile manager can operate in the dataless mode or database mode.

Q

query. In a Tivoli environment, a combination of statements that are used to search the configuration repository for systems that meet certain criteria. The query object is created within a query library. See also query library.

query library. In a Tivoli environment, a facility that provides a way to create and manage Tivoli queries. See also query.

R

RDBMS Interface Module (RIM). In Tivoli Management Framework, the module in the distributed object database that contains information about the installation of the relational database management system (RDBMS).

reference model. In the context of Tivoli software, the model configuration for a system or set of systems that is used to maintain consistent configurations in a distributed environment. In Tivoli Inventory, reference models are created in the configuration repository.

repeater. In a Tivoli environment, a managed node that receives a single copy of data and distributes it to the next tier of clients.

repeater depot. A data repository for temporary or permanent storage of distribution data.

repeater range. The Tivoli clients that receive data from a repeater site.

resource. A hardware, software, or data entity that is managed by Tivoli software. See also managed resource

resource gateway. In a Tivoli environment, the software that enables Tivoli applications to gain access to resources, such as pervasive devices, and perform operations on those resources.

resource manager. An extension of the Tivoli Management Framework that manages resources, like pervasive computing devices and users.

resource role. In a Tivoli environment, the role an administrator has over specific resources in the local Tivoli management region (region) and any connected region (for example, policy regions or the Administrator collection).

RIM. See RDBMS Interface Module.

RIM host. In a Tivoli environment, the managed node on which one or more RIM objects is installed. See also RIM object.

RIM object. An object that provides the attributes and methods that enable applications to access an RDBMS.

RIM repository. In a Tivoli environment, a relational database that contains information that is collected or generated by Tivoli applications. Examples of a RIM repository include the configuration repository and the event database.

role. A job function that identifies the tasks that a user can perform and the resources to which a user has access. A user can be assigned one or more roles.

root administrator. In a Tivoli environment, the initial Tivoli administrator that is created during the installation of the Tivoli Management Framework. This administrator is the root user on UNIX systems and a member of the administrator group on Microsoft Windows systems.

rule. A set of logical statements that enable the event server to recognize relationships among events and to execute automated responses accordingly. See also rule base.

rule base. One or more rule sets and the event class definitions for which the rules are written. The Tivoli Enterprise Console product uses the rule base in managing events. An organization can create many rule bases, with each rule base fulfilling a different set of needs for network computing management. However, only one rule base can be active at a time.

S

scanner. The software used to gather hardware information and software information from systems and devices.

senior role. See authorization role.

signature. In Tivoli Software Distribution, Version 4, unique identifying information that forms the first line in a software package definition file. Tivoli Software Distribution uses the signature for version checking and for determining what kind of importer to use to read the software package definition file.

signature package. A logical grouping of two or more signatures.

software package. In Tivoli Software Distribution, Version 4, a database object that contains a sequential list of actions to be executed on a target system.

software package block. In Tivoli Software Distribution, Version 4, a file that contains the resources referred to by the actions in a software package.

software package definition. In Tivoli Software Distribution, Version 4, an ASCII text file used to describe package contents. It consists of a sequence of stanzas that describe commands to be executed. See also stanza.

Software Package Editor. In Tivoli Software Distribution, Version 4, a graphical user interface (GUI) for creating and customizing software packages.

source host. The managed node on which the source files and directories referred to in a software package or a file package reside. See also software package and file package.

stanza. In Tivoli Software Distribution, Version 4, a section of a software package definition. A stanza can define, for example, an action to be performed, a list of targets on which the action is to be performed, or a set of conditions under which an action is to be executed. Stanzas can be nested, and there is a single stanza (the root stanza) that contains the entire software package definition.

subscriber. In a Tivoli environment, a resource that is subscribed to a profile manager.

subscription. In a Tivoli environment, the process of identifying the subscribers to which profiles will be distributed.

super role. See authorization role.

T

target. In Tivoli Software Distribution, Version 4, a workstation on which the actions defined in a software package are executed. The Tivoli management agent must be installed on the workstation.

task. (1) An activity that has business value, is initiated by a user, and is performed by software. (2) In a Tivoli environment, the definition of an action that must be routinely performed on various managed resources throughout the network. A task defines the executables to be run when the task is executed, the authorization role required to execute the task, and the user or group name under which the task will execute.

task library. In a Tivoli environment, a container in which a Tivoli administrator can create and store tasks and jobs.

Tivoli administrator. In a Tivoli environment, a system administrator who has been authorized to perform systems management tasks and manage policy regions in one or more networks.

Tivoli client. A client of a Tivoli server. See also Tivoli management region server.

Tivoli desktop. In the Tivoli environment, the desktop that system administrators use to manage their network computing environments.

Tivoli environment. The Tivoli applications, based upon the Tivoli Management Framework, that are installed at a specific customer location and that address network computing management issues across many platforms.

Tivoli management agent. In a Tivoli environment, an agent on the endpoint that securely performs administrative operations.

Tivoli management gateway. In a Tivoli environment, a system that enables bidirectional communication with Tivoli management agents.

Tivoli management region. In a Tivoli environment, a Tivoli server and the set of clients that it serves. An organization can have more than one region. A Tivoli management region addresses the physical connectivity of resources whereas a policy region addresses the logical organization of resources.

Tivoli management region server (Tivoli server). The server for a specific Tivoli management region that holds or references the complete set of Tivoli software, including the full object database.

Tivoli name registry. In a Tivoli environment, the table that maps names of managed resources to resource identifiers (and the corresponding information) within a Tivoli management region.

Tivoli server. See Tivoli management region server.

transactional mode. In Tivoli Software Distribution, Version 4, a mode of executing install or remove operations in two phases: the preparation phase and the commit phase. See also preparation phase and commit phase.

U

undoable-in-transactional mode. In Tivoli Software Distribution, Version 4, a variation of transactional mode in which disk space for backup copies (required for undoability) is reserved during the preparation phase, which minimizes the risk of failure attributable to insufficient disk space during the commit phase.

undoable mode. In Tivoli Software Distribution, Version 4, a mode of operation in which actions can be rolled back even if they are already committed, because a backup copy is saved.

user. A person who uses Tivoli management software and is assigned one or more roles.

user role. See authorization role.

V

validation policy. In a Tivoli environment, the policy that ensures that all resources in a policy region comply with the region's established policy. Validation policy prevents Tivoli administrators from creating or modifying resources that do not conform to the policy of the policy region in which the resources were created. Contrast with default policy.

W

Web Gateway. A Tivoli software component that extends Tivoli enterprise management capabilities to the Web environment, enabling the Inventory and Software Distribution components to manage Web-attached pervasive devices like PDAs, handheld PCs, and phones. In addition, users can now manage their Web-attached workstations by connecting to the Web Gateway server and pulling down application updates to their workstations on request.

Index

Special characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

Program Number: 5724-C06

Printed in USA

SC23-4712-05