What's New in the Sun N1 Service Provisioning System 5.2 Update 2 Release

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–0251
January 2007

070125@16599

Contents

Preface

The *What's New in the Sun N1 Service Provisioning System 5.2 Update 2 Release* document describes how to install and use the new features in the Sun N1™ 5.2 Update 2 release.

Who Should Use This Book

This book is for system administrators and IT operators who are responsible for configuring and maintaining the Sun N1 Service Provisioning System.

Before You Read This Book

This book assumes that you have already installed the Sun N1 Service Provisioning System 5.2 release in your data center environment. For information about how to install the Sun N1 Service Provisioning System 5.2 release, see the following books:

- *Sun N1 Service Provisioning System 5.2 Installation Guide*.
- *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

How This Book Is Organized

This book explains the following topics:

- Chapter 1 describes how to install the software on SPARC and x86 based systems.
- Chapter 2 describes the new variable substitution feature that enables dynamic targeting, which repository component targeters it affects, and explains how to use it.
- Chapter 3 describes a new feature for this release of the N1 Service Provisioning System, which allows you to supply component variable values in the `<install>` step of a plan.
- Chapter 4 describes the new locking feature, and explains how to use it.
- Chapter 5 describes an import-export feature, which is a new, simple way to copy objects such as components or plans between master servers.
- Appendix A provides details of new commands provided to support the import-export feature.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

Typographic Conventions

The following table describes the typographic conventions that are used in this book.

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename.*

TABLE P–1 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

Shell	Prompt
C shell	`machine_name%`
C shell for superuser	`machine_name#`
Bourne shell and Korn shell	`$`
Bourne shell and Korn shell for superuser	`#`

◆ ◆ ◆ CHAPTER 1

Installing the N1 Service Provisioning System 5.2 Update 2

This release contains scripts that migrate your master server to run N1 Service Provisioning System 5.2 Update 2 from compatible previous releases and compatible previous patch updates.

Installing the N1 Service Provisioning System 5.2 Update 2 Release

This section describes how to install N1 Service Provisioning System 5.2 Update 2. This section explains the following topics:

You can check which version of the N1 Service Provisioning System master server software you are running by using the `cr_server` command as described in the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

You can check which version of the N1 Service Provisioning System command line interface software you are running by using the `cr_cli -version` command as described in the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

Supported Platforms and Operating Systems

The supported platforms and operating systems for this release are as follows.

Remote agents and local distributors	All operating systems and hardware platforms are supported as listed in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide*.

Master server and command line interface	All hardware platforms are supported as listed in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide*. Only those versions of the Solaris operating system listed in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide* are supported. Other listed operating systems are not supported in this release.

▼ How to Acquire the N1 Service Provisioning System 5.2 Update 2 Release

The N1 Service Provisioning System 5.2 Update 2 release is available from `http://www.sun.com`. This procedure describes how to acquire this release from the specific page `http://www.sun.com/software/swportfolio/get.jsp`

You can also download the release patches from the SunSolve site at `http://sunsolve.sun.com` Download the appropriate patches as follows:

- SPARC– based systems:
  - 122989-12 for the Master Server
  - 122991-12 for the Command Line Interface
- x86-based systems:
  - 122990-12 for the Master Server
  - 122992-12 for the Command Line Interface

Before You Begin Make sure your system meets the hardware requirements set out in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide*.

1. **In your web browser, type the URL `http://www.sun.com/software/swportfolio/get.jsp`.**

   The Sun's Software Portfolio download page is displayed.

2. **In the Select Your Software section, click the check box next to Sun N1 Service Provisioning System.**

3. **Click the Get Downloads Media button.**

4. **If prompted, log in or register for a Sun Online account.**

   The Download Sun Products page is displayed.

5. **In the My Downloads table, click the arrow beneath the product platform that you want to download.**

   The Download Page is displayed.

6. **Review and accept the license agreement.**

   If you do not agree to the terms of the license agreement, click the radio button beside Decline License Agreement to cancel your download.

7. **Click the check boxes next to the files that you want to download.**

   You must download the README file. Select the `.zip` file of the product that you want to install. You can also download ISO images compressed in `.zip` files if you want to create physical media for the product.

8. **Click the Download selected with Sun Download Manager button to download the files.**

   The files are downloaded to your system.

9. **After you download the files, unzip the `.zip` files on your system.**

▼ SPARC: To Install the N1 Service Provisioning System 5.2 Update 2 Release

Before You Begin Make sure your system meets the hardware requirements set out in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide*.

To install the patches that comprise this update release of the N1 Service Provisioning System, you must already have installed and configured a compatible previous release or patch update of the Sun N1 Service Provisioning System. Compatible previous releases and patch updates are as follows:

- N1 Service Provisioning System 5.2
- N1 Service Provisioning System 5.2.1 Patch Update
- N1 Service Provisioning System 5.2.2 Patch Update
- N1 Service Provisioning System 5.2.3 Patch Update

For full details of how to install the N1 Service Provisioning System 5.2, see *Sun N1 Service Provisioning System 5.2 Installation Guide*. Details on how to install the patch updates can be found in the relevant patch documentation.

1 **If you do not already have a compatible version of the N1 Service Provisioning System installed and configured on your system, download and install the `n1_sps-5_2_1-ga-solaris-sparc.zip` file from the Sun's Software Portfolio download page.**

For full details of how to install the N1 Service Provisioning System 5.2.1, see *Sun N1 Service Provisioning System 5.2 Installation Guide*

Then, to bring N1 Service Provisioning System to the 5.2 Update 2 level, install the following patches:

- 122989-12 for the Master Server
- 122991-12 for the Command Line Interface

2 **After you apply the patches to the master server, update all the remote agents and local distributors in your environment.**

To auto-upgrade, log in to the patched master server. Go to the master server's host detail page and choose Upgrade Entire Host Network.

You can also execute a selective upgrade using the `node.*` commands. See Chapter 9, "node: CLI Commands for Performing Auto-upgrade Tasks," in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual* for more information.

This release of the N1 Service Provisioning System contains scripts that migrate your master server from compatible previous releases and patch updates to run the N1 Service Provisioning System 5.2 Update 2 Release.

Note – Only the Command Line Interface patch provided with this release can communicate as a command line interface with the master server patch provided with this release. If you upgrade your master server, you must also upgrade your Command Line Interface.

▼ x86: To Install the N1 Service Provisioning System 5.2 Update 2 Release

Before You Begin Make sure your system meets the hardware requirements set out in Chapter 2, "System Requirements for the Sun N1 Service Provisioning System 5.2," in *Sun N1 Service Provisioning System 5.2 Installation Guide*.

To install the patches that comprise this update release of the N1 Service Provisioning System, you must already have installed and configured a compatible previous release or patch update of the Sun N1 Service Provisioning System. Compatible previous releases and patch updates are as follows:

- N1 Service Provisioning System 5.2
- N1 Service Provisioning System 5.2.1 Patch Update
- N1 Service Provisioning System 5.2.2 Patch Update
- N1 Service Provisioning System 5.2.3 Patch Update

For full details of how to install the N1 Service Provisioning System 5.2, see *Sun N1 Service Provisioning System 5.2 Installation Guide*. Details on how to install the patch updates can be found in the relevant patch documentation.

1 **If you do not already have a compatible version of the N1 Service Provisioning System installed and configured on your system, download and install the `n1_sps-5_2_1-ga-solaris-x86.zip` file from the Sun's Software Portfolio download page.**

   For full details of how to install the N1 Service Provisioning System 5.2.1, see *Sun N1 Service Provisioning System 5.2 Installation Guide*

   Then, to bring N1 Service Provisioning System to the 5.2 Update 2 level, install the following patches:

   - 122990-12 for the Master Server
   - 122992-12 for the Command Line Interface

2 **After you apply the patches to the master server, update all the remote agents and local distributors in your environment.**

   To auto-upgrade, log in to the patched master server. Go to the master server's host detail page and choose Upgrade Entire Host Network.

   You can also execute a selective upgrade using the `node.*` commands. See Chapter 9, "node: CLI Commands for Performing Auto-upgrade Tasks," in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual* for more information.

   This release of the N1 Service Provisioning System contains scripts that migrate your master server from compatible previous releases and patch updates to run the N1 Service Provisioning System 5.2 Update 2 Release.

Note – Only the Command Line Interface patch provided with this release can communicate as a command line interface with the master server patch provided with this release. If you upgrade your master server, you must also upgrade your Command Line Interface.

Cohabitation of the N1 Service Provisioning System with the N1 System Manager

This section explains how to manage installation this version of the N1 Service Provisioning System on the same system as a compatible version of the N1 System Manager.

Solaris: Installing an N1 Service Provisioning System Master Server and an N1 System Manager Management Server on the Same System

If you use the default installation scripts for the N1 Service Provisioning System master server and the N1 System Manager management server, you cannot install both applications on the same system.

To install both applications on the same system, modify the N1 Service Provisioning System Master Server installation script to install the master server in an alternate root directory. To use this workaround, perform the following steps.

Note – Do not install the N1 Service Provisioning System OS Provisioning plug-in on a system that hosts both the N1 System Manager's management server and the N1 Service Provisioning System's master server.

---

1. Install the N1 System Manager management server. For more information, see *Sun N1 System Manager 1.3 Installation and Configuration Guide*.
2. Log in as root on the N1 System Manager management server system.
3. Copy the appropriate N1 Service Provisioning System master server installation script to the system.

   For an x86 installation, copy the `cr_ms_solaris_x86_pkg_5.2.sh` from the N1 Service Provisioning System distribution.

   For a SPARC installation, copy the `cr_ms_solaris_sparc_pkg_5.2.sh` from the N1 Service Provisioning System distribution.
4. If necessary, start a Korn shell session.

   ```
   # ksh
   ```
5. Create a shell script file that is named `aliases.sh` with the following contents.

   ```
   #!/bin/ksh
   alias -x pkgadd='pkgadd -R $NEW_PKG_ROOT'
   alias -x pkginfo='pkginfo -R $NEW_PKG_ROOT'
   alias -x pkgparam='pkgparam -R $NEW_PKG_ROOT
   ```
6. Create an alternate root directory for the N1 Service Provisioning System installation.

   ```
   # mkdir -p <alternate-root-path>
   ```
7. Export the value of the alternate root directory to use with the N1 Service Provisioning System installation script.

```
# export NEW_PKG_ROOT=alternate-root-path
```

In the previous step, *alternate-root-path* specifies the root directory in which you want to install the N1 Service Provisioning System Master Server, for example, `root1`. This alternate root directory will contain a new package repository for the N1 Service Provisioning System Master Server packages.

Note – The path for *NEW_PKG_ROOT* variable is critical. All steps must be executed in the shell where this new variable is exported.

---

8. Create a symbolic link from the opt directory to an alternate `opt` directory in the new alternate root directory.

   ```
   # ln -s /opt alternate-root-path/opt
   ```

9. Create a new installation script by prepending the `aliases.sh` script to the default N1 Service Provisioning System Master Server installation script. For example:

   ```
   # cat aliases.sh cr_ms_solaris_x86_pkg_5.2.sh > new_ms_installer.sh
   ```

10. Install the N1 Service Provisioning System Master Server by running the new installation script.

    ```
    # ./new_ms_installer.sh
    ```

11. Answer the installer questions.

    During the installation, consider the following limitations.

    - When you are prompted to specify the installation directory for the master server, enter a new subdirectory under `/opt`. To avoid overwriting the master server that is installed with N1 System Manager, ensure that this new subdirectory is different from the installation directory of the master server that is installed with N1 System Manager. For example, if, in step 8, you created a symbolic link from `/opt` to `/root1/opt`, specifying an installation directory of `/opt/ms1` during the installation installs the new N1 Service Provisioning System master server in `/root1/opt/ms1`.
    - When you are prompted to choose a port to use with the master server, ensure that you specify a port that is different from the port that is used by the master server used by N1 System Manager.

    For more information about how to install N1 Service Provisioning System, see *Sun N1 Service Provisioning System 5.2 Installation Guide*.

12. After the installation is completed, change to the directory that contains the N1 Service Provisioning System master server scripts. For example:

    ```
    # cd /root1/opt/ms1/N1_Service_Provisioning_System_5.2/server/bin
    ```

13. Create a backup copy of the N1 Service Provisioning System Master Server uninstallation script.

```
# cp cr_uninstall_ms.sh cr_uninstall_ms.sh.backup
```

14. Edit the `cr_uninstall_ms.sh.backup` script to perform the package operations on the alternate root. You can use the `sed` command to make these changes to the script by typing the following command.

```
# /usr/bin/sed -e 's!pkginfo!& -R '${NEW_PKG_ROOT}'!g' \
-e 's!pkgrm!& -R '${NEW_PKG_ROOT}'!g' \
-e 's!pkgparam!& -R '${NEW_PKG_ROOT}'!g' \
cr_uninstall_ms.sh.backup > cr_uninstall_ms.sh
```

15. Remove the backup copy of the uninstallation script.

```
# rm cr_uninstall_ms.sh.backup
```

To uninstall the N1 Service Provisioning System Master Server, use this revised version of the `cr_uninstall_ms.sh` script.

Note – Do not remove the new alternate root directory. This directory is required to uninstall the N1 Service Provisioning System master server.

◆ ◆ ◆ CHAPTER 2

Variable Substitution in Repository Component Targeters

This chapter describes new functionality for variable substitution in Repository Component Targeters.

Changes to Elements for Variable Substitution in Repository Component Targeters

This table updates the tables in "Using Substitution Variables" in *Sun N1 Service Provisioning System 5.2 Plan and Component Developer's Guide*:

Parent Element	Substitution Attribute
`<component>`	name, path, version, host
`<toplevelRef>`	name, host
`<nestedRef>`	name

If the name attribute of any of the above repository component targeters, or the name, path and/or version of the `<component>` contain variable substitution:

- The browser user interface does not display the component in the list of subplan parameters.
- You cannot specify the component version or variable settings for a component in the command line interface, through the `-comp` or `-vs` options, if the name, path, and/or version attributes of the `<component>` or the name attribute of the `<toplevelRef>`, or `<nestedRef>` is substituted by a variable.

Examples Illustrating Variable Substitution in Repository Component Targeters

This section provides simple examples to explain how the introduction of the new variable substitution for repository component targeters feature allows dynamic targeting in the N1 Service Provisioning System.

Look at the example plan below. The component name in the block named `secondInstall` is substituted by a variable, `varA`, where the default value is `comp2`.

```
<varList>
  <var name="varA" default="comp2"/>
</varList>
<simpleSteps>
  <install blockname="firstInstall">
    <component name="comp1" path="../apps" version="1.2"/>
  </install>
  <install blockname="secondInstall">
    <component name=":[varA]" path="../apps" version="1.3"/>
  </install>
</simpleSteps>
```

In this case, the browser user interface of the N1 Service Provisioning System does not show `comp2`, since `comp2` is substituted by a variable.

You can only specify the settings for the first component in this example, since it is not substituted by a variable. This plan could be run with the following command:

```
# pe.p.run -PID NM:/plan1 -tar H:NM:host1
-comp "1.1" -vs "varsetname" -pto 100 -nto 100
```

As explained in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*, the `-comp` argument for the `pe.p.run` command is used to specify component versions to install as a part of the plan and subplans. Versions are ordered according to component selectors within subplan prompts. Each referenced component must be represented in this list. Use the component version number, or "+" for default, "#" for recommended, and "-" for latest. Do not specify when there are no component versions.

With the inclusion of the new Dynamic Targeting feature in this version of the N1 Service Provisioning System, do not specify a component whose name, path, or version number contains substitution variables.

The `-vs` argument for the `pe.p.run` command is used to specify variable settings to use with each selected component version. Use variable settings name, or "+" for default. Do not specify when there are no variable settings.

With the `-vs` argument, as with the `-comp` argument, do not specify a component whose name, path, or version number contains substitution variables.

As with the browser user interface, any components that have been substituted by variables are not provided in the output of the `pe.p.lp` command.

You can therefore also use the output of the `pe.p.lp` command to list those components for which you can specify settings when using the `pe.p.run` command.

See the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual* for explanations of the other terms used in this command.

This table updates the `pe.p.run` table in "Overview of the pe Commands" in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*:

Argument	Result	Syntax	Description
`comp`	O/R	StringArrayArray	Component versions to install as a part of the plan and subplans. Versions are ordered according to component selectors within subplan prompts. Each referenced component must be represented in this list. Use the component version number, or "+" for default, "#" for recommended, and "-" for latest. Do not specify when there are no component versions. Do not specify when component name is substituted by a variable
`vs`	O/R	StringArrayArray	Variable settings to use with each selected component version. Use variable settings name, or "+" for default. Do not specify when there are no variable settings. Do not specify when component name is substituted by a variable.

Note – Do not specify settings for components that are dynamically targeted.

Another example that addresses a slightly different situation is shown below:

```
<varList>
  <var name="varA" default="comp2"/>
</varList>
<simpleSteps>
  <install blockname="firstInstall">
    <component name="comp1" path="../apps" version="1.2"/>
  </install>
  <install blockname="secondInstall">
```

```
    <component name=":[varA]" path="../apps" version="1.3"/>
  </install>
  <install blockname="thirdInstall">
    <component name="comp3" path="../apps" version="1.3"/>
  </install>
</simpleSteps>
```

Suppose this plan were run with the following command:

```
# pe.p.run -PID NM:myplan -tar <target host list>
 -comp "2.0,3.0" -vs "VS1,VS3" -pto 100 -nto 100
```

Suppose that `VS1` and `VS2` both are valid names of existing component variable settings.

In this example, although there are three components being installed, only two values for the `-comp` parameters and for the `-vs` parameters have been specified with the `pe.p.run` command. The result of this command would be as follows:

- Version 2.0 of component `comp1` would be installed with variable setting `VS1`.
- Version 1.3 of component `comp2` would be installed with the default variable setting for `comp2`.
- Finally, version 3.0 of component `comp3` would be installed with variable setting `VS3`.

The targeter component name that is substituted by a variable is skipped because specifying settings for dynamically targeted components is not supported at the command line or in the browser user interface.

◆ ◆ ◆ CHAPTER 3

Install Step Component Variables

This chapter describes a new feature for this release of the N1 Service Provisioning System, which allows you to supply component variable values in the `<install>` step of a plan.

Understanding Install Step Component Variables

To specify component variable values for top-level components in a plan, it is no longer necessary to select component variable sets only in the Plan Run screen of the browser user interface, or to specify existing component variable sets only at plan run time using the command line interface. In this release of the N1 Service Provisioning System, you can supply component variable values in the `<install>` step of a plan.

As plan author, you no longer have to wait until the plan is accessible in the browser user interface or the command line interface. To be able to supply component variable values in the `<install>` step of a plan, two new elements have been provided.

The two optional new child elements of the `<install>` step are:

- `<compVarSet>`
- `<compVarList>`

Both elements can be used for the same target component. The `<compVarSet>` element enables you specify an existing named variable setting for the target component. The `<compVarList>` element enables you to override one or more specific variables for the target component. If the same variable is used in both elements, the values specified in `<compVarList>` override those of `<compVarSet>`. The `<install>` step is described in the *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

The <compVarSet> Element

The <compVarSet> element allows you to supply the name of an existing component variable setting in the <install> step of a plan, when a <component> child element of the same step exists.

The <compVarSet> element is a child of the <install> element and is valid only if the <install> element uses a <component> repository targeter.

The <compVarSet> element appears just after the <component> element. For information about <component> repository targeters, see "Repository Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

The <compVarSet> element has one attribute: name. The name attribute for the <compVarSet> element supports variable substitution and must be an existing component variable setting name in the target component.

If a <compVarSet> element is specified for a targeter component in an install step, you cannot override the component's name either in the Plan Run screen of the browser interface, or when running the plan at the command line, with the pe.p.run command and the -vs argument.

The <compVarList> Element

The new <compVarList> element allows you to supply the values of component variable settings during the <install> step of a plan. The <compVarList> element is a child of the <install> element and is valid only if the <install> element uses a <component> repository targeter. For information about <component> repository targeters, see "Repository Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

The purpose of the <compVarList> element is to specify a list of values to be used as overriding component variable settings for the component target when it is installed.

If a <compVarList> element is specified for a targeter component in an install step, you cannot override the component's version either in the Plan Run screen of the browser interface, or when running the plan at the command line, with the pe.p.run command and the -vs argument. This restriction applies in exactly the same way as with dynamic targeting. See "Changes to Elements for Variable Substitution in Repository Component Targeters" on page 17 for more information on dynamic targeting.

Each attribute of the <compVarList> element names a variable in the resolved target component. The value of each of these attributes takes precedence when the resolved component is installed. If any of these attributes does not correspond to a variable that has been declared in the target component variable settings, the plan check-in does not succeed.

Any component variables that are not named in the <compVarList> element are given their default value when installed, and if the <compVarList> element is not used, all component variables are given their default values when installed.

All component variables named in the <compVarList> element must be accessible to the <install> step of the plan, and must be declared with access mode PUBLIC or PROTECTED.

Component variables named in the <compVarList> element cannot be declared with access mode FINAL or PRIVATE. If component variables named in the <compVarList> element are declared with access mode FINAL or PRIVATE, an error occurs during plan check-in.

Here is an example of the <compVarList> element, with a single variable, and variable substitution:

```
<install blockName='default'>
   <component name='foo' path='/tmp'/>
   <compVarList comp1var1='val1' comp1var2=':[localVar]'/>
</install>
```

◆ ◆ ◆ CHAPTER 4

Unrestrictive Host Locking

This release of the N1 Service Provisioning System provides new functionality in the area of host locking. In previous releases, the N1 Service Provisioning System locked a host while executing steps on that host. To prevent plans interfering with each other, no other plans or subplans could be executed against a host while the host was locked by a plan. Now, you can elect to use less restrictive host locking.

The `implicitLocking` Attribute

The `<simpleSteps>` element has been modified to accept a new attribute, called `implicitLocking`. Attributes for the `<simpleSteps>` element are now as follows:

`executionMode` An optional attribute already part of the N1 Service Provisioning System, that indicates whether the contained steps should be executed in a series or in parallel over the target hosts. The legal values are `PARALLEL` and `SERIES`. If the `executionMode` attribute is omitted, the value is `PARALLEL`.

`limitToHostSet` An optional attribute already part of the N1 Service Provisioning System, that specifies the name of the host set that contains the hosts that can be valid targets for this plan. If this attribute is omitted, all hosts can be valid targets. Otherwise, the targets that you specify must be a subset of the hosts that are contained in the named host set.

If the targets include a host that is not contained in the named host set, the plan issues a runtime error. If you specify a name that does not correspond to an existing supported host set, the plan also issues a runtime error. If the specified host set is defined by a plug-in, the `pluginName` must be prefixed to the host set name, such as `pluginName#hostSetName`. These plan runtime errors are reported during validation before the preflight starts.

`implicitLocking` A new attribute for the release of N1 Service Provisioning System. The `implicitLocking` attribute indicates what type of host locking to use for the contained steps. If the `implicitLocking` attribute is omitted, the value is set to `TRUE`, so no other plans or subplans can be executed against that host until the steps enclosed in the `<simpleSteps>` element have completed. If a plan is retargeted to another host, this retargeted host is also locked until the completion of the retargeted step.

If the `implicitLocking` attribute is set to false, the browser interface displays the following new message for a retarget step:

```
connect and check hosts
```

If the `implicitLocking` attribute is set to `FALSE`, no hosts are locked by a plan during the completion of the `<simpleSteps>` element, including retargeted hosts, unless a `lockHost` step is used.

The `lockHost` Step

The `lockHost` step indicates that the target host should be locked for the duration of the step. The `lockHost` step can be a child of the following steps:

- The `<call>` step
- The `<retarget>` step
- The `<install>` step
- The `<uninstall>` step
- The `<catch>` step
- The `<finally>` step
- The `<then>` step
- The `<else>` step
- The `<block>` step of the `<try>` step

The `lockHost` step has one child element. See "Attributes and Child Elements of the `lockHost` Step" on page 27 for more information on the child element of the `lockHost` step.

If the `lockHost` steps contain a retargeted step, the host to which the step is retargeted is also locked. Execution is suspended until the required lock is obtained. The lock is released at the conclusion of the `lockHost` step regardless of whether the steps succeed or fail.

If the `lockHost` steps do not contain a retargeted step, the inner `lockHost` steps would lock the host on which the step is executed. A nested `lockHost` step does not block waiting for a parent `lockHost` step issued as part of the same plan.

Attributes and Child Elements of the lockHost Step

The lockHost step has the following optional attribute:

lockHost Step Attribute Name	Type	Description
lockingMode	one of: VIRTUAL, PHYSICAL or BOTH	Indicates type of locking method to use. VIRTUAL locking locks only the virtual host. PHYSICAL locking locks only the physical host. BOTH will lock both the physical host and the virtual host. If the target type of the host on which the plan is being executed is physical, no virtual hosts are locked. BOTH is the default. If omitted, BOTH is assumed.

Note – It is not possible to lock a physical host and all its virtual hosts with a single lockHost step.

The lockHost step is used to lock a target host for the duration of a step when the implicitLocking attribute of the <simpleSteps> element is set to FALSE. The lockHost step can also be used when the implicitLocking attribute of the <simpleSteps> element is set to TRUE.

Note – If a plan is executed on a virtual host and the implicitLocking attribute of the <simpleSteps> element is set to TRUE, the virtual host is locked. If the plan includes a lockHost step with the lockingMode attribute set to PHYSICAL, the physical host is also locked, but the virtual host is not unlocked. The virtual host is not unlocked because implicitLocking attribute of the <simpleSteps> element is still set to TRUE.

The host locking in this case is the union of both the implicitLocking attribute of the <simpleSteps> element and the lockHost step.

<steps> Element

The lockHost step has one child element: steps. The steps element can contain zero or more steps, and indicates which steps to execute while the host is locked. Any shared step or simple plan step is permitted.

The lockHost step and its child steps are visible in the Plan Run History section of the browser user interface.

Elements Modified By Introduction of the lockHost Step

With the introduction of the lockHost step, several elements described in the *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* gain a new attribute: the requireLocking attribute. The requireLocking attribute is designed to safeguard a component against use in an unlocked environment.

Existing Element (Child of)	New Attribute	Description
■ installSteps (installList) ■ uninstallSteps (uninstallList) ■ control (controlList)	requireLocking	An optional attribute indicating if the component install, uninstall, orcontrol block can be used in an unlocked plan. The default value is TRUE. Plans using use this component block must lock the host for the duration of the component installation, uninstallation, control or snapshot operations. Locking can be achieved by implicit plan locking or by explicit locking with the lockHost step.

Examples of the lockHost Step

EXAMPLE 4–1 Nested lockHost Steps

It is possible to specify nested lockHost steps. You can also specify a lockingMode attribute for each nested lockHost step:

```
1st group of steps
    <lockHost lockingMode="PHYSICAL">
      2nd nested group of steps
      <lockHost lockingMode="VIRTUAL">
        3rd nested group of steps
      </lockHost>
      4th nested group of steps
    </lockHost>
    5th nested group of steps
```

EXAMPLE 4–1 Nested lockHost Steps *(Continued)*

In the above example, no hosts are locked for the 1st group of steps, since there are no locks: the lockHost step has not been used.

Only the physical host is locked for 2nd group of steps, because the lockHost step has been used with the lockingMode attribute set to PHYSICAL.

Both the physical and the virtual host are locked for the 3rd group of steps, because a lockHost step has been used with the lockingMode attribute set to VIRTUAL and the physical lock has not yet been closed.

Only the physical host is locked for 4th group of steps, because the virtual lock has been closed. No hosts are locked for the 5th group of steps, because the physical lock has now been closed.

EXAMPLE 4–2 Simple Usage of the lockHost Step

Here is a simple example of the lockHost step:

```
<simpleSteps implicitLocking="false">
  <pause delaySecs="1"/>
  <lockHost>
    <pause delaySecs="1"/>
  </lockHost>
</simpleSteps>
```

Note that implicitLocking is set to false. To lock the host in the above simple example, the lockHost step would therefore be required. Since the lockingMode attribute is omitted, both the physical host and any virtual host are locked, if the plan is executed on a virtual host. If the plan is executed on a physical host, the physical host only would be locked as there is no virtual host to lock.

EXAMPLE 4–3 The lockHost Step with a Retargeted Step

Here is a more complex example of the lockHost step including a retargeted step:

```
<simpleSteps implicitLocking="false">
  <pause delaySecs="1"/>
  <lockHost lockingMode="VIRTUAL">
    <pause delaySecs="1"/>
    <retarget host="newhost">
      <pause delaySecs="2"/>
    </retarget>
  </lockHost>
 <pause delaySecs="3"/>
</simpleSteps>
```

EXAMPLE 4–3 The lockHost Step with a Retargeted Step *(Continued)*

In this example:

- implicitLocking is set to false, so if the plan is executed on host1, neither newhost nor host1 are locked initially.
- When the lockHost step is executed, host1 is locked. The first pause step is executed before host1 is locked.
- If host1 is already locked, the plan does not act on host1 until host1 becomes available.
- newhost is locked when the retarget step is executed. The lock on host1 is not released, so both hosts are now locked. If locking newhost would result in a deadlock, then an error is thrown. Depending in the configuration, the error message is either Deadlock detected when initial host *X* tried locking host *Y*. That lock is already held by host *Z* or Locking the host results in a deadlock
- The lock on newhost is released on completion of the retarget step.
- The lock on host1 is released on completion of the lockHost step.
- The 1 second and 3 second pause steps are executed on an unlocked host; the 2 second pause step is executed on a locked host.
- The lockingMode is set to VIRTUAL, so only the virtual host to which the retarget step is aimed is locked.

Contrast the above example with this:

```
<simpleSteps implicitLocking="false">
   <pause delaySecs="1"/>
     <pause delaySecs="1"/>
     <retarget host="newhost">
       <lockHost>
         <pause delaySecs="2"/>
       </lockHost>
     </retarget>
   <pause delaySecs="3"/>
 </simpleSteps>
```

Here, only newhost is locked, and only on the execution of the lockHost step inside the retarget step. The retargeted host is not locked when the retarget step is executed. Only the 2 second pause step is executed on a locked host.

EXAMPLE 4–4 The Interaction of the requireLocking Attribute and the lockHost Step

For clarity, several mandatory component attributes are omitted from this example.

```
<component name="comp1"...>
 <installList>
```

EXAMPLE 4–4 The Interaction of the requireLocking Attribute and the lockHost Step *(Continued)*

```
    <installSteps name="installA">
      <pause delaySecs="1">
    </installSteps>
  </installList>
</component>
```

The installA installSteps element does not specify the requireLocking attribute. The requireLocking attribute is therefore set to true, by default. Executing the following plan would therefore cause a preflight error:

```
  <simpleSteps implicitLocking="false">
    <install blockName="installA">
      <component path="/" name="comp1"/>
    </install>
  </simpleSteps>
```

The plan indicates that locking should be disabled by specifying the implicitLocking attribute with a value of false. The plan attempts to operate on a component using a block that requires locking without locking the host. The install block executed by the plan does not specify the lockingMode attribute, which means that the block requires the host to be locked upon execution. The plan could be changed to explicitly lock the host before installing the component:

```
 <simpleSteps implicitLocking="false">
   <lockHost>
     <install blockName="installA">
       <component path="/" name="comp1"/>
     </install>
   </lockHost>
 </simpleSteps>
```

Here is an simplified sample of a component that does not always require the host to be locked:

```
<component name="comp2"...>
  <installList>
    <installSteps name="installA" requireLocking="false">
      <pause delaySecs="1">
   </installSteps>
    <uninstallSteps name="uninstallB" requireLocking="true">
      <pause delaySecs="1">
    </uninstallSteps>
    <control name="controlC">
      <pause delaySecs="1">
    </control>
  </installList>
</component>
```

EXAMPLE 4–4 The Interaction of the requireLocking Attribute and the lockHost Step *(Continued)*

In the above simplified sample:

- The install block, installA, does not require the host to be locked
- uninstallB and controlC both require the host to be locked. uninstallB states the explicit requirement while controlC does not, but the default value if unspecified is true.

The following plan is provides an example of legal operation with this component:

```
<simpleSteps implicitLocking="false">
  <install blockName="installA">
    <component path="/" name="comp1"/>
  </install>
  <lockHost>
    <call blockName="controlC">
      <installedComponent path="/" name="comp1" =installPath="/tmp"/>
    </call>
    <uninstall blockName="uninstallB">
      <installedComponent name="comp1"/>
    </uninstall>
  </lockHost>
</simpleSteps>
```

A component block that does not require the host to be locked can be called while the host is locked; the install step above could legally be moved inside the lockHost step. Neither the uninstall nor the call, however, could be moved outside the lockHost step.

◆ ◆ ◆ CHAPTER 5

Copying Objects Between Master Servers

In this release of the N1 Service Provisioning System, a new Import-Export feature has been provided to enable you to easily copy various Service Provisioning System objects at once from your development master server to your testing server, to your staging master server, and from there to your production master server. Service Provisioning System objects can include:

- Component Types
- System Services
- Host Types
- Host Sets
- Host Searches
- Folders
- Components
- Plans

Overview of the Import-Export Feature

Service Provisioning System objects such as components, plans and host sets can now easily be copied from one master server into a single file and on to another master server. This is done as follows:

1. On the source master server, create a *bundle template*.

   A bundle template contains an ordered list of search criteria that add your selection of service provisioning system objects such as the following:

   - Component Types
   - System Services
   - Host Types
   - Host Sets
   - Host Searches
   - Folders
   - Components

- Plans

The ordering of these objects is important, because you export these objects in to the bundle template, using search criteria, in the order of the objects' mutual dependencies: matched objects are exported into the bundle template in the order of the search criteria that they match. See "Declaring Search Criteria for a Bundle Template" on page 35 for details. If the objects are not exported in the correct order, the export succeeds but the subsequent import operation can fail.

2. When you have created and saved the bundle template, export the bundle template in to a *bundle jar*. See "Exporting a Bundle Template to a Bundle Jar" on page 41 for details.

   You can manually create a bundle jar, or modify it after it has been automatically created by the system. This is on the condition that the jar contents are always consistent with the bundle descriptor file. See "Contents of a Bundle Jar" on page 43 for details.

3. On the destination master server, test to see if the import of the bundle jar would work, using the validate feature. See Example 5–8 for details.
4. If the validation succeeds, import the bundle jar. See Example 5–9 for details.

Note – All users can use the import-export feature.

Creating a Bundle Template

Create a bundle template using the `bdb.b.add` command. Use the following arguments with the `bdb.b.add` command:

`name`	The name of the bundle template
`desc`	A description of the bundle template
`criteria`	Search criteria for entities to be included in the bundle template.

The result of the `bdb.b.add` command is the generation of a unique *bundle ID*.

For full usage of this command, see Appendix A.

Note – The bundle ID can be a cumbersome ID to enter at the command-line. Consider substituting the ID for a shorter term using the `NM` command as described in "Using NM: to Perform ID Substitution" in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

Note – A bundle template cannot contain another bundle template.

Declaring Search Criteria for a Bundle Template

When you create the bundle template, specify search criteria for plans, components or other objects in the correct order of their dependencies.

To specify search criteria for the bundle template, use the `bdb.b.add` command.

The order in which you declare search criteria using the `bdb.b.add` command is important. Matched objects are exported into the bundle jar in the order of the search criteria that they match. The search criteria must be in the correct order of the matched object's dependencies.

Note – The ordering of `systemService` and `componentType` search criteria is not significant. The component to which they refer must be included in the same bundle template.

Search criteria can be specified as set out in the following table.

TABLE 5–1 Search Criteria and How They Can Be Specified

Search criteria	Declared using the `bdb.b.add -criteria` command as	Name	Description	Visibility	Version	Folder path	Type	Label
`SystemService`	SS	X	X					
`HostType`	HT	X	X	X				
`HostSet`	HS	X	X	X				
`HostSearch`	HR	X	X	X				
`ComponentType`	CT	X	X					
`Plan`	P	X	X	X	X	X		
`Folder`	F	X	X	X		X		
`Component`	C	X	X	X	X	X	X	X

For example, it is possible to specify the `name` and `description` search criteria for system services.

If you specify a search criterion for an entity and the entity does not exist, the creation of the bundle template is successful but the export to a bundle jar does not succeed.

Bundle templates can have zero or more search criteria.

Criteria are declared as attribute=value pairs, separated by a comma. Multiple similar criteria are separated by a semicolon. Pattern strings that contain a comma, semicolon or an equals character in them can be included by the use of '\' (backslash) as an escape character. Backslash itself could be specified as '\\' in the pattern. Attributes are declared as follows:

`name` The name of the entity or object to be searched for.

`desc` The description of the entity or object to be searched for. This corresponds to the description property of the entity searched for.

`vis` Visibility of the entity or object to be searched for.

`ver` The version of the entity or object to be searched for.

`ct` `extendsType`. Refers to the `componentType` the sought component is extending. This helps search components that extend a particular Component Type. This attribute is only applicable to the component search.

`lab` The label of the entity or object to be searched for.

Search Criteria Validation

Search criteria are validated internally using the following character limits.

- Name: 512 characters
- Description: 1024 characters
- Folder path: 512 characters
- Type: 129 characters
- Label: 32 characters
- Visibility: this can be either hidden or, by default, visible.

EXAMPLE 5–1 Creating a Bundle Template

This example shows a bundle template being created with the `bdb.b.add` command. Note that the search criteria are separated by semicolons, and attribute value pairs are separated by = as described in "Declaring Search Criteria for a Bundle Template" on page 35.

```
bdb.b.add -name b1 -desc d1 -criteria "C:name=comp*;P:name=myPlan;F:name=myFolder;"
```

The result of the command is that the bundle template is created and the associated bundle ID is returned.

```
129158239041-1163688372443-00830-2096382958
```

For full usage of this command, see Appendix A.

Viewing and Listing Bundle Templates

This section explains how to view a bundle template and how to list all bundle templates on the master server.

Listing All Bundle Templates

To list all bundle templates, use the `bdb.b.la` command.

The output of the `bdb.b.la` commands is as follows:

BundleID	The ID of the bundle template
Bundle Name	The name of the bundle template
Bundle Description	The description of the bundle template.
Last Updated	The date, time, and time zone information for when the bundle template was last updated.

For full usage of this command, see Appendix A.

EXAMPLE 5–2 Output of the `bdb.b.la` Command

This example shows a sample output returned after issuing the `bdb.b.la` command, which lists all bundle templates on the master server.

```
|------------------|----------|------------|-----------------------------------|
|BundleID |Name |Description |LastUpdated |
|-----------------------------------------|----------|------------|---------------------------|
|129158239109-1160744183744-00241-1568077714|bplan4 |bundes1 |Fri Oct 13 18:26:24 IST 2006|
|129158239109-1160744183744-00241-1568077712|bplan3 |bundes3 |Fri Oct 13 12:26:24 IST 2006|
|-----------------------------------------|----------|------------|-------------------------|
```

The output shows that there are two bundle templates. The first string of numbers is the bundle ID. Looking at the first of the two bundles, its ID is `129158239109-1160744183744-00241-1568077714` and its bundle name is `bplan4`. Its bundle description is `bundles1`.

Viewing a Single Bundle Template

To display the details of a single bundle template, including the defined search criteria and the corresponding elements or objects, use the `bdb.b.lo` command with the `-ID` argument, where ID is the ID of the bundle template. You can substitute the ID as explained in "Creating a Bundle Template" on page 34.

The default output of the `bdb.b.lo` command is the same as that of the `bdb.b.la` command.

In addition, with the `bdb.b.lo` you can use the `-o` option to obtain a more detailed summary. The following information is displayed in tabular form:

BundleID	The ID of the bundle template
Bundle Name	The name of the bundle template
Bundle Description	The description of the bundle template.
Last Updated	The date, time, and time zone information for when the bundle template was last updated.
Criteria Pattern Details	The search criteria of the bundle template.
Entities Matched	The `objectType`, `ID`, `name` and `version` of entities that matched the search criteria of the bundle template.

For full usage of this command, see Appendix A.

EXAMPLE 5–3 Output of the `bdb.b.lo` Command

This example shows a sample output returned after issuing the `bdb.b.lo` command to list the details of a specified single bundle template. In this example, the command is issued with the `-o` option, and the `NM` command has been used to replace the bundle ID with a shorter term.

`bdb.b.lo -ID NM:`*`b3`* **`-o detail`**

The result is as follows.

```
BundleID : 129158239041-1162211396014-00217-0193112936
Name : b3
Description : bundes3
LastUpdated : Mon Oct 30 17:59:56 IST 2006
Criteria Pattern Details :
|----------|------|----------|------------|-------|----------|------------|-----------|
|CriteriaType|Select ALL|Name |Description |Version|Label |FolderPath|ExtendsTypeName|
|----------|------|----------|------------|-------|----------|------------|-----------|
|HostType |False |bundleHost2|descBundle2 | | | | |
|HostType |False |bundleHost1|descBundle1 | | | | |
|----------|------|----------|------------|-------|----------|------------|-----------|
Entities Matched :
|---------------|---------------|----------|-------|
|ObjectType |ObjectID |Name |Version|
|---------------|---------------|----------|-------|
|HostType |129158239041-1162211393799-00201-0215113755|bundleHost1-2-1162211393664| |
|HostType |129158239041-1162211393838-00205-0238623680|bundleHost2-3-1162211393830| |
|---------------|---------------|----------|-------|
```

EXAMPLE 5–3 Output of the `bdb.b.lo` Command *(Continued)*

The `bdb.b.lo` command can be used without the `-o` option, to get a summary of the details of a single bundle template.

`bdb.b.lo -ID NM`:*b3*

The result is as follows.

```
BundleID : 129158239041-1162211396014-00217-0193112936
Name : b3
Description : bundes3
LastUpdated : Mon Oct 30 17:59:56 PST 2006
```

Modifying a Bundle Template

Once you have created a bundle template, you can edit its name, description and search criteria. Use the `bdb.b.mod` command to modify the bundle template. Use the following arguments with the `bdb.b.mod` command:

`ID`	The ID of the bundle template.
`desc`	The new description of the bundle template.
`criteria`	Search criteria for entities to be included in the bundle template. Criteria are declared as indicated in "Declaring Search Criteria for a Bundle Template" on page 35.

Note – The `bdb.b.mod` command does not change the ID of the bundle template.

Once you are satisfied with the bundle template, save it and export it in the form of a bundle jar.

The `bdb.b.mod` command overwrites existing search criteria.

To delete a single search criteria, all existing criteria except the one to be deleted, need to be provided to the `bdb.b.mod` command.

Note – When you use the `bdb.b.mod` command with the `-criteria` option, all existing are overwritten, even if you do not declare them with the `bdb.b.mod` command. Criteria that you do not declare are removed.

For full usage of this command, see Appendix A.

EXAMPLE 5–4 Using the `bdb.b.mod` Command to Modify a Bundle Template By Overwriting Existing Criteria

This example shows how to use the `bdb.b.mod` command to modify a bundle template, overwriting all existing search criteria.

```
bdb.b.mod -ID 129158239041-00830 -criteria
"CT:name=myCompType;HS:name=myHostSet;HR:name=myHostSearch;HT:name=myHostType;"
```

The output of the command is the bundle template ID.

```
129158239041-00830
```

The ID remains that same as it was before the bundle template search criteria were modified.

EXAMPLE 5–5 Using the `bdb.b.mod` Command to Modify a Bundle Template and Remove Existing Criteria

This example shows how to use the `bdb.b.mod` command to modify a bundle template, removing all existing search criteria.

```
bdb.b.mod -ID NM:b1 -desc "modified" -criteria "empty"
```

The following command also works.

```
bdb.b.mod -ID NM:b1 -desc "modified" -criteria ""
```

The output of the command is the bundle template ID.

```
129158239041-1163688372443-00830-2096382958
```

The ID remains that same as it was before the bundle template's search criteria were removed.

Deleting a Bundle Template

Delete a bundle template using the `bdb.b.del` command.

The `bdb.b.del` command takes the ID of the bundle template as an argument. You can delete a bundle template without deleting the entities that it references. This is because the entity search criteria, and not the entities themselves, are saved in the bundle template.

EXAMPLE 5–6 Using the `bdb.b.del` Command to Delete a Bundle Template

This example shows how to use the `bdb.b.del` command to delete a bundle template.

```
bdb.b.del -ID NM:myBundle
```

For full usage of this command, see Appendix A.

Exporting a Bundle Template to a Bundle Jar

When you have created the bundle template, you are ready to export it to a bundle jar.

Specify the file system path in which to save the bundle jar. The maximum size of a bundle jar is 2GB.

Use the `bdb.b.exp` command to export a bundle template to a bundle jar: This command takes the ID of the bundle template as an argument. The path argument is also used, to specify the path to the file system in which to save the bundle jar.

For full usage of this command, see Appendix A.

During the export of the bundle template, if the resultant bundle jar would exceed 2GB, the export operation fails.

If more than one search criteria match the same object, only the first criteria is considered.

Only a saved bundle template can be used to export a bundle jar.

Bundle template search criteria must refer to a single version of objects that can have versions. Export of multiple versions of the same object is not supported.

The default value for the visibility filter is `visible` therefore, unless specified otherwise in the command criteria, only visible entities are exported.

Do not export directories that have symbolic links within them. Export of a directory containing a symbolic link fails.

On export, the resource version number is set to 1.0, if it is already not 1.0 in the component XML. If the resulting bundle jar was imported to a master server that did not already have this resource checked in, import would have failed. To avoid this, the version is set to 1.0 implicitly.

The export operation fetches the bundle template immediately, hence any subsequent edit or delete operations on the bundle template do not impact the export operation.

EXAMPLE 5–7 Using the `bdb.b.exp` Command to Export a Bundle Template

This example shows how to use the `bdb.b.exp` command to export a bundle template to a Bundle Jar. The jar path is input as `/aa/aa/abc.jar`

```
bdb.b.exp -ID NM:myBundle -path "/aa/aa/abc.jar"
```

The result is as follows.

EXAMPLE 5–7 Using the `bdb.b.exp` Command to Export a Bundle Template *(Continued)*

```
Processed:Folder=1, Plan=1
```

The output shows that the operation was successful.

Importing a Bundle Jar

A bundle jar can be imported by selecting it from the local filesystem. If the operation is successful, its entities are added to the destination master server in the order specified in the bundle descriptor file.

Ensure that all entities satisfy the usual requirements of the provisioning system, and that any plugins on which they depend are already imported.

Use the `bdb.b.imp` command to import a bundle jar from the local file system. This command takes the following arguments:

- `path`: The path to the file system from which to import the bundle jar.
- `owner`: The group destined to be owner of folders created through the import operation. Use this to specify the owner group of all folders to be created by the bundle jar. When a new leaf or interior folder is created, ownership is by the same group.
- `v`: Default value is false. Set this to true if you want to validate the import, but do not want to actually execute the import operation. Setting the value to true simulates the import, and returns any possible errors. If no errors would occur, this returns the number of entities that would be imported. Setting this value to true means that the actual import operation does not take place.

For full usage of this command, see Appendix A.

On each master server, you can only import one bundle jar at any time.

After the import operation, the newly imported objects are visible, but are not categorized. Any previous versions of these objects are implicitly hidden.

Entities added or updated by the import operation have no record that they were checked in by a bundle jar.

If plans or components owned by a plugin and that depend on elements in the plugin, are included in a bundle jar, the plans or components might not function after the import operation, unless all elements on which they depend are also imported. By default, therefore, search criteria in bundle templates exclude entities that are contained in plugins.

Folder paths for plans or components in the bundle jar cannot be owned by a plugin that is already imported on the master server. Best practice is not to include versions when referring to subplans, subcomponents and targeters, because subplans, subcomponents and targeters may not work reliably after import in different environments.

Only one import operation is permitted for a master server at any time.

EXAMPLE 5–8 Using the `bdb.b.imp` Command to Validate the Import of a Bundle Jar

This example shows how to use the `bdb.b.imp` command with the `-v` option to test if the import of a bundle jar would be successful. The jar path is input as `/bb/bb/bundle.jar`

```
bdb.b.imp -v true -owner NM:myGroup -path "/bb/bb/bundle.jar"
```

The result is as follows.

```
Processed:Folder=1, Plan=1
```

The output shows that the operation would be successful.

EXAMPLE 5–9 Using the `bdb.b.imp` Command to Import a Bundle Jar

This example shows how to use the `bdb.b.imp` command to import a bundle jar. The jar path is input as `/bb/bb/bundle.jar`

```
bdb.b.imp -v false -owner NM:myGroup -path "/bb/bb/bundle.jar"
```

The result is as follows.

```
Processed:Folder=1, Plan=1
```

The output shows that the import operation was successful.

Contents of a Bundle Jar

The contents of a bundle jar are described in an XML descriptor file, located in the top level directory of the bundle jar file. The rest of the jar file is composed of XML representations of the entities that were the result of bundle template's search criteria.

The XML descriptor file is enclosed in the `<bundle>` element. The `<bundle>` element is a new element for this release, and is not described in the XML Schema Guide.

You can manually create a bundle jar, or modify it after it has been automatically created by the system. This is on the condition that the jar contents are always consistent with the bundle descriptor file.

Attributes of the <bundle> Element

The <bundle> element has the following attributes.

TABLE 5–2 Bundle Element Attributes

Bundle Element Attribute Name	Type	Required	Description
xmlns	String	yes	Required value: the first component of the schemaLocation attribute is http://www.sun.com/schema/SPS
xmlns:xsi	String	yes	Required value: http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation	String	no	Recommended value: http://www.sun.com/schema/SPS bundle.xsd
name	String	yes	The name of the bundle
description	String	no	The description of the bundle. Default is no description
exportedAt	String	yes	The date, time, time zone information on when the bundle was exported.
source	String	yes	IP address/Port number of the master server from which the bundle was exported.
userName	String	yes	Name of the user who exported the bundle.
schemaVersion	schemaVersion	yes	The version of the plugin XML schema being used.

Child Elements of the <bundle> element

The <bundle> element has the following child element.

`memberList`: List of member objects to create as part of the bundle. These member objects can include 1 or more objects of:

- `folder`
- `hostType`
- `hostSet`
- `hostSearch`
- `component`
- `plan`

Child Elements of the `<memberList>` Element

The `<memberList>` element has the following child elements:

- `<folder>`
- `<hostType>`
- `<hostSet>`
- `<hostSearch>`
- `<component>`
- `<plan>`

`<folder>` Element

The `<folder>` element is a child of the `<memberList>`element. It is used to declare a folder to be referenced by the bundle. The owner of these folders is the group you choose when importing.

Attributes of the `<folder>` Element

The `<folder>` element has two attributes:

- *name* – The name of the folder to be referenced by the bundle jar.
- *description* – The optional description of the folder. Default is none.

`<hostType>` Element

The `<hostType>` element is a child of the `<memberList>`element. It is used to declare a hostType to be referenced by the bundle.

Attributes of the `<hostType>` Element

The `<hostType>` element has two attributes:

- *name* – The name of the hostType to be referenced by the bundle jar.
- *description* – The optional description of the hostType. Default is no description.

<varList> Element

The <hostType> element contains an optional <varList> child element. The <varList> element specifies a list of variables to be added to the <hostType> element and later used by hosts in configuration.

The <varlist> element contains one or more <var> child elements. The <var> element provides <hostType> element variable declaration through two required attributes:

- *name* – The name of the variable
- *default* – The default value of the variable

<hostSet> Element

The <hostSet> element is a child of the <memberList> element and is used to declare a host set to be referenced by the bundle. The <hostSet> element cannot contain hosts, since bundles cannot define hosts.

The <hostSet> element contains two optional child elements:

- <hostSetRef>
- <hostSearchRef>

Attributes for the <hostSet> Element

The <hostSet> element has two attributes:

- *name* – The name of the host set.
- *description* – An optional description of the host set

<hostSetRef> Element

The <hostSetRef> element is a child of the <hostSet> element. It specifies a sub-host set. This host set must have been previously defined either in this bundle or in the master server on which the bundle is being imported.

Attributes for the <hostSetRef> Element

The <hostSetRef> element has one attribute, *name*. This attribute provides the name of the host set reference.

<hostSearchRef> Element

The <hostSearchRef> element is a child of the <hostSet> element. It specifies a sub-host search. This host search must have been previously defined either in this bundle or in the master server on which the bundle is being imported.

Attributes for the <hostSearchRef> Element

The <hostSearchRef> element has one attribute, *name*. This attribute provides the name of the host search reference.

<hostSearch> Element

The bundle <hostSearch> element is a child of the <memberList> element and is used to declare a host search to be referenced by the bundle.

The <hostSearch> element contains at least one of the following child elements:

- <criteriaList>
- <appTypeCriteria>
- <physicalCriteria>

Note – Although the <criteriaList>, <appTypeCriteria>, and <physicalCriteria> elements are each optional, one of the three must be provided.

Attributes for the <hostSearch> Element

The <hostSearch> element has two attributes:

- *name* – The name of the host search.
- *description* – An optional description of the host search. The default is no description.

<criteriaList> Element

The <criteriaList> element is a child of the <hostSearch> element. It specifies a list of criteria to be added to the <hostSearch> element. The <criteriaList> element must be specified if <appTypeCriteria> and <physicalCriteria> are not specified.

The <criteriaList> element contains one or more <criteria> elements. The <criteria> element specifies a search criteria, including name, match type, and pattern.

Attributes for the <criteria> Element

The <criteria> element has three attributes:

- *name* – The name of the host variable to match.
- *pattern* – The pattern to match.
- *match* – The match type of the criteria. Valid values are EQUALS or CONTAINS. The default is EQUALS.

<appTypeCriteria> Element

The <appTypeCriteria> element is a child of the <hostSearch> element. It specifies a list of application type criteria to be added to the <hostSearch> element. The arguments of the <appTypeCriteria> element are expressed as attributes, and order is not important. If all values are false or the element is empty or unspecified, the search disregards this criteria when performing the search. The <appTypeCriteria> element must be specified if <criteriaList> and <physicalCriteria> are not.

Attributes for the <appTypeCriteria> Element

The <appTypeCriteria> element has three optional attributes:

- *ms* – If true, match `MasterServer` application type in host search. The default is false.
- *ld* – If true, match `LocalDistributor` application type in host search. The default is false.
- *ra* – If true, match `RemoteAgent` application type in host search. The default is false.

<physicalCriteria> Element

The <physicalCriteria> element is a child of the <hostSearch> element. It specifies a list of physical type criteria to be added to the <hostSearch> element. The arguments of the <physicalCriteria> element are expressed as attributes, and order is not important. If all values are false or the element is empty or unspecified, the search disregards this criteria when performing the search. The <physicalCriteria> element must be specified if <criteriaList> and <appTypeCriteria> are not.

Attributes for the <physicalCriteria> Element

The <physicalCriteria> element has two optional attributes:

- *physical* – If true, match physical host types in host search. The default is false.
- *virtual* – If true, match virtual host types in host search. The default is false.

<component> Element

The <component> element is a child of the <memberList> element and is used to declare a component in the bundle jar. All objects referenced by this component must have been previously defined either in this bundle or on the master server on which the bundle is being imported.

The <component> element contains three optional child elements:

- <systemService>
- <componentType>
- <resource>

Attributes for the <component> Element

The <component> element has two attributes:

- *jarPath* – The location of the component XML file, relative to the root of the bundle jar (leading / or . characters are not permitted). The format of the component XML is specified by the Plan and Component Language specification. See Chapter 3, "Component Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* for more information.
- *majorVersion* – An optional attribute that determines whether to check in the component as a new major version. The default is false.

<systemService> Element

The <systemService> element is a child of the <component> element and is used to declare a system service backed by the containing component. This element may not be used with the <componentType> element. When the <systemService> element is used in a <component> element, a component is loaded and a <systemServiceRef> that references that component is created.

Attributes for the <systemService> Element

The <systemService> element has two attributes:

- *name* – The name of the system service
- *description* – An optional description of the system service

<componentType> Element

The <componentType> element is a child of the <component> element and is used to declare a component type backed by the containing component. The <componentType> element may not be used with the <systemService> element. When the <componentType> element used in a <component> element, a component is loaded and a component type that is backed by that component is created.

Attributes for the <componentType> Element

The <componentType> element has five attributes:

- *name* – The name of the component type.

  A name has a maximum of 64 characters. The name must start with a letter or underscore, followed by any number of letters, digits, or special characters, such as underscore (_), period (.), plus (+), minus (-), and space ( ). Unicode letters and digits are permitted.
- *description* – An optional description of the component type.
- *group* – The group name of the component type.

- *order* – A number that identifies where to put this component type.
- *indentLevel* – A number between 0 and 10 that identifies the level to which to indent this component type in a hierarchy of component types.

`<resource>` Element

The `<resource>` element is a child of the `<component>` element. It specifies a resource file name and location in the jar. A resource is always checked in as a either a file-typed or a directory-typed resource. The component that contains the `<resource>` element must be a simple component whose `<resourceRef>` element refers to the resource created by the `<resource>` element. No component variables are automatically added to the component that contains the `<resource>` element, regardless of the value of the `config` attribute.

Attributes for the `<resource>` Element

The `<resource>` element has three attributes:

- *jarPath* – The location of the resource file, relative to the root of the bundle jar. Leading `/` or `.` characters are not permitted. For directory-type resources, this path is assumed to be a directory, and is expected to end with a `/`. Everything in this directory defines the contents of this resource.
- *majorVersion* – An optional attribute that determines whether to check in the resource as a new major version. The default is false.
- *name* – An optional attribute that is the name of the resource. If not specified, the name will default to the absolute *jarPath*, which is converted to absolute if specified as relative.
- *config* – An optional attribute that specifies whether this resource is a configuration template.The default is false.
- *type* – An optional attribute that specifies whether the resource is a file or a directory. Use `FILE` for a file resource. Use `DIRECTORY` for a directory resource. If omitted, the default is `FILE`.
- *checkInMode* – An optional attribute that specifies whether a directory-type resource should be replaced or appended. Use `REPLACE` if the check in of this resource should replace an existing version. Use `ADD_TO` if the check in should add to the resource. The default is `REPLACE`. This attribute only applies to a resource that has a *type* of `DIRECTORY`.
- *descriptorPath* – An optional attribute that specifies the path to a resource descriptor file, relative to the root of the bundle jar. Leading `/` or `.` characters are not permitted. The format of the resource descriptor file follows the Resource Descriptor schema, as described in Chapter 5, “Resource Descriptor Schema,” in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

  If no resource descriptor file is specified, permissions information is used from the default file system settings of the master server. In this case, owner and group are not stored. This is also the case for settings that are omitted from a descriptor (either no entry or a partial entry for a file within the resource).

<plan> Element

The <plan> element is a child of the <memberList> element and is used to declare a plan in the bundle jar. All objects referenced by this plan must have been previously defined either in this bundle or on the master server on which the bundle is being imported.

Attributes for the <plan> Element

The <plan> element has two attributes:

- *jarPath* – The location of the plan XML, relative to the root of the bundle jar. Leading / or . characters are not permitted. The format of the plan XML is specified by the Plan and Component Language specification. See Chapter 4, "Plan Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* for more information.
- *majorVersion* – An optional attribute that determines whether to check in the plan as a new major version. The default is false.

◆ ◆ ◆ APPENDIX A

Appendix — New Commands

This appendix provides tables summarizing the usage of new commands provided with this release of the N1 Service Provisioning System.

TABLE A–1 bdb.b.la – Lists All Bundle Templates

Argument/Result	Class	Description
result	Bundle	The Bundle template instances

The `bdb.b.lo` command displays the details of a bundle template. Summary formatter is the default formatter. The summary formatter displays the same information as that in the `bdb.b.la` command, but for a single bundle template. Details could be retrieved with a `-o` detail argument.

TABLE A–2 bdb.b.lo – Displays Details of a Bundle Template

Argument/Result		Class	Description
ID	[R]	BundleID	The ID of the bundle template to view
result		Bundle	The bundle template details, the contained search criteria, a generated list of entities corresponding to these criteria, if any

TABLE A–3 bdb.b.add – Adds a New Bundle Template

Argument/Result		Class	Description
name	[R]	String	The name of the bundle template
desc	[O]	String	The description of the bundle template

TABLE A–3 bdb.b.add – Adds a New Bundle Template *(Continued)*

Argument/Result		Class	Description
criteria	[O]	SearchCriteriaArray	Search criteria for entities to be included in the bundle template. Criteria start with either F:(folder) HT:(hostType) HR:(hostSearch) HS:(hostSet) C:(component) CT:(componentType) SS:(systemServices) P:(plan), followed by comma seperated attribute=value pairs. Separate multiple such criteria by ';'. Attributes can be a combination of: `name`, `desc`, `vis` (visibility), `path` (folder path),`ver` (version), `ct` (extendsType), `lab` (label). Attributes depend the entity type.
result		BundleID	ID of the newly created bundle template

The `bdb.b.mod` command modifies the contents of a bundle template. Criteria that you specified with the `bdb.b.mod` replace existing criteria for the entity type. Criteria specified on the command line replace the pre-existing ones for that entity.

TABLE A–4 bdb.b.mod – Modifies a Bundle Template

Argument/Result		Class	Description
ID	[R]	BundleID	The ID of the bundle template to modify
desc	[O]	String	The description of the bundle template
criteria	[O]	SearchCriteriaArray	Search criteria for entities to be included in the bundle template. Criteria specified through this command overwrite any existing criteria the bundle template had. The syntax of usage is the same as for `bdb.d.add`
result		BundleID	ID of the modified bundle template

Note – For the `bdb.b.add` and `bdb.b.mod` command, the order in which the entities are exported or imported is by default decided by the order of their search criteria in the bundle template. Pattern strings that contain ',' , ';' & '=' characters can be included by using '\' as an escape character. Backslash itself can be specified as '\\' in the pattern.

TABLE A–5 bdb.b.del – Deletes a Bundle Template

Argument/Result		Class	Description
ID	[R]	BundleID	The ID of the bundle template to delete

TABLE A–6 bdb.b.exp – Exports a Bundle Template

Argument/Result		Class	Description
ID	[R]	BundleID	The ID of the bundle template to export
path	[R]	String	The path of the file system in which to save bundle jar that is created by the export

TABLE A–7 bdb.b.imp – Imports a Bundle Jar

Argument/Result		Class	Description
path	[R]	String	The path to the file system from which to import the bundle jar
owner	[R]	GroupID	The group that will be owner of folders created by the import
v	[O]	boolean	True if you want only to validate an import, without actually executing one. Default value is false.
result		BundleSessionInfo	Summary of the number and types of entities imported

Index

I

J

L

M

N

P

R

S

T

V