Sun N1 Service Provisioning System 5.2 Plan and Component Developer's Guide

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

Part No: 819–4446–10
April 2006

060331@14558

Contents

Preface

The *Sun N1 Service Provisioning System 5.2 Plan and Component Developer's Guide* provides information that helps developers write plans and components. It includes information about concepts that are related to plan and component development.

Who Should Use This Book

This book is for developers who want to use the Sun N1™ Service Provisioning System to install and manage applications in data centers.

How This Book Is Organized

Chapter 1 describes concepts that pertain to components, plans, component types, system services, and session variables.

Chapter 2 explains how to perform component-related tasks.

Chapter 3 describes the built-in component types.

Chapter 4 explains how to create, manage, and run plans.

Chapter 5 explains how to manage session variables.

Chapter 6 describes configuration generation and substitution variables.

Appendix A provides a description of the grammar used for variable substitution.

Related Books

The Sun N1 Service Provisioning System documentation includes these other books:

- *Sun N1 Service Provisioning System 5.2 Release Notes*
- *Sun N1 Service Provisioning System 5.2 Installation Guide*
- *Sun N1 Service Provisioning System 5.2 System Administration Guide*
- *Sun N1 Service Provisioning System 5.2 Operations and Provisioning Guide*

- *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*
- *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*
- *Sun N1 Service Provisioning System 5.2 Plug-In Developer's Guide*

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*.
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

Plan and Component Development Concepts

This chapter describes plan and component concepts that relate to the Sun N1 Service Provisioning System software, henceforth referred to as the provisioning system.

This chapter covers the following topics:

Component Concepts

A *component* is a logical grouping of *resources* that defines an application. It also includes a set of instructions that specifies how to handle the resources that make up the application. A component might be a collection of the following:

- Files and directories
- Autonomous archives, such as J2EE™ Enterprise Archives (EAR) or COM components
- Complete applications, such as the BEA WebLogic Server
- Operating system-level updates, such as patches or service packs
- Pointers to other components

You can use the provisioning system to manage applications in a data center. Before you can manage your applications with this product, you must first *model* them as components. The provisioning system enables you to do the following:

- Create application models that include a carefully defined group of software resources for each application, along with information about how the application should be installed, configured, and analyzed
- Store components in a component repository that employs version control so you can retrieve any previous version of a component

- Make components available to plans that perform data center operations in a step-by-step manner to take advantage of the knowledge that is embedded in each component
- Compare components to one another as well as to installations of software

The provisioning system supports two basic types of components:

- **Simple components.** A *simple component* contains a single resource, but cannot contain references to other components.
- **Composite components.** A *composite component* contains only references to other components, both simple and composite, but cannot contain any resources.

A composite component contains references to components. The referenced components are referred to as *contained components* (child components), and the referencing component is referred to as the *container component* (parent component).

A composite component declares whether each of its contained components is to be installed as a *top-level component* or a *nested component*. If a contained component is installed as top-level, it can be used by any component just as if it had been directly installed by a plan. However, if a contained component is installed as nested, its services are only available to the container component. A nested contained component defines a finer-grained unit of functionality required by the container component, but is not otherwise useful to other components. Whereas a top-level contained component defines services that will be used by the container component, it can also be used by other components.

Note – Composite components contain only references to other components, not to the components themselves. The referenced components are other existing components that are updated and managed separately from the container component. A component can be referenced by any number of composite components. A component's name is not affected by whether it is referenced by other composite components. Name conflicts are resolved using paths.

The provisioning system manages the physical resources associated with a component. It also includes a number of predefined components that you can use directly, or as samples for constructing other components.

A component also includes procedures. A *procedure* is a program that is contained in a component that controls deployment of the component. Typical component procedures are defined for installation, uninstallation, and capturing *snapshots*. Other procedures can be defined in the control block. A *control* is a series of instructions in a component that can be used to manage the deployed application. For example, controls might be used to start or shut down an application. A component might include instructions that test for dependencies on other components or that verify that a particular process is running.

A component can also declare *substitution variables* that can be used within the component itself or for any resources that the component contains. These variables can be used as placeholders for values that will be substituted when the plan is run.

Modeling a Component

The provisioning system offers great flexibility when you model components. The application that you are modeling determines which of the following approaches to use:

- **Fully automated modeling.** You can check in a component from a *gold server* or from a source code control system. When the check-in completes, the provisioning system has automatically generated a *component*, from which you can run installation, configuration, and *comparison* procedures.

  Use this approach when you model applications for which a plug-in is already defined and imported. You can use built-in component types or imported component types to automatically model the most common resources that make up application components. The component type templates include built-in procedures for basic operations, such as installations. This means that you can perform basic operations on common types of components without having to write a plan.

  For descriptions of the built-in component types, see Chapter 3. For a list of component types delivered through plug-ins, see the *Sun N1 Service Provisioning System Plug-In Collection — Current Version* at `http://docs.sun.com/db/coll/1329.1`.

- **Extending built-in component types with XML authoring.** You can customize an automatically generated component by editing its XML directly on the Advanced Edit page of the browser interface. Another way to customize the component is to download a file that contains the XML to your system and edit it with an XML-schema-validating editor, such as Turbo XML.

  When you edit the XML, you can do the following:

  - Customize the component by supplying additional variables
  - Add calls to extended control procedures, such as starting and stopping IIS or Microsoft Windows services

- **Authoring component models in XML.** You can create a component on your own by using an XML editor and by referring to the component schema descriptions in Chapter 2, "Shared Schema Used by Components and Simple Plans," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*, Chapter 3, "Component Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*, and Chapter 4, "Plan Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

  Before you can use the component, you must check the component's XML file and its resources in to the repository.

- **Authoring component models by using the browser interface.** You can use the browser interface to create a component. Saving it automatically builds the component. Then, you can install the component by doing one of the following:

  - Running the installation procedure, which installs the component on a single *host* or a *host set*
  - Writing a plan to install the component on one or more hosts or host sets

    For more information about plans, see Chapter 4.

  Before you can use the component, you must first check it in to the repository.

When you check in a component, it is built, and particular versions of the component resources are assigned to it. The build also assigns a version number to the component and ensures that the appropriate version of the component is always associated with particular versions of the component's resources.

Component Characteristics

A component is distinguished by many characteristics. The following list describes some of the more common component characteristics:

- **Path.** A location in the folder in which to store components. Use folders to organize components in a hierarchical manner.
- **Component name.** The name of the component.
- **Component type name.** The name of the component type that is associated with a component. See "Component Type Concepts" on page 17.

  The provisioning system includes some component types that support generic models, such as files and directories. Additional component types can be added to the system by importing plug-ins that are specific to application domains, such as Microsoft Windows and WebLogic.
- **Version number.** The version number of the component. Each time a component is modified, the version number increments.

  At check-in time, you choose how to increment the version number. You can increment by the major number, which is to the left of the decimal point, or by the minor number, which is to the right of the decimal point.
- **Platform.** The operating system on which this component can be installed.
- **Check-in date.** The date and time when the component was checked in.
- **Check-in user.** The user ID of the person who checked in the component. This attribute is useful when you want to audit provisioning system processes.
- **Label.** An optional string (maximum length of 32 characters) that you can use to categorize or group components.
- **Category.** An optional object that you can use to filter the component list. After you create a category object, you can subsequently use it to group components.
- **Description.** An optional string that describes the component. Use this attribute to provide meaningful information about the component.
- **Source.** The resource that has been included in this component. The source can be a single file resource or a list of other components.
- **Component variables.** A list of variables and their values (name-value pairs) that are required to deploy a component resource. See "Component Variables" on page 15.
- **Procedures.** A set of instructions that specifies what to do with the resources and variables.
- **Hidden.** A characteristic that indicates whether you can view the component in a list of components. By default, components are not hidden.

You might want to shorten a component list by hiding components that are older versions or that are not being managed. The shortened list would include any non-hidden components, which might be those that you are currently managing. Hidden components do not appear in lists unless you request to view them.

Component Procedures

A *component procedure* (or *procedure*) is a program that is contained in and manages a component. A component procedure might install, uninstall, or control a component. A procedure is created when you build the component.

You can run a procedure in these ways:

- Directly from the Details page of a component, where a plan is generated and then executed
- By writing a plan that calls the procedure, then running the plan
- By calling the procedure from other components and plans

Your component can have several procedures. For example, a component might include a control procedure to start or stop the application that it models.

When you create a component that extends from a component type, the component inherits the procedures that are defined by the component type.

When you create a composite component, it inherits default installation and uninstallation procedures. Usually, components inherit procedures from the associated component type.

Component Inheritance

Component inheritance is the means by which a component obtains attributes and behavior from another component. When you create a component, it inherits any *variables*, *snapshots*, and *procedures* from the associated component type.

The use of inheritance makes the component model powerful and flexible. For example, suppose that you have a thousand components that are based on the IIS Application component type. You want to add more functionality to these components. By adding the functionality to the IIS Application component type, each of the thousand components that extend it will inherit the new functionality.

Component Variables

Components support the use of *variables*, which are user-definable containers that store values and are used during *deployment*.

A *component variable* is used to make parts of a component accessible and configurable by objects that are external to the component. For example, a component might have a variable named `installPath` that is overridden on a per-host basis. This variable defines where to install each component.

The value of a component variable can be a reference to another component variable, which includes variables that are defined by the component's container. When a nested component is added to a container component, the browser interface verifies that the referenced variable is defined in the container. If the variable is not defined in the container component, the browser interface automatically adds the referenced variable to the list of the container's variables.

For example, simple components typically define their `installPath` variable to have the value of the container component's `installPath` variable. In this example, the syntax of the referenced variable is `:[container:installPath]`. For more information about variable substitution, see Chapter 6.

Component variables are evaluated and assigned a value when the component is deployed. Component variables are used to specify information (such as host names, IP addresses, and paths) that is required to implement a deployment.

Variable Settings

Variable settings are collections of variable values that can be used to override the default values of one or more component variables. Based on the variable settings that you use, you can specify different values for component variables. You specify which variable settings to use when you run a plan.

For example, a component is deployed to different environments, such as a production environment and a development environment. If the defined component variables are set up to recognize the differences between environments, you can use variable settings to configure the component for each environment. For example, use one set to configure the production environment and another set to configure the component for the development environment.

Variable Overrides

Variable overrides enable you to override variable default values for composite components. They cannot be used for simple components.

When a component contains other components, called *child components*, variable settings only affect the top-level *parent component*. All child components use the default values for their variables. A child component can obtain variable values from its parent component in these two ways:

- **The container (parent) component "pushes" variable values to the contained (child) components.** To override the default values for child components, set the variable overrides when you create a component that contains child components. Each referenced component has a set of variable overrides that you can use.
- **The contained (child) components "pull" variable values from the container (parent) component.** One or more of the contained component's variable values are defined based on the value of a variable in the container component. The contained component uses the variable substitution syntax `:[container:`*varname*`]` in the default value of its variables.

Steps

Steps are simple instructions that can be part of both plans and components.

For information about steps, see "Step Overview" on page 18.

Component Type Concepts

A *component type* is a component that has been designed as an encapsulation of behavior for reuse by other components. A component type usually represents behavior that is specific to a particular type of application. For example, an Enterprise JavaBeans™ (EJB™) component type might contain generic procedures that can be used to deploy and undeploy an EJB archive to and from an application server. When a component *extends* from a component type, it automatically inherits the behavior that is defined in that component type.

Components are usually associated with a component type. If you select `untyped` from the browser interface, your component will not extend from any other component type. The provisioning system includes some built-in component types. See Chapter 3.

The files, directories, and other tree structures that are referred to by a simple component are managed as a discrete unit in a component.

For example, an IIS application, which the provisioning system would manage as referenced resources, might include the following:

- Directory and its contents
- IIS web site settings
- COM+ application
- Microsoft Windows registry settings

Some of the resources that are referenced by components, such as files and directories, can be copied from a gold server or another data source. Others, such as IIS web site settings or Microsoft Windows registry entries, must be extracted from a data source in a special way so that they can be treated as independent, manageable entities.

With its built-in component types, the provisioning system can recognize the most common source items that are used for Java™ 2 Platform, Enterprise Edition (J2EE platform) and Microsoft Windows applications. These component types can accurately extract data for use as a component resource, store the component resource in a repository, and correctly install the resources in the specified location.

System Services Concepts

System services are components that are automatically deployed to all applicable hosts when the hosts are prepared. System services define utility controls and resources that can be used by other components for component installation and management.

System services enrich the set of services that are available during plan execution.

Plan Concepts

A *plan*, also called an *execution plan*, is a sequence of instructions that is used to manage one or more components on the specified hosts. For example, an execution plan might install three components and initiate the "startup" control of another component.

A plan can also include a sequence of other plans, which enables common instruction sequences to be written as plans and then shared among plans. For example, a plan might instruct the provisioning system to install three components and initiate the startup control for another.

The provisioning system provides an in-memory representation of the objects expressed by the *XML schema*. This representation also defines a process for the validation, persistence, and version control for those objects.

When the provisioning system executes a plan, substitution variables that are declared by a component are replaced by actual values. The provisioning system also supports a notification feature that can send email in response to events that are related to plan execution.

Plan Types

The provisioning system supports two types of plans:

- **Simple plans.** A *simple plan* contains a collection of simple steps, but cannot call other plans.
- **Composite plans.** A *composite plan* contains only other plans, called subplans.

The XML schema enforces the distinction between the two types of plans. Thus, you can use a top-level plan that contains calls to other subplans, or a simple plan that contains simple steps but no calls to subplans.

This distinction is important because the steps in a simple plan can only execute on the same set of target hosts, whereas the steps of a composite plan can execute on different sets of target hosts. A composite plan can use one set of target hosts for each simple plan that it contains.

Step Overview

Steps are simple instructions that can be part of both plans and components. The provisioning system supports the following kinds of steps:

- **Steps that can only be called from within a component.** Such steps can only be called from install blocks or from uninstall blocks. See "Install-Only Steps for Components" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* and "Uninstall-Only Steps for Components" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.
- **Steps that can only be called from a plan.** Such steps can only be called from composite plans or from simple plans. See "Plan-Only Steps for Composite Plans" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* and "Plan-Only Steps for Simple Plans" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.
- **Steps that can be called from either a plan or from a component.** See Chapter 2, "Shared Schema Used by Components and Simple Plans," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

Component References

A number of steps can reference components in the following situations:

- Components that have yet to be installed can use the `<install>` step.

  Steps that refer to components that have yet to be installed only need to specify the name of the component and an optional version.
- Components that have already been installed can use the following steps:
  - `<uninstall>`
  - `<call>`
  - `<checkDependency>`
  - `<createDependency>`
  - `<addSnapshot>`

  Steps that refer to an installed component must specify the *installPath* attribute because the same component can be installed more than once on the same server.

EXAMPLE 1–1 Using Component References

Assume that the "Apache" component is installed on a host with the following attribute settings.

Component Instance	*installPath*	*version*	*installDate*
A	`/opt`	1.3	6/1/01 6:00 p.m.
B	`/usr/local`	1.4	6/1/01 5:00 p.m.
C	`/opt`	1.2	6/2/01 5:00 p.m.
D	`/usr/local/bin`	1.4	6/3/01 5:00 p.m.
E	`/export`	1.1	6/4/01 5:00 p.m.

The following shows which installed component is referenced when values for various combinations of the *installPath* and *version* attributes are supplied.

installPath	*version*	*versionOp*	**Result**	**Explanation**
None	None	None	E	The most recently installed component on the target host is used, regardless of the values of *version* and *installPath*.
`/opt`	None	None	C	The most recently installed component in the named install path is used regardless of the value of *version*.
`/usr/bin`	None	None	ERROR	No component is installed in the specified path.
None	1.4	=	D	The most recently installed component that has the specified version is chosen, regardless of the value of *installPath*.
None	1.5	Any	ERROR	No component is installed with the specified version.
`/usr/local`	1.4	=, >=	B	The component that matches the specified *installPath* and *version* attribute values is chosen.
`/usr/local`	1.2	=	ERROR	No such version is installed at the specified install path.
`/usr/local`	1.2	>, >=	B	The values for *installPath*, *version*, and *versionOp* match.
`/opt`	3	Any	ERROR	If two or more components are installed in the same path and they have the same name, the most recently installed component effectively overwrites any other components that have been installed earlier. Components that have been installed at an earlier time cannot be accessed, even if directly specified.

Resource Descriptor File Concepts

A *resource descriptor file* specifies the owner, group, and permission settings to use for the files and directories that comprise the resource of a simple component. This resource descriptor is an XML file. For information about the XML schema for this file, see Chapter 5, "Resource Descriptor Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*. By using a resource descriptor file, you can override the permissions that are determined at component check-in time.

When a resource descriptor file is not used, a resource uses the owner, group, and permission settings it has at check-in time. This situation is the default case when you perform the check-in on a UNIX system. When you check in a component on a Windows system, the default settings are as if you used the `:NONE:` value for each of the settings in a resource descriptor file.

When you use a resource descriptor file, a resource uses the owner, group, and permission settings specified by the resource descriptor. If an <entry> element is specified for a resource, the settings are taken from that entry. If the entry does not specify all of the settings, the missing setting values are taken from the <defaultEntry>, if present. If those setting values are not specified in a <defaultEntry>, the resource uses the setting values it had at check-in time.

If no <entry> element is specified for a resource, the resource uses the settings specified in <defaultEntry>, if present. If no <defaultEntry> is present, the resource uses the settings it had at check-in time.

Use the :NONE: value to tell the provisioning system to use the default settings from the file system on which the component will be deployed. You can specify the :NONE: value for any setting specified in a <defaultEntry> block or in an <entry> for a resource.

A resource descriptor file is only used when deploying a component to a UNIX system. If a component is deployed to a Windows system, the resource descriptor file is ignored. So, if your component only applies to Windows systems, do not create a resource descriptor file.

A resource descriptor file can be used by simple components that extend the system#file and system#directory component types. A resource descriptor file can also be used by a component that extends the com.sun.linux#rpm component type, which is part of the Linux plug-in.

You check in the resource descriptor file at the same time that you check in your component. When you attempt a checkin-current for a component that used a resource descriptor file for its last checkin, the provisioning system expects to find the resource descriptor in the original check-in location. Thus, if you move the file to a different location and attempt a checkin-current for the component, the check-in operation fails.

You can download the resource descriptor for a simple component that has been checked in to see the settings for every file that is part of the component's resource.

You might use this download feature to update the file and check in an updated version of the component. First, you download the resource descriptor file and then check out the associated component's resource. Then, you modify the resource descriptor file and use it to check in a new version of the component.

The resource descriptor you download might differ from the resource descriptor you used to check in the component. Differences might appear because the descriptor you use to check in a component is not required to have information about every file in the resource, or to have full information for every entry. Notice that no <defaultEntry> block appears in the downloaded resource descriptor file. Instead, every file is described in its own <entry>.

Session Variable Concepts

A *session* is initiated when you log in to the browser interface or use the CLI. A session persists until you log out or inactivity causes the session to expire. Logically, a session represents the authenticated credentials of a particular user. A session is used to identify the user throughout a series of related requests without reauthentication.

Each session can have a set of *session variables* that are initialized from the database when you log in. You can use session variables to preserve session-related information, such as user authentication and other credentials. You can modify the session variables in the current session without affecting the session variables that are saved in the database. If you make changes to session variables and their values, you can save them to the database. When saved, all of the variables and their values are saved and are available in future sessions. You can make session variable references when you execute plans and comparisons.

If you have initiated more than one session and try to save the session variables at the same time, only the first session to save the variables succeeds. After the first session successfully saves the session variables, the other sessions are prevented from saving changes to their session variables. To save those changes, do the following:

1. Log out of all sessions.
2. Log in again to restart a session.

   The new set of session variables and values are retrieved from the database based on the last successful save.
3. Make the changes you want to the session variables.
4. Save the changes to the session variables.

Session variables exist in a global namespace, which means that the session variable names you declare are available to all user sessions. For example, suppose that you define a session variable named `passwd`. Any plan that requests the `passwd` session variable is referring to the same variable. Session variable names are not scoped to current plans, components, blocks, or hosts. Thus, you must ensure that your session variable names are unique. For example, you might use your initials and birthday month and day, or some other identifier, to make your session variables unique.

Secure Session Variables

A session variable is made up of a name, value, and the `secure` flag. The session variable name has the same limits as a host-type variable name. The `secure` flag is a Boolean value that describes whether the value should be securely stored.

When a value is securely stored, the `secure` flag is set to true. The value of the session variable appears as *** and is encrypted when the variable is saved. When this value is used during plan execution, the value of this variable is obscured from plan history output, as well. Use this `secure` flag when the variable value is sensitive data, such as passwords. By default, session variables are created with the `secure` flag set to true.

If any secure variables are saved to the database, you must type your password, which is used as the encryption key. If you supply the wrong password, you are prompted for the password again.

If the password you enter is not recognized, an error is issued. This might occur if the system administrator has changed your password or your login configuration. You are given the following options:

- **Reencrypt the session variables.** Supply your user name, the password with which the session variables have been encrypted, and your new password.
- **Flush the session variables.** Delete all of the session variables that you declared. Supply your user name and your current password.

  Use this option if you forgot the password with which the session variables were encrypted.

◆ ◆ ◆ CHAPTER 2

Components

This chapter describes how to manage components and covers the following topics:

Managing Components

You can use the Sun N1 Service Provisioning System software browser interface to manage components.

You can also use the command-line interface (CLI) to manage components. See "Summary of Component CLI Commands" on page 30. For detailed information about CLI commands, see the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

The following pages in the browser interface include information about how to view and manage components:

- **Components page.** List components and add new components to the list. You can also access other component pages to view component details.
  - To *see* which components are already checked in to the repository, choose Components from the navigation menu.
  - To *see details* about a component, go to the Components page, click the name of the component you want, and view details on the Details page.
  - To *create* a component, see "How to Create a Component" on page 26.
  - To *see where a component is installed*, go to the component's Details page and click Where Installed.
- **Details page.** View detailed information about the component, such as its attributes and values. This page also provides information and buttons that enable you to manage the component.

  The information shown on the Details page depends on the component type of the component.

- To *edit* a component, go to the component's Details page, click Edit, modify the component configuration on the Edit page, and click Check In. You can choose which version number to use, then click Continue To Check In.
- To *delete* a component, see "How to Delete a Component" on page 29.
- To *rename* a component, go to the component's Details page and click Rename. Specify the new name of the component and click Rename, or click Cancel if you do not want to rename the component.
- To *move* a component to another folder, go to the component's Details page and click Move. Specify the name of the folder in which to move the component and click Move, or click Cancel if you do not want to move the component.

Note – If you plan to move a component into a folder that does not yet exist, click New Folder in the Move Component window. This new folder inherits the permissions from its parent folder. After you click Create, the folder tree displays with the new folder selected. Click Move.

- To *edit the component's XML*, go to the component's Details page and click Advanced Edit. Make the changes you want to the XML. To overwrite the current XML, click Check In. To create a new component based on the changes to the XML, click Check In As. Or, click Cancel if you do not want to change the component's XML.
- To *download the component's XML* to your system, go to the component's Details page and click Download. Click OK to continue the download, or click Cancel to cancel. Specify the directory in which to download the component's XML and click Save, or click Cancel if you do not want to download the component's XML.
- To *modify the component's variable settings*, go to the component's Details page and click Variable Settings. Click Create Set to create new variable settings, or click Import Set from Component to import variable settings from your system.
- To *see where the component is installed*, go to the component's Details page and click Where Installed.

- **Edit page.** Change component attribute values. Note that all of the fields cannot be changed.

  The information shown on this page depends on the component type of the component.

▼ How to Create a Component

Use this procedure to create simple and composite components.

1 From the navigation menu, choose Components.

The Components page appears and lists the components that are already checked in.

2 (Optional) Click Change Folder and specify the folder that will contain the new component.

A window appears where you specify the name of the folder in which to create the component.

- **If the folder exists, select the name of the folder, and click Change to Selected Folder.**

  The Components page displays the list of components in the specified folder.

- **If the folder does not exist, create the folder.**

  a. **Specify the parent folder, and click New Folder.**

     The Create New Folder window appears.

  b. **Type the name of the new folder, and click Create.**

     The new folder inherits its permissions from its parent folder. After you click Create, the folder tree displays with the new folder selected.

     Note – You cannot create multiple folder levels at one time. To create several nested folders, you must return to the Change Folder window and repeat steps A and B.

  c. **After the folder is created, select the name of the folder, and click Change to Selected Folder.**

     The Components page displays the list of components in the specified folder.

3 Type a name for the new component in the Component field, and click Create.

The new component's Edit page appears.

4 Define the component.

a. **(Optional) Change the component's name in the Component field if necessary.**

b. **Choose the component type from the Component Type drop-down menu.**

   Some of the resulting fields might change to match those that are used by the component type that you selected.

c. **From the Platform Type drop-down menu, choose the platform type to which this component can be deployed.**

d. **(Optional) Specify a label in the Label field.**

e. **(Optional) Specify a description in the Description field.**

f. **Supply other required information.**

   The information that is required is based on the component type that you selected.

If your component references other components, local names are created for each of the components that this component references.

g. Click Check In.

A window appears that reports that the provisioning system is checking in the component as Version 1.0.

5 Click Continue To Check In.

How to Clone a Variable Set

Variable set cloning allows for the creation of new variable sets based on existing variable sets, instead of from the default set.

The variable settings from previously checked in components can be downloaded to files and then imported. Select the Create Set link from the Create and Edit Variable Settings table of the `components>details>variable settings` page. The following selections are available from the page:

- Create Set link is used to create a new variable set. If selected, a new column will be added only if there were no previously created variable sets. The column is headed by a Set Name field that is awaiting input of a unique variable set name in which to store the component variable settings. The edit links are Save, Cancel, and Delete
- Import Set From Component link will import variable sets from other components
- Download All Sets link will download all variable sets
- Import Sets From File field, Browse, and Import links allow for the selected file to be uploaded and imported. New variables sets are created for the imported variables sets. No variable sets will be created on errors. On success, a message is sent in the session about the list of variable sets created. On error, an error message is sent in the session.

1 From the navigation menu under Application Deployment, choose Components.

The `components>[top]` page appears and lists the components that are already checked in.

2 Select an already checked in component.

The `components>details` page should appear.

3 Scroll down to the bottom of the page and select Variable Settings.

The `components>details>variable settings` page should appear.

4 Click on the Download All Sets link..

Export variable sets to a file (assuming that variable sets have been created already)..

5. **Save the exported data.**

   A window will pop up to ask for the file path to save the exported data. On some browers, if the exported data displays, the user has to use the browser menu to save the exported data (Choose File->SaveAs).

6. **Click on Done when you finish exporting variable settings.**

7. **To use a cloned variable set, select Create from the `components>[top]` page.**

   Create a new component and perform a check in.

8. **Select the newly created component and click on Variable Settings.**

9. **Import Variable Sets**

   - **To import variable sets from another component, click on Import Set From Component. A popup window will be displayed to allow the user to select a component to import. Once selected, the variable sets in the import component will be imported if both have variables in common.**

   - **To import variable sets from a file, click on the Browse button at the bottom of the variable settings page to specify an import file. An example of an import file would be previously exported data, such as in Steps 4 and 5, then click on the Import link. The file content is uploaded, parsed and new variable sets are imported and created.**

10. **Click Done, and the newly created component has used cloned variable settings.**

▼ How to Delete a Component

Note the following restrictions before deleting a component:

- A component that is referred to by another component cannot be deleted.
- An installed component cannot be deleted until it has been uninstalled.
- A component that has been created by a plug-in cannot be deleted except by deleting the plug-in.
- The root component of a family cannot be deleted if non-root members of the family exist.
- A component cannot be deleted during a plan run because all delete operations and plan runs acquire a system lock that keeps those operations from running concurrently.

When a component is deleted, the installation records are also deleted. This applies only to a component that has been installed and uninstalled, so no user-visible change is noticeable. A resource that is associated with a component, if any, is deleted with the component. Any plans that were created by running a direct-run component procedure are also deleted. The plan history that installed the component is edited to indicate that the component the plan installed was deleted. Plan history itself is not removed.

1. **From the navigation menu, choose Components.**

   The Components page appears and lists the components that are already checked in.

2. **Click Change Folder.**

   A window appears where you specify the name of the folder from which to delete the component.

3. **Specify the name of the folder, and click Change to Selected Folder.**

   The Components page now lists the components in the specified folder.

4. **Click the name of the component that you want.**

   The Details page for that component appears.

5. **Click Delete.**

   A window appears that reports that the provisioning system is about to delete the component you selected.

6. **Click Continue To Delete.**

   Click Cancel if you do not want to delete the component.

Summary of Component CLI Commands

You can use the `cdb.c` commands to manage components. See Chapter 3, "cdb: CLI Commands for Managing Components," in *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

- `cdb.c.ci` – Creates a new version of an existing component, or creates an initial version of a nonbrowsable component using XML format
- `cdb.c.co` – Checks out a component in XML format
- `cdb.c.del` – Deletes a component
- `cdb.c.la` – Lists all versions of all components
- `cdb.c.lo` – Shows detailed information about a component
- `cdb.c.lv` – Lists all versions of a component
- `cdb.c.mod` – Modifies a component
- `cdb.c.mv`– Moves or renames a component
- `cdb.c.sc` – Applies one or more categories to a component
- `cdb.c.sh` – Shows or hides a component

The following `cdb.ic` commands retrieve information about components that have already been installed on hosts:

- `cdb.ic.lbc` – Lists all of the hosts on which a component is installed

- `cdb.ic.lbh` – Lists all of the components that are installed on a specific host
- `cdb.ic.ldo`–Lists dependencies on other components
- `cdb.ic.lod`–Lists the components that are dependent upon this component
- `cdb.ic.vs.lo` – Shows the details of the specified generated variable settings

The following `cdb.rsrc` commands manage resources:

- `cdb.rsrc.ci` – Checks in certain components and their source objects
- `cdb.rsrc.cib` – Checks in all of the resources listed in a batch file
- `cdb.rsrc.co` – Checks out the specified resource
- `cdb.rsrc.gd` – Generates a resource descriptor for the specified component
- `cdb.rsrc.rci` – Rechecks in a resource
- `cdb.rsrc.showopts` – Shows the check-in options supported by a particular component type

The following `cdb.vs` commands manage variable settings for components:

- `cdb.vs.add` – Adds new variable settings
- `cdb.vs.del` – Deletes variable settings
- `cdb.vs.imp` – Imports variable settings from one component to another
- `cdb.vs.la` – Lists all variable settings that are associated with a specific component
- `cdb.vs.lo` – Lists the details of specific variable settings
- `cdb.vs.mod` – Modifies variable settings

Checking In a Component by Using the Command-Line Interface

When you check in a component, you copy a particular resource from a data source, such as a directory on a gold server, to the component repository. The *component repository* is a hierarchical namespace. Within this namespace, components are identified by name and version number.

A component must also have a component type that identifies the format and, in many cases, the function of a component. The built-in component types that are available with the provisioning system are described in Chapter 3.

When you use the `cdb.rsrc.ci` command to check in a component, use the following options:

- `-src` – Specifies the source location of the resource
- `-dst` – Specifies the place in which to store the component in the component repository
- `-type` – Specifies the component type

More than one component can reference the same resource. Checking in a resource using the `cdb.rsrc.ci` command associates that resource with the specified component.

◆ ◆ ◆ CHAPTER 3

Built-in Component Types

You can use built-in component types to quickly model many of the most common WebLogic, Microsoft Windows, and J2EE application components. These component types enable you to automatically associate installation, uninstallation, export, and snapshot behavior with a particular component.

The following component types are available as part of the base product:

Other component types, such as those for WebLogic and Microsoft Windows applications, are available through plug-ins. For more information, see the `http://docs.sun.com/app/docs/coll/1329.1`.

Component Type: `system#file`

The `system#file` component type represents a single file that is taken from a target server. The provisioning system deploys a file directly with no special postprocessing. The `system#file` component type includes installation, uninstallation, and snapshot procedures.

The `system#file` component type defines the following variables:

- `installName` – The name to use for the resource when it is installed. The default value is the name of the component.
- `installPath` – The path in which to install the resource. The default value is the value of the `installPath` variable of the container component.
- `installPermissions` – The permissions to assign to the resource when it is installed. See the `chmod`(1M) man page. The default value is empty.
- `installUser` – The owner of this resource when it is installed. The default value is empty.

- installGroup – The group to assign to this resource when it is installed. The default value is empty.
- installDiffDeploy – Specifies whether the resource should be deployed in differential deploy mode. The default value is TRUE.
- overrideRsrcInstallPath – The absolute path in which to install the resource. If not defined, the resource is installed in the path specified by installPath. This variable is not commonly used so that installPath can be used to specify the path in which to install both the component and the resource.
- installDDeployMode –Specifies whether to add files to the directory (ADD_TO) or replace the files in the directory (REPLACE). The variable and value are fixed at REPLACE for this type.

Browsing

	UNIX Systems	Microsoft Windows Systems
Root Path	/	List of physical drives on the host or network are mapped to a drive letter. Removable media is not shown.
Delimiter	/	\
Ordering	Alphabetical with directories appearing first	
Selection Type	User can select a file for check-in. Double click a directory to view its contents.	
Sample Path	/foo/foo.txt	C:\foo\foo.txt
Special	Links display their local name and the location pointed to: foo->/usr/bar	

Extended Control Procedures

The following procedures are defined for the system#file component type:

- default **install block** – Deploys the contained file based on the state of the variables. No snapshots are captured.
- markOnly **install block** – Marks the system to indicate that a new version of the component is installed. No resources are transferred, and no snapshots are captured.
- default **uninstall block** – Undeploys the contained file from the location to which it was originally deployed.

- markOnly **uninstall block** – Marks the system to indicate that the current version of the component has been uninstalled. The component's resources are not actually removed from the target host.
- default **snapshot block** – Captures a snapshot of the deployed file. Because the default install block does not capture the file by default, container components must call this routine explicitly if you want a snapshot.

Component Type: system#directory

The provisioning system can accommodate any number of components and can place them in any directory structure that you create. The number of components that can be created is only limited by the file system on the master server. Each file system limits the number of files that can be created, and by creating too many files in a single directory, you can exceed the file systems limitations. The relationship between components and files is not necessarily an equal ratio. For example, one component might exceed file system limitations if it has 100,000 files, and 100,000 components might fall within the file system limitations if none of the components contain files.

Exceeding file system limits might not result in a direct failure when the files are created. However, problems might arise in the form of increasingly severe performance degradation or unpredictable failures at other times.

Limits to the number of files that a directory can support vary by file system and can be influenced by how the operating system is configured. You can avoid overloading a directory by understanding your file system's limitations and placing components into smaller subdirectories, rather than into a single, large directory. A conservative practice to use when creating components is to place no more than 30,000 files in a single directory.

The system#directory component type represents a collection of files and folders that are taken from a target server. The system#directory component type includes installation, uninstallation, and snapshot procedures.

The system#directory component type defines the following variables:

- installName – The name to use for the resource when it is installed. The default value is the name of the component.
- installPath – The path in which to install the resource. The default value is the value of the installPath variable of the container component.
- installPermissions – The permissions to assign to the resource when it is installed. See the chmod(1M) man page. The default value is empty.
- installUser – The owner of this resource when it is installed. The default value is empty.
- installGroup – The group to assign to this resource when it is installed. The default value is empty.
- installDiffDeploy – Specifies whether the resource should be deployed in differential deploy mode. The default value is TRUE.

- overrideRsrcInstallPath – The absolute path in which to install the resource. If not defined, the resource is installed in the path specified by installPath. This variable is not commonly used so that installPath can be used to specify the path in which to install both the component and the resource.
- installDeployMode – Specifies whether to add files to the directory (ADD_TO) or replace the files in the directory (REPLACE). The default value is REPLACE.

Browsing

	UNIX Systems	Microsoft Windows Systems
Root Path	/	List of physical drives on the host or network are mapped to a drive letter. Removable media is not shown.
Delimiter	/	\
Ordering	Alphabetical with directories appearing first	
Selection Type	User can select a directory for check-in. Double click a directory to view its contents.	
Sample Path	/foo/foo/	C:\foo\foo\
Filters	None	
Special	Links display their local name and the location pointed to: foo/->/usr/bar/	

Extended Control Procedures

The following procedures are defined for the system#directory component type:

- default **install block** – Deploys the contained file based on the state of the variables. No snapshots are captured.
- markOnly **install block** – Marks the system to indicate that a new version of the component is installed. No resources are transferred, and no snapshots are captured.
- default **uninstall block** – Undeploys the contained file from the location to which it was originally deployed.
- markOnly **uninstall block** – Marks the system to indicate that the current version of the component has been uninstalled. The component's resources are not actually removed from the target host.

- default **snapshot block** – Captures a snapshot of the deployed file. Because the default install block does not capture the file by default, container components must call this routine explicitly if you want a snapshot.

Component Type: system#symbolic link

The system#symbolic link component type represents a single symbolic link that is taken from a target server. The provisioning system deploys a symbolic link by creating a new link on the target server. The file that the symbolic link points to is taken from the value of the symlinkTarget variable. Any file that exists in the install location of the link is removed before the new symbolic link is created.

The system#symbolic link component type includes installation, uninstallation, and snapshot procedures. The uninstallation procedure removes the symbolic link, not the file to which it links. The snapshot procedure also captures the name of the file to which the symbolic link points, not to the file itself.

The system#symbolic link component type defines the following variables:

- installName – The name to use for the resource when it is installed. The default value is the name of the component.
- installPath – The path in which to install the resource. The default value is the value of the installPath variable of the container component.
- symlinkTarget – The absolute path of the file to be linked to by this link. The default value is the file that was originally linked to by the file.

The following procedures are defined for the system#symbolic link component type:

- default**install block** – Removes any file that previously existed in the install path and replaces it with a symbolic link that links to the value of the symlinkTarget variable. No snapshots are captured.
- default**uninstall block** – Removes the deployed link without removing the file to which that it links.
- default **snapshot block** – Captures a snapshot of the deployed link. Because the default install block does not capture the file by default, container components must call this routine explicitly if you want a snapshot. If you request a snapshot of a symbolic link, the path pointed to by the symbolic link is captured.

Browsing

	UNIX Systems	Microsoft Windows Systems
Root Path	/	List of physical drives on the host or network are mapped to a drive letter. Removable media is not shown.
Delimiter	/	\
Ordering	Alphabetical with directories appearing first	
Selection Type	User can select a file for check-in. Double click a directory to view its contents. Only symbolic link type files can be selected for check-in.	
Sample Path	/foo/foo.txt	C:\foo\foo.txt
Special	Links display their local name and the location pointed to: foo->/usr/bar	

Exported/Internal File Format

Symbolic links contain a resource that serves as a symbolic placeholder. Symbolic link data is stored as a set of variables, one for each name and location, in the component.

Component Type: system#container

A system#container component type should be used whenever components must be grouped together and installed as a single component. The container component has an install, uninstall, and snapshot block. These blocks automatically call in to the child components. These calls enable a container to have child components added directly by using the Component Builder user interface, without having to edit the container component's XML.

Browsing

The container component type is a composite component and cannot be browsed.

Model to Install Difference

This comparison optionally takes a snapshot of all child components.

The default snapshot block captures snapshots of all child components with installMode="NESTED" by using the default snapshot blocks of each child.

Extended Control Procedures

The following procedures are defined for the system#container component type:

- default **install block** – Installs only those child components that have installMode="NESTED" set.
- markOnly **install block** – Calls the markOnly install block of all the nested child components, then marks this component as installed.
- default **uninstall block** – Uninstalls only those child components that have installMode="NESTED" set.
- markOnly **uninstall block** – Calls the markOnly uninstall block of all the nested child components, then marks this component as uninstalled.
- default **snapshot block** – Captures a snapshot of all child components if createSnapshot is set to true. Only child components that have installMode="NESTED" set are installed. By default, createSnapshot is set to false.

Snapshots are not always enabled. If the component is deployed as a nested component of another component, it should be deployed with createSnapshot="false" because the container component will capture a snapshot of this component after all of its contained components have been deployed. If the component is being deployed as a top-level component, you can enable snapshots by setting createSnapshot="true".

Snapshots are not captured when a component is contained in a container component. The deployment of the other contained components after this component might affect the installed state of this component. Thus, the snapshot is captured after the top-level installations have completed.

Component Type: untyped

Components of type untyped extend from no base component and have no default behavior. They are intended for advanced edit use when custom types or XML is being added to the component and none of the default behaviors of the built-in types are relevant.

◆ ◆ ◆ CHAPTER 4

Plans

A plan is used to perform operations on hosts that are managed by the provisioning system.

This chapter covers the following topics:

Managing Plans

You can use the Sun N1 Service Provisioning System software browser interface to manage plans. The following pages in the browser interface include information about how to view and manage plans:

- **Plans page.** List plans and add new plans to the list. You can also access other plan pages to view plan details.

  To *see* which plans are already checked in, choose Plans from the navigation menu.
- **Details page.** View detailed information about the plan, such as its attributes and values. This page also provides information and buttons that enable you to manage the plan.
  - To *see details* about a plan, go to the Plans page, click the name of the plan you want, and view details on the Details page.
  - To *delete* a plan, go to the plan's Details page and click Delete. Click Continue To Delete, or click Cancel if you do not want to delete the plan.

Note – You do not need to delete generated plans or direct-run component procedures. These plans are removed automatically from the system when the component from which they were generated is deleted.

- To *move* a plan to another folder, go to the plan's Details page and click Move. Specify the name of the folder in which to move the plan and click Move, or click Cancel if you do not want to move the plan.

  Note – If you plan to move a plan into a folder that does not yet exist, click New Folder in the Move Component window. This new folder inherits the permissions from its parent folder. After you click Create, the folder tree displays with the new folder selected. Click Move.

- **Advanced Edit page.** Change what a plan does by changing the plan's XML.

  To *edit* a plan, see "How to Edit a Plan" on page 42.

In addition to managing plans, you can create plans and run plans as described later in this chapter.

You can also use the CLI to work with plans. See "Summary of Plan CLI Commands" on page 51. For detailed information about the CLI commands, see the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

▼ How to Edit a Plan

1. **Go to the Details page of the plan you want, and click Advanced Edit.**
2. **Modify the XML or import new XML for the plan.**

   To import a plan from another system, do one of the following:

   - **Type the full path of the file, then click Replace.**
   - **Click Browse and locate the file, then click Replace.**
3. **Check in the plan.**

   - **Click Check In to save the plan with the same name.**

     The Plans page appears.
   - **Click Check In As to save the plan with a different name.**

     A new page appears that tells you that you are about to check in a plan. The version number increments.
4. **Click Continue To Check In.**

Creating Plans

You can create a generated plan, create a custom plan, or customize a generated plan.

The provisioning system can generate automatically a plan that consists of more than one *direct-run component procedures*. You can run this plan directly or save it for use as a template for more complex plans that you author in XML.

For deployments that involve the coordination of multiple components, multiple host sets, or both, write a plan and use the provisioning system XML schema to define operations. Such operations might include dependency checks and scripting that execute commands on an application console. Once the plan is written, you need to check it in to the plan repository.

How to Create Generated Plans

You create a generated plan from a component's Details page.

Before You Begin To create a generated plan, you must belong to a user group that has Create, Edit, and Delete permission on the folder that contains the component for which you want to generate a plan.

1. **From the navigation menu, choose Components.**
   The Components page lists the components that are already checked in.
2. **If necessary, change folders so that you can view the component that you plan to deploy.**
   a. **Click Change Folder.**
   b. **Specify the name of the folder that contains the component, and click Change to Selected Folder.**
      The Components page now lists the components in the specified folder.
3. **In the table listing components, find the row describing the component you would like to deploy, and click Details.**
   The component's Details page displays.
4. **In the Component Procedures table, select each procedure to include in the plan.**
5. **In the bottom row of the Component Procedures table, click Generate Plan With Checked Procedures.**
   The generated plan's Advanced Edit page appears.
6. **In the Plan field, type the new plan's name.**
7. **Click Check In.**

Note – You might see this message: `Warning - plan names and/or paths differ.` This message indicates that you specified a plan name that is different from the plan name specified in the XML model. The XML model is updated according to what you type in the plan name field.

8 Confirm the component check-in.

a. Verify that you specified the correct plan name.

b. If necessary, select the component's new version number.

c. Click Continue to Check In.

The new plan's Details page appears.

▼ How to Create a Custom Plan

You can create a plan by using a schema-validating editor, such as TurboXML, or on the Advanced Edit page of the browser interface.

This procedure describes how to import a plan that you have written by using an XML editor.

1 Write a plan.

See the XML schema elements described in Chapter 2, "Shared Schema Used by Components and Simple Plans," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide* and Chapter 4, "Plan Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

2 Launch the browser interface and go to the Plans page.

3 (Optional) Click Change Folder and specify the folder that will contain the new plan.

A window appears where you specify the name of the folder in which to create the component.

- **If the folder exists, select the name of the folder, and click Change to Selected Folder.**

  The Plans page displays the list of plans in the specified folder.

- **If the folder does not exist, create the folder.**

  a. Specify the parent folder, and click New Folder.

  The Create New Folder window appears.

  b. Type the name of the new folder, and click Create.

  The new folder inherits its permissions from its parent folder. After you click Create, the folder tree displays with the new folder selected.

Note – You cannot create multiple folder levels at one time. To create several nested folders, you must return to the Change Folder window and repeat steps A and B.

c. **After the folder is created, select the name of the folder, and click Change to Selected Folder.**

The Plans page displays showing the list of plans in the specified folder.

4 **Type the name and brief description for the plan that you want to create, and click Create.**

The Advanced Edit page for the plan appears.

The provisioning system creates an XML skeleton for the plan.

5 **In the Plan Definition field, do the following:**

- **Enter the XML for the plan.**
- **To import a plan that is stored on your local system, type the full path name of the file or click the Browse button. Then, click Replace.**

6 **Click Check In.**

A window appears that tells you that you are about to check in a plan. The plan is assigned Version 1.0.

7 **Click Continue To Check In.**

Running Plans

A plan must be checked in to the plan repository before it can be run. When you run a plan, it performs operations on the hosts that you specify.

A *preflight* is the simulated execution of a plan to a simulated UNIX® environment that finds and reports any errors or potential errors that might affect the deployment. A preflight always precedes a deployment, but you can run a preflight as a standalone operation.

You can also run a detailed preflight, which performs the following functions:

- Checks the availability and the connectivity of target hosts
- Confirms permissions
- Validates dependencies

▼ How to Run a Plan

Use this procedure to run a plan from a plan's Details page in the browser interface.

Before You Begin Ensure that you have the installation information that you need to be able to set the plan's variable settings.

To run a plan, you must belong to a user group that has `Allow` on `Host Set` permissions on the folder that contains the plan you want to run.

1 Go to the plan's Details page, and click Run.

The plan's Run page appears.

2 In the Plan Parameters area, select the variable settings for the component you want to deploy.

- **If the variable settings have been established for this component, choose the appropriate settings from the menu.**
- **If the settings are not available from the menu, click Select From List.**

  The Select Variable Settings From List window appears.

  - **If you want to use another component's variable settings, click Import Set from Component.**

    The Import Variable Settings window appears.

    a. **If necessary, go to the folder that contains the component with the variable settings you want to import.**

    b. **Select the component version.**

    Note – Variable settings can vary between component versions.

    c. **Click Import Variable Settings.**

    The imported variable settings appear in the table.

    d. **Click Select.**

  - **If you want to create new variable settings, click Create Set.**

    The Select Variable Settings From List window appears.

    a. **In the text field at the top of the table, type a name to use for the new variable settings.**

    b. **Select the component variable you want to change.**

    The table cell is highlighted and a text field appears.

    c. **Type a new value for the component variable.**

d. (Optional) Repeat the previous substeps b and c for each variable value you want to change.

e. After updating the variable settings values, click Save.

The new variable settings appear in the table.

f. Click Select.

3 Specify where to deploy the component.

- **To specify a host, type the host name in the Target Host field, or click Select From List and select a host.**
- **To select a target host set, choose one from the Target Host Set menu.**

4 In the Plan Variables area, configure the variables you want to set.

Whether there are variables to configure depends on the contents of the plan.

5 In the Options area, specify whether to perform a detailed preflight.

The detailed preflight attempts to simulate every step of the plan that will run on the remote agent. The detailed preflight tests the following functions:

- Resource installation
- `<execNative>` calls
- File transformations
- File deletion, when uninstalling resources

Running a detailed preflight increases the amount of time the preflight takes.

6 (Optional) To limit the number of hosts running at the same time, type the number of hosts on which to run a plan in the field provided.

When a plan is run on several hosts simultaneously, all hosts must complete a particular step before any host can begin the next step. If the number of hosts is too high, the network connection can time out between steps.

For example, the time interval between Host A finishing the first step and being able to run the second step might be too long. In this case, the length of the time interval causes the network connection to time out.

7 (Optional) To limit the overall running time of a plan or of native calls, type a number in the fields provided, then choose the time unit, such as minutes, from the menu.

By limiting the runtime of a plan or native call, you prevent a nonresponsive host from tying up the progress of a running plan.

If you set up a notification rule to alert you to plan failures, when the plan times out, you will be notified that the plan was unsuccessful.

For information about setting up notification rules, see "Configuring Email Notification" in *Sun N1 Service Provisioning System 5.2 System Administration Guide*.

8 Specify whether to deploy the component.

- **To run just the preflight test of the procedure, click Run Preflight Only.**
- **To run the preflight test and then the procedure itself, click Run Plan (Includes Preflight).**
  Note that the procedure is not run if the preflight fails.

▼ How to Deploy a Component by Using a Direct-Run Component Procedure

Most components include install, uninstall, and snapshot procedures. A component might also include control procedures that manage the installed component. For example, a control procedure might start or stop the application.

You might be able to use one of the direct-run component procedures that the provisioning system automatically generated when you checked in your component. Use this method for straightforward deployments that do not involve more than one component or synchronization between hosts.

1 From the navigation menu, choose Components.
The Components page lists the components that are already checked in.

2 If necessary, change folders so that you can view the component that you plan to deploy.

a. Click Change Folder.

b. Specify the name of the folder that contains the component, and click Change to Selected Folder.
The Components page now lists the components in the specified folder.

3 Click the name of the component that you want.
The Details page for that component appears.

4 In the Component Procedures table, determine which procedure to run, and click Run.
The provisioning system generates a plan, which is stored in the `/system/autogen` folder.

The Run page for the generated plan appears.

5 In the Plan Parameters area, select the variable settings for the component you want to deploy.

- **If the variable settings have been established for this component, choose the appropriate settings from the menu.**

- **If the settings are not available from the menu, click Select From List.**
  The Select Variable Settings From List window appears.

  - **If you want to use another component's variable settings, click Import Set from Component.**
    The Import Variable Settings window appears.

    a. **If necessary, go to the folder that contains the component with the variable settings you want to import.**

    b. **Select the component version.**

    Note – Variable settings can vary between component versions.

    c. **Click Import Variable Settings.**
    The imported variable settings appear in the table.

    d. **Click Select.**

  - **If you want to create new variable settings, click Create Set.**
    The Select Variable Settings From List window appears.

    a. **In the text field at the top of the table, type a name to use for the new variable settings.**

    b. **Select the component variable you want to change.**
    The table cell is highlighted and a text field appears.

    c. **Type a new value for the component variable.**

    d. **(Optional) Repeat the previous substeps b and c for each variable value you want to change.**

    e. **After updating the variable settings values, click Save.**
    The new variable settings appear in the table.

    f. **Click Select.**

6 **Specify where to deploy the component.**

- **To select a host, type the host name in the Target Host field or click Select From List and select a host.**

- **To select a target host set, choose one from the Target Host Set menu.**

7 In the Plan Variables area, configure the variables you want to set.

Whether there are variables to configure depends on the contents of the component procedure you select.

8 In the Options area, select whether to perform a detailed preflight.

The detailed preflight attempts to simulate every step of the plan that will run on the remote agent. The detailed preflight tests the following functions.

- Resource installation
- `<execNative>` calls
- File transformations
- File deletion, when uninstalling resources

Running a detailed preflight increases the amount of time the preflight takes.

9 (Optional) To limit the number of hosts running at the same time, type the number of hosts on which to run a plan in the field provided.

When a plan is run on several hosts simultaneously, all hosts must complete a particular step before any host can begin the next step. If the number of hosts is too high, the network connection can time out between steps.

For example, the time interval between Host A finishing the first step and being able to run the second step might be too long. In this case, the length of the time interval causes the network connection to time out.

10 (Optional) To limit the overall running time of a plan or of native calls, type a number in the fields provided, then choose the time unit, such as minutes, from the menu.

By limiting the runtime of a plan or native call, you prevent a nonresponsive host from tying up the progress of a running plan.

If you set up a notification rule to alert you to plan failures, when the plan times out, you will be notified that the plan was unsuccessful.

For information about setting up notification rules, see "Configuring Email Notification" in *Sun N1 Service Provisioning System 5.2 System Administration Guide*.

11 Specify whether to deploy the component.

- **To run just the preflight test of the procedure, click Run Preflight Only.**
- **To run the preflight test and then the procedure itself, click Run Plan (Includes Preflight).**

  Note that the procedure is not run if the preflight fails.

Summary of Plan CLI Commands

You can use the following `pdb` commands to manage plans:

- `pdb.p.ci` – Checks in a new version of a plan
- `pdb.p.co` – Checks out a plan in XML format
- `pdb.p.del` – Deletes a plan
- `pdb.p.genplan` – Generates and displays output for a plan in XML format
- `pdb.p.la` – Lists all versions of all plans
- `pdb.p.lo` – Shows detailed information about a plan
- `pdb.p.lv` – Lists all versions of a plan
- `pdb.p.mv`–Moves and/or renames a plan
- `pdb.p.sc` – Associates a plan with a set of categories
- `pdb.p.sh` – Shows or hides a plan

The following `pe` commands are associated with running plans:

- `pe.h.prep` – Prepares a set of hosts
- `pe.p.del` – Deletes the history of a completed plan run
- `pe.p.en` – Displays the output of an `<execNative>` step
- `pe.p.la` – Lists running and completed plans
- `pe.p.lo` – Lists detailed information about a running or completed plan
- `pe.p.lp` – Lists the subplans and targets that are associated with a plan
- `pe.p.run` – Runs a plan
- `pe.p.stop` – Stops a plan that is running
- `pe.pi.lo` – Lists the parameters that are used to run a plan

For more information about the provisioning system CLI commands, see the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

◆ ◆ ◆ CHAPTER 5

Session Variables

This chapter describes session variables and how to manage them.

This chapter covers the following topics:

Managing Session Variables

Session variables are global values associated with deployments that expand the flexibility of the provisioning system. Session variables can be created and saved for future use with any deployment or created and not saved for use with a single deployment.

You can use the Sun N1 Service Provisioning System software browser interface to manage session variables. The following pages of the browser interface include information about session variables:

- **Session variables page.** List session variables and add new ones to the list. You can also access other session variable pages to view details.

  To *see* which session variables are already checked in, choose Session Variables from the navigation menu.
- **Details page.** View detailed information about session variables, such as their attributes and values. This page also provides information and buttons that enable you to manage the session variables.

  To *see details* about a session variable, go to the Session Variables page, click the name of the session variable you want, and view the details on the Session Variables Details page.
- **Edit page.** Change session variable information.

  To *edit* a session variable, go to the Session Variables Details page and click Edit. On the Session Variables Edit page, modify the session variable. Click Save to save your changes, or click Cancel to discard the changes.

Summary of Session Variable CLI Commands

The following `udb` commands manage session variables:

- `udb.sv.add` – Adds a new session variable
- `udb.sv.del` – Deletes a session variable
- `udb.sv.fl` – Flushes all of a user's session variables
- `udb.sv.la` – Lists all session variables
- `udb.sv.lo` – Retrieves detailed information about a session variable
- `udb.sv.mod` – Modifies a session variable
- `udb.sv.re` – Reencrypts all of a user's session variables

For more information about the provisioning system CLI commands, see the *Sun N1 Service Provisioning System 5.2 Command-Line Interface Reference Manual*.

◆ ◆ ◆ CHAPTER 6

Configuration Generation

The Sun N1 Service Provisioning System software enables you to customize the deployment of any component by using *substitution variables* and *variable settings*.

This chapter covers the following topics:

Configuration Generation Overview

You can model an application as a component so that it can be customized to run in different environments.

For example, suppose that you use a database application to manage your online catalog business. Before you deploy the application to the production environment, you test it in a test environment. These two environments run the same application but are configured differently. When the application is deployed to the test environment, it uses a test database. When the application is deployed to the production environment, it uses the production database.

You can store configuration information about the application, such as which database to use, in configuration files. To support each environment, these files must be customized. These configuration files can include substitution variables that are replaced by variable setting values when you deploy the component.

Not only can you use substitution variables in configuration files, but also in plans and components. For example, you might use them to specify the directory in which to install the application. The provisioning system enables you to define and manage distinct variable settings for each deployment environment.

A substitution variable reference in a configuration file, plan, or component can obtain the following kinds of values:

- User-defined value
- Component-specific value, such as a name or label
- Target host-specific value, such as an IP address
- User-defined value that is specific to a host in the provisioning system
- Value of a session variable
- Value associated with a component that has already been installed

The provisioning system uses a *configuration generation engine* to replace substitution variable references with the appropriate variable setting values. This engine interacts with the host repository and component repository to resolve values any time that you run a plan to deploy a component.

The provisioning system has a repository for storing variable settings so you can reuse them at a later time. You can view the variable settings that have been used to install components and to run plans to determine the state of the variables. This repository is also used to determine the state of substitution variables when running a comparison with an installed component.

Adding Substitution Variables to Components

When you run a plan that deploys an application to more than one host, you can use configuration generation to automatically replace substitution variables with appropriate values for each host.

To do this, you add substitution variable definitions to your components. These can be used, for example, as a way to configure the directory into which an application is installed. Using the provisioning system, you can define and manage different variable settings for application deployments on each of your target hosts, as follows:

- Each version of a component can declare its own variable definitions.
- Each version of a component has its own variable settings (possibly imported from a previous version).
- Each component can be installed using any of its variable settings.

Substitution Variable Values

The values that the provisioning system substitutes when you install an application can be any of the following:

- Value that you specify
- Component-specific value, such as a name or a label
- Value that is associated with a previously installed component
- Value of a session variable
- Value that is specific to a target host, such as an IP address or a user-defined host variable

Generation Context

Variable substitution happens when a target step is run on a target host. The step can be in a plan or in a component being installed or already installed on the target host. If there is a state associated with the target host and component, it is used to determine the value of a particular substitution variable. The state can include the following:

- **Target component –** The component on which to perform operations
- **Target host –** The current host on which the target step is executing
- **Target variable settings source –** A collection of name-value pairs that override the default values defined in the component when it is being installed
- **Local variables –** Any variables in the target step itself, such as those that you have declared in the enclosing block or plan

Input Source

The variable substitution engine operates on any text-based input source in the form of a `String` or `Reader`. In practice, however, only these entities are used as input sources:

- Configuration-type resource files
- Input to CLI commands that perform variable substitution on demand
- Some of the component and plan attributes, such as the *defaultInstallPath* attribute of the `<installSteps>` element and the *command* attribute of the `<execNative>` step

  See Chapter 2, "Shared Schema Used by Components and Simple Plans," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*, Chapter 3, "Component Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*, and Chapter 4, "Plan Schema," in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

Substitution Variable Validation

Substitution variables are parsed and examined for syntax errors during a component, plan, or configuration template checkin.

The provisioning system performs context-sensitive syntax validation. For example, the expression `:[target:sys.hostName]` can be used in a component variable default value, but cannot be used in a step.

Note that the provisioning system does not perform semantic validation of entities that are referenced by substitution variables. For example, if a variable references a system service, the provisioning system does not confirm that the system service actually exists.

Types of Variables Available for Substitution

You can refer to the following types of variables from within components, plans, variable settings, and configurable resources:

- Simple substitution variables
- External component substitution variables
- Session substitution variables
- Target substitution variables

Simple Substitution Variable References

The input source can contain any number of simple substitution variable references. A simple substitution variable has the following syntax:

:[*variable*]

variable is one of the following:

- **Local variable name.** You can reference local variable names in plan steps and in steps that are in a component's install, uninstall, snapshot, and control service blocks.
- **Parameter name.** You can reference parameters in plan steps and in steps that are in a component's install, uninstall, snapshot, and control service blocks.
- **Predefined component variable name.** You can only use predefined component variable names in input sources that are associated with a component, but not with a plan.

  Predefined component variable names are always resolved relative to the actual component that is (or is being) installed, even if the variable reference occurs in a base component. From an accessibility standpoint, predefined component variables are treated as `PUBLIC` variables.

  See "Predefined Component Variable Names" on page 58 for a list of names.
- **Dynamic component variable name.** A component can reference both locally declared and inherited variables. The value that will be substituted for the reference is either the override value of the associated variable settings source or the variable's default value. The value that is used is expanded before it is substituted.

  A dynamic component variable name is declared or inherited by the component associated with the target component. In this case, the value to be substituted for the variable reference is either the associated override variable settings value or the default value defined by the component. The value of this variable is expanded before it is substituted.

Predefined Component Variable Names

These predefined component variable names refer to the associated component attributes of the generation context:

- `sys.name` – Component name
- `sys.version` – Version number of the component in the standard format *major.minor*

- `sys.description` – Component description
- `sys.label` – Label of the component
- `sys.softwareVendor` – Software vendor that wrote the application that is modeled by the component
- `sys.author` – Creator of the component
- `sys.path` – Path of the component

  The `sys.path` variable is like the *path* attribute of the component, but it includes a trailing slash (`/`). `sys.path` can be directly combined with the `sys.name` variable to create the full name of the component.

  ```
  :[sys.path]:[sys.name]
  ```

- `sys.rsrcInstallPath` – Location where the component resource is installed

  The `sys.rsrcInstallPath` variable is in the file format of the *remote agent* on which the component is, or is being, installed. You can only use this variable for simple components. Since the value of this variable might require other dynamic component variables to be computed, you cannot use `sys.rsrcInstallPath` in a component variable default value or variable settings override value. This variable also cannot be used as the value of the `<component>` *installPath* attribute or in the `<resourceRef>`/`<installSpec>` element attributes. These restrictions prevent the possibility of circular references.

- `sys.targetRefName` – Name of the associated component targeting host

  The `sys.targetRefName` variable is only defined for *targetable components*. The component targeting host is defined by the resolved value of the *hostName* attribute of the targetable component's `<targetRef>` element. It cannot appear as a simple substitution variable in a component variable value because component variables might be used to compute the `sys.targetRefName` value. This variable is most often used with the `<retarget>` step to install or manipulate components that are in a targetable component.

Predefined component variables are always resolved relative to the actual component that is, or is being, installed even if the variable reference occurs in a base component. Predefined component variables are treated as `PUBLIC` variables.

Escape Sequences for Substitution Variables

If you want to include `:[` in input source, you must escape it by using `:[[` instead. The configuration generator replaces occurrences of `:[[` with `:[`.

For example, to literally include the text `:[box]` without the provisioning system substituting a value for *box*, use `:[[box]`.

When the input source is an attribute of a component that is authored in XML, the value is subject to the following rules for XML-based escape sequences:

- To get the quote character (`""`), use the `&quot;` sequence.
- To get the apostrophe character (`'`), use the `&apos;` sequence.
- To get the less-than character (`<`), use the `&lt;` sequence.

- To get the greater-than character (>), use the `>` sequence.
- To get the ampersand character (&), use the `&` sequence.

These escape rules do not apply to override values input through the provisioning system browser interface, configuration resource files, or input to the configuration generation CLI commands.

Substitution Variable Expansion

When a substitution variable reference is replaced by a variable setting value, the value is recursively expanded before being substituted. This expansion is necessary because the value itself might contain references to simple substitution variables. To prevent the substitution of values, escape input sources with `:[[`.

You can only use host and external component substitution references in default or override substitution values, parameter default values, and local variable default values. Such references are not permitted in other input sources.

Simple Substitution Variable Reference Expansion

The syntax for a simple substitution variable reference in a variable setting value is the same as for input sources:

`:[`*varname*`]`

In this case, the variable name that is referenced must be a variable that has been declared before the variable that contains the reference. This restriction prevents circular references.

For derived components, variables that are inherited from the base component are first expanded in the order of declaration in the parent. Then, local non-override variables are expanded in the order of declaration. Local variables that override inherited variables are expanded in place of the inherited variable in the same order that the inherited variable would have been expanded. Therefore, override variables can only refer to other inherited and override variables that have already been declared. For example, base component A declares variables *x* and *y*. Then derived component B declares variables *z* and *y*. The order of evaluation of variables in component B is *x*, which is inherited from component A, then *y*, which is overridden by component B, and finally *z*, which is local to component B.

EXAMPLE 6–1 Using Simple Substitution Variables

The following table shows examples of substitution variables, as well as their expanded and contracted values.

Variable Name	Contracted Value	Expanded Value
`foo`	`silly`	`silly`
`bar`	`:[foo]`	`silly`

EXAMPLE 6–1 Using Simple Substitution Variables *(Continued)*

Variable Name	Contracted Value	Expanded Value
`baz`	`a :[foo] :[bar] example`	`a silly silly example`
`badFrob`	`:[frob]`	Error – forward reference
`frob`	`:[[foo]`	`:[foo]`
`compName`	`:[sys.name]`	Name of the target component
`badFoz`	`:[foz]`	Error – `foz` has not been declared

External Component Substitution Variable References

A variable setting value can include one or more external component substitution references. An external component substitution reference is one of the following:

- :[*primary-component*:*varname*]
- :[*primary-component*:*secondary-component-list*:*varname*]

Note that an external component substitution reference does not include any white space.

primary-component is one of the following:

- *explicit-external-component*
- *system-service-component*
- *system-type-component*
- *targetable-component*
- *secondary-component*

secondary-component is one of the following:

- *nested-component*
- *toplevel-component*
- *dependee-component*
- *container-component*

secondary-component-list is a sequence of one or more colon-separated *secondary-components*.

varname is either a predefined substitution variable name or a dynamic substitution variable name that is declared by the referenced component.

Explicit External Component Expansion

An *explicit-external-component* explicitly specifies a component that is expected to be installed on a particular host in a particular location. It has the component resolution semantics of the `<installedComponent>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

explicit-external-component has the following syntax:

component-target:*explicit-component-reference*

component-target specifies the host on which the specified component is installed and is one of the following:

- `component` – Specifies the current target host in the generation context
- `component(`*host-redirect*`)` – Refers to the host named by *host-redirect* (see "Host Redirects" on page 68)

explicit-component-reference is a reference to a component that is already installed on the target host that is specified by *component-target*. The *explicit-component-reference* minimally includes the installed component name, but can also include a version and an install path, as follows:

full-component-name
full-component-name#*version*
full-component-name@{*path*}
full-component-name#*version*@{*path*}

full-component-name is an absolute or relative reference to the specified component. A relative reference is expanded relative to the plan or component that contains *explicit-component-reference*.

path is the absolute install path name of the specified component, or a substitution variable reference that, when expanded, represents the absolute install path of the specified component. If *path* includes the } character, it must be escaped using the }} sequence.

System Service Component Expansion

A *system-service-component* specifies a system service component that is expected to be installed on the root *physical host* of the current target. It has the component resolution semantics of the `<systemService>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

system-service-component has the following syntax:

`systemService:`*system-service-name*

If the system service is defined by a plug-in, *system-service-name* must be prefixed with the name of the plug-in that defines it.

System Type Component Expansion

A *system-type-component* specifies the component that is derived from a specified component type that is most recently installed on a particular host in a particular location. It has the component resolution semantics of the `<systemType>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

system-type-component has the following syntax:

`systemType:`*component-type-name*
`systemType:`*component-type-name*`@{`*path*`}`
`systemType(`*host-redirect*`):`*component-type-name*
`systemType(`*host-redirect*`):`*component-type-name*`@{`*path*`}`

path is optional. If specified, *path* is the absolute install path name of the specified component or a substitution variable reference. The reference, when expanded, represents the absolute install path of the specified component. If *path* includes the `}` character, it must be escaped using the `}}` sequence.

host-redirect is also optional and indicates the host on which the component is installed. See "Host Redirects" on page 68.

If the component type is defined by a plug-in, *component-type-name* must be prefixed with the name of the plug-in that defines it.

`targetableComponent` Component Expansion

A *targetable-component* references variables that are associated with a targetable component and is similar to the `<targetableComponent>` targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

targetable-component has the following syntax:

- `targetableComponent`
- `targetableComponent(`*host-redirect*`)`

host-redirect is the name of a component targeting host. If unspecified, the host on which the plan is currently executing is used. These constructs resolve to the targetable component that is associated with a particular component targeting host. See "Host Redirects" on page 68.

Secondary Component Expansion

Nested Component Expansion

A *nested-component* specifies the component that is referenced by a nested component reference that is declared by the current component. It has the component resolution semantics of the `<nestedRef>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

nested-component has the following syntax:

`nestedRef:`*component-reference-name*

component-reference-name is the name of a `<componentRef>` element where `installMode=NESTED` for the current component.

nested-component can only be used when the current component is a composite component. It can only be used as a *primary-component* when the external component substitution reference appears in

a composite component. When used as a *primary-component*, it cannot be used as the default value of a component variable or override variable setting. This is because the referenced component will not yet have been installed when the variable is evaluated.

Top-Level Component Expansion

A *toplevel-component* specifies the component that is referenced by a top-level component reference that is declared by the current component. It has the component resolution semantics of the `<toplevelRef>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

toplevel-component has the following syntax:

`toplevelRef:`*component-reference-name*
`toplevelRef:`*component-reference-name*`@{`*path*`}`
`toplevelRef(`*host-redirect*`):`*component-reference-name*
`toplevelRef(`*host-redirect*`):`*component-reference-name*`@{`*path*`}`

component-reference-name is the name of a `<componentRef>` element where `installMode=TOPLEVEL` for the current component.

path is the absolute install path name of the specified component, or a substitution variable reference. The reference, when expanded, represents the absolute install path of the specified component. If *path* includes the `}` character, it must be escaped using the `}}` sequence.

host-redirect is the host on which the referenced component is expected to be installed. The syntax for *host-redirect* is described in "Host Redirects" on page 68.

toplevel-component can only be used when the current component is a composite component. It can only be used as a *primary-component* when the external component substitution reference appears in a composite component. When used as a *primary-component*, it cannot be used as the default value of a component variable or an override variable setting. This is because the referenced component has already been installed.

Dependee Component Expansion

A *dependee-component* specifies the component on which the current component depends as a result of a dependency created by a `<createDependency>` step. *dependee-component* has the component resolution semantics of the `<dependee>` installed component targeter. See "Installed Component Targeters" in *Sun N1 Service Provisioning System 5.2 XML Schema Reference Guide*.

dependee-component has the following syntax:

`dependee:`*dependency-name*

dependency-name is the name of a dependency that is created by the current component.

dependee-component can only be used as a *primary-component* when the external component substitution reference appears in a component. When used as a *primary-component*, it cannot be used as the default value of a component variable or an override variable setting. This is because the dependency will not yet have been created when the variable is evaluated.

Container Component Expansion

A *container-component* specifies the component that contains the current component as a nested reference.

container-component has the following syntax:

```
container
```

container-component can only be used as a *primary-component* when the external component substitution reference appears in a component that has been installed as a nested component by another "container" component.

Resolving External Component Substitution References

The value of an external component substitution reference is computed first by resolving the component that is referenced by *primary-component*. If *secondary-component* is used as a *primary-component*, the component that contains the external component substitution reference serves as the initial current component that is used when resolving the *primary-component*. The component that is resolved by *primary-component* becomes the new current component. Then, each *secondary-component* in *secondary-component-list* is resolved by using the last resolved component as the current component. Finally, the provisioning system looks up and returns the variable, *varname*, in the last resolved component.

The value of an external component substitution reference is computed based on the value of the specified variable in the referenced component at the time the component was installed. You cannot refer to a variable that is not declared by the installed component or to a component that is not installed on the target host. You also cannot refer to a variable that is not accessible to the component or plan that declares the external component substitution reference. The variable is accessible only if it is declared with an accessible access mode, the declaring component is accessible, and each primary and secondary component that is traversed to get to that component is accessible.

External component substitution references are computed at the time that they are encountered during a plan run, not at the start of the run. Thus, an actual installed component that is being referenced might change based on the steps executed earlier in the plan. Furthermore, references to components that are installed on a host other than the current target host might be affected by other plans that are running concurrently on the other host. For predictable results, only refer to hosts that are included in the target set of the current plan run because they are guaranteed to be locked. Also, synchronize any `<install>` or `<uninstall>` operations on an externally referenced host in the plan by using retargeting or series execution mode.

EXAMPLE 6–2 Using External Component Substitution Variable References

- The following are examples of external component substitution variable references that use explicit-external components:

```
:[component:installApache:sys.label]
:[component:jdk#1.3:classpath]
:[component:webApp#2.4@{/usr/local}:bannerColor]
```

EXAMPLE 6–2 Using External Component Substitution Variable References *(Continued)*

- In the following example, the provisioning system resolves the component that is installed on the root physical host of the current target host:

  ```
  :[component(/):IIS global settings:install_path]
  ```

- In this example, `webAppPath` is resolved as the name of a simple substitution variable that is set in the associated component of the generation context. It is not a variable that is defined in the externally referenced component.

  ```
  :[component:webApp@{:[webAppPath]}:bannerColor]
  ```

- The following examples show external component substitution variable references that use *targetable-component*:

  ```
  :[targetableComponent:hostName]
  :[targetableComponent(host1):port]
  ```

- The following are examples of external component substitution variable references that use *nested-component* and *toplevel-component*:

  ```
  :[nestedRef:ref1:sys.label]
  :[toplevelRef:ref2:sys.name]
  :[toplevelRef(/):ref3@{/usr/local}:var1]
  ```

- The following are examples of external component substitution variable references that use *system-type-component* and *system-service-component*:

  ```
  :[systemService:com.sun.windows#Windows SS:classPath]
  :[systemType:com.sun.weblogic#WL Target Type:serverName]
  :[systemType(..):MyType@{/tmp}:var1]
  ```

- The following is an example of an external component substitution variable reference that uses *dependee-component*:

  ```
  :[dependee:app2domain:domainName]
  ```

- The following is an example of an external component substitution variable reference that uses *container-component*:

  ```
  :[container:sys.installPath]
  ```

- The following is an example of a chained component reference. This reference assumes that the current target component is installed as a nested component of another container component. The `container:` part resolves to the component that contains the current component, which for this example is called X. The `nestedRef:ref1` part resolves to the component that is referenced by component X by using its component reference named `ref1`, which is component Y. The `toplevelRef:ref1` part resolves to the component that component Y refers to by using its component reference `ref1`, which is component Z. Finally, the provisioning system resolves and uses the value of the *label* attribute that is defined by component Z.

  ```
  :[container:nestedRef:ref1:toplevelRef:ref1:sys.label]
  ```

Session Substitution Variable References

Session variables enable you to enter data, such as your WebLogic credentials, one time per user session. You can also securely save the contents of these variables and not have to re-enter data each time you log in. Session variables are integrated at the modeling level by using substitution variables.

A session substitution variable reference has the following syntax:

`:[session:`*varname*`]`

varname is the name of a session variable.

A session substitution variable reference resolves to the value of the session variable in the current user's session. An attempt to resolve a session variable that is not defined in the current user's session results in an error. When a failure to generate a session variable is encountered, an attempt is made to add the session variable, with an empty value, to the user's current session variable set.

A session variable can be used anywhere that a simple substitution variable reference can be used. However, a session variable cannot be used in the value of another session variable.

Predefined Session Variables

`sys:sessionID` is a predefined session variable that resolves to the ID of the current user session. The system `sessionID` variable enables you to write steps that call back into the *master server* through the CLI by using the same credentials as the user who ran the plan. The system `sessionID` variable can only be resolved in the context of certain hosts, per the `config.allowSessionIDOnHosts` configuration variable. Thus, the system `sessionID` variable can only be resolved if the target host of the current generation context is included within the hosts named by the `config.allowSessionIDOnHosts` configuration variable.

When configuring a CLI for use in callbacks during plan execution, a Remote Agent and CLI Client must be installed on the same server. For maximum security, you should install the CLI Client and the Remote Agent on the same server as the Master Server. In this scenario, plans and components that want to make a callback through the CLI would use a `<retarget>` step to redirect execution to the single host that contains the CLI. In addition, the system `sessionID` variable would be resolved after the `<retarget>` step, and the `config.allowSessionIDOnHosts` configuration variable would name only the master server host.

Target Substitution Variable References

You can use target substitution variables to obtain values directly from a particular host.

Target substitution variable references use the following syntax:

`:[target:`*varname*`]`
`:[target(`*host-redirect*`):`*varname*`]`

A variable name that begins with `target` refers to the logical host on which a plan is currently being run.

varname represents an attribute value that is specified in the target host's definition for the application that you install. Two types of host variables can be referenced: predefined and dynamic. See "Predefined Host Variable Names" on page 69 and "Dynamic Host Variable Names" on page 70.

If you also specify *host-redirect*, the value for *varname* is retrieved from that particular host. That host can be a host other than the host on which the plan is currently being run. You can also specify another substitution variable for *hostname* that, when expanded, resolves to the name of a host. The syntax for *host-redirect* is described in "Host Redirects" on page 68.

Predefined Host Substitution Shorthands

You can use these predefined host substitution shorthands anywhere that a host substitution reference can be used:

- `:[/]` – Expands to the operating system-specific file separator, which is based on the root physical host of the current target host
  - UNIX systems – `:[/]` expands to `/`
  - Microsoft Windows systems – `:[/]` expands to `\`
- `:[:]` – Expands to the operating system-specific path separator, which is based on the root physical host of the current target host
  - UNIX systems – `:[:]` expands to `:`
  - Microsoft Windows systems – `:[:]` expands to `;`

Host Redirects

Target host substitution variable references and external component substitution variable references can optionally include a host redirect. This means that the host on which the variable or component lookup occurs is the specified host rather than the current target. *host-redirect* has one of the following forms:

- *parent-ref*
- *hostname*
- *hostname*/*parent-ref*

hostname is either the name of a host or a substitution variable reference that, when expanded, resolves to the name of a host.

parent-ref is one of the following:

- `/` – The root parent host of the specified host, or the current target host if *hostname* is omitted
- **A series of one or more slash-separated `..` indicators –** The *n*th parent of the specified host, or current target host if *hostname* is omitted, where *n* is the number of `..` indicators that appear

The *parent-ref* operators are used primarily for *virtual hosts*, but they can also be used for physical hosts. Applying either operator to a physical host is a no-op.

EXAMPLE 6–3 Using Target Substitution Variable References

- This example shows how to look up variable `var1` on host `myHost`:

  ```
  :[target(myHost):var1]
  ```

- This example shows how to look up variable `var1` on the host specified by substitution variable `hostName`:

  ```
  :[target(:[hostName]):var1]
  ```

- This example shows how to look up variable `var1` on the logical host on which the plan is currently running:

  ```
  :[target:var1]
  ```

Predefined Host Variable Names

These predefined host variable names refer to the associated attribute of the referenced host:

- `sys.hostName` – Target host name
- `sys.description` – Target host description
- `sys.hostType` – Host type of the target host
- `sys.ipAddress` – IP address of the remote agent on the target host
- `sys.portNumber` – Port number of the remote agent on the target host
- `sys.raHomeDir` – Absolute path of the remote agent home directory on the target host
- `sys.raDataDir` – Absolute path of the remote agent data directory on the target host
- `sys.raTmpDir` – Absolute path of the remote agent temporary directory on the target host
- `sys.raConfigDir` – Absolute path of the remote agent configuration directory on the target host
- `sys.OSArch` – Architecture of the target host, for example, `sparc`
- `sys.OSName` – Name of the operating system that runs on the target host
- `sys.OSVersion` – Version of the operating system that runs on the target host

EXAMPLE 6–4 Using Host Substitution Variable References

The following are examples of host substitution variable references and their values:

- `:[target:sys.ipAddress]` – IP address of the current host
- `:[target:var1]` – Dynamic variable `var1` of the current host
- `:[target(host1):var1]` – Dynamic variable `var1` of host `host1`
- `:[target(:[hostNameVar]):var1]` – Dynamic variable `var1` of the host whose name is the value of the `hostNameVar` substitution reference
- `:[target(/):var1]` – Dynamic variable `var1` of the root parent host of the current host
- `:[target(host1//):var1]` – Dynamic variable `var1` of the root parent host of the host named `host1`

EXAMPLE 6–4 Using Host Substitution Variable References *(Continued)*

- `:[target(../..):var1]` – Dynamic variable `var1` of the second parent host of the current host
- `:[target(host1/../..):var1]` – Dynamic variable `var1` of the second parent host of the host named `host1`

Dynamic Host Variable Names

A dynamic host variable name is any attribute name that is declared by the *host type* of the referenced host. In this case, the value to be substituted for the variable reference is the corresponding value of the attribute that is defined by the referenced host. The value must be expanded prior to substitution, if applicable.

Using Substitution Variables

You can use simple substitution variables in any input source, including configuration files. You can use external component and target substitution variables in a number of places, including `<varList>`/`<var>` default values of the `<installSteps>`, `<uninstallSteps>`, `<controlService>`, and `<executionPlan>` elements.

Simple substitution variable references of the form `:[`*varname*`]` can be used in any input source, including configuration files and configurable component attributes. However, host substitution references (`:[target:`*varname*`]`) and external component substitution references (`:[component:`*compRef*`:`*varname*`]`) can only be used in variable setting values.

Variable setting values include the *default* attribute of the `<var>` element and values in the variable setting overrides, but not configuration files or other configurable component attributes. This limitation ensures that all variables used in a component and in the configuration files that it references are explicitly declared and validated by the `<varlist>` element of the component.

Following are all substitution variables that can be used in particular input sources. The elements and attributes listed are the input source, and the sublist contains the types of variables that are permitted.

- `<component>`/`<varList>`/`<var>`/ *default* attribute:

  Variable settings override value:

  - Component substitution references, excluding `sys.rsrcInstallPath` and `sys.targetRefName`
  - Host substitution references
  - External component substitution references, excluding *nested-component* and *dependee-component* as primary components
  - Session variable references
- `<component>`/`<installList>`/`<installSteps>`/`<paramList>`/`<param>`/ *default* attribute:

`<component>/<uninstallList>/<uninstallSteps>/ <paramList>/<param>/`*default* attribute:

`<component>/<controlList>/<control>/<paramList>/<param>/`*default* attribute:

`<component>/<snapshotList>/<snapshot>/<paramList>/<param>/` *default* attribute:

- Component substitution references
- Host substitution references
- External component substitution references
- Session variable references

- `<component>/<installList>/<installSteps>/<varList>/<var>/` *default* attribute:

  `<component>/<uninstallList>/<uninstallSteps>/<varList>/<var>/` *default* attribute:

  `<component>/<controlList>/<control>/<varList>/<var>/`*default* attribute:

  `<component>/<snapshotList>/<snapshot>/<varlList>/<var>/`*default* attribute:

  - Local variable substitution references that have been previously declared
  - Parameter substitution references
  - Component substitution references
  - Host substitution references
  - External component substitution references
  - Session variable references

- `<component>/<installList>/<installSteps>`/child steps:

  `<component>/<uninstallList>/<uninstallSteps>`/child steps:

  `<component>/<controlList>/<control>`/child steps:

  `<component>/<snapshotList>/<snapshot>/<prepare>`/child steps:

  `<component>/<snapshotList>/<snapshot>/<capture>`/child steps:

  `<component>/<snapshotList>/<snapshot>/<cleanup>`/child steps:

  - Local variable substitution references
  - Parameter substitution references
  - Component substitution references

- `<component>/<resourceRef>/<installSpec>`:

  `<component>/`*installPath* attribute:

  - Component substitution references, excluding `sys.rsrcInstallPath`

- `<component>/<diff>/<ignore>`:

  `<component>/<componentRef>/<argList>`:

  Configuration files:

  - Component substitution references

- `<component>/<targetRef>`:

  `<component>/<targetRef>/<agent>`:

  - Component substitution references, excluding `sys.targetRefName` and `sys.rsrcInstallPath`

- <executionPlan>/<paramList>/<param>/ *default* attribute:
  - Session variable references
- <executionPlan>/<varList>/<var>/ *default* attribute:

  <executionPlan>/*/<inlineSubplan>/<varList>/<var>/ *default* attribute:
  - Session variable references
  - Local variable substitution references that have been previously declared
  - Unhidden local variable substitution references of enclosing plans
  - Unhidden parameter substitution references
  - Host substitution references for simple plans and subplans only
  - External component substitution references for simple plans and subplans only, but excluding container, nested, top-level, and dependee as primary component
- <executionPlan>/*/child steps:
  - Local variable substitution references
  - Unhidden local variable substitution references of enclosing plans
  - Unhidden parameter substitution references
- Host attributes
  - Session variable references

The following *configuration* attributes of a component can include substitution variable references.

Parent Element	Substitutable Attributes
<capture>/<addFile>	*displayName*, *path*
<component>	*installPath*
<diff>/<ignore>	*path*
<installSpec>	*name*, *path*, *permissions*, *user*, *group*, *deployMode*, *diffDeploy*
<paramList>/<param>	*default*
<targetRef>	*hostName*
<targetRef>/<agent>	*connection*, *ipAddr*, *port*, *params*
<varList>/<var>	*default*

The following *step* attributes that are in a component or a plan can include substitution variable references.

Parent Element	Substitutable Attributes
<argList>	Attributes of the <argList> element, which are free form
<execJava>	*className*, *classPath*
<execNative>	*dir*, *userToRunAs*
<execNative>/<ouputFile>	*name*
<execNative>/<errorFile>	*name*
<execNative>/<env>	*name*, *value*
<execNative>/<exec>	*cmd*
<execNative>/<exec>/<arg>	*value*
<execNative>/<shell>	*cmd*, <body>
<execNative>/<successCriteria>	*outputMatches*, *errorMatches*
<execNative>/<inputFile>	*name*
<execNative>/<inputText>	<body>
<if>/<condition>/<equals>	*value1*, *value2*
<if>/<condition>/<istrue>	*value*
<if>/<condition>/<matches>	*value*, *pattern*
<processTest>	*processNamePattern*, *user*
<raise>	*message*
<retarget>	*host*
<retarget>/<varList>/<var>	*default*
<sendCustomEvent>	*message*
<transform>	*input*, *output*
<transform>/<source>	*name*
<transform>/<stylesheet>	<body>
<transform>/<subst>	*match*, *replace*
<urlTest>	*url*, *pattern*

The following plan attributes can include substitution variable references.

Parent Element	Substitutable Attribute
<paramList>/<param>	*default*
<varList>/<var>	*default*

The following attributes of installed component targeters can include substitution variable references.

Parent Element	Substitutable Attributes
<installedComponent>	*installPath*, *host*
<systemType>	*installPath*, *host*
<targetableComponent>	*host*
<toplevelRef>	*installPath*, *host*

The following attributes of repository component targeters can include substitution variable references.

Parent Element	Substitutable Attribute
<component>	*host*
<toplevelRef>	*host*

EXAMPLE 6–5 Using Substitution Variables

The following example lists the variables that are defined in a component for the Apache web server:

```
<varList>
    <var name="domainname" default=":[target:domainname]"/>
    <var name="name" default="apache"/>
    <var name="installPath" default="/opt/apache"/>
    <var name="execNativeShutdown"
       default=":[installPath]/bin/apachectlstop"/>
    <var name="execNativeStartUp"
       default=":[installPath]/bin/apachectlstart"/>
</varList>
```

This component specifies the following:

- The domain name must be dynamically retrieved from the target host.
- The default value of *name* is apache.
- The variable called installPath has a value of /opt/apache.
- The value of installPath is used in the definition of two <execNative> variables.

EXAMPLE 6–5 Using Substitution Variables *(Continued)*

Although you cannot use a target substitution variable in a configuration file, you can reference a host-specific value from a configuration file. You can do this because the value of the variable that you reference can be computed as a host-specific value. For example, you might have the following variable defined in the `<varList>` section of the component:

```
<var name="box" value=":[target:room]">
```

You can then reference `:[box]` from a configuration file. When `:[box]` is substituted, it is substituted with the value of the `room` variable that is defined by the target host.

◆ ◆ ◆ APPENDIX A

Variable Substitution Grammar

Only a *session-variable-substitution-reference* is permitted to occur in host attribute values. For the following grammar, a *simple-substitution-variable-reference* can be used in configuration resource files and certain component element attributes. *complex-substitution-variable-reference* can be used in substitution variable default and override values.

In this grammar, `\p{N}` represents all Unicode numbers, and `\p{L}` represents all Unicode letters.

simple-substitution-variable-reference:
    local-component-substitution-reference

complex-substitution-variable-reference:
    simple-substitution-variable-reference
    host-substitution-reference
    external-component-substitution-reference
    session-variable-substitution-reference

local-component-substitution-reference:
    `:[`*component-variable-name*`]`

component-variable-name:
    predefined-component-variable-name
    dynamic-component-variable-name

predefined-component-variable-name:
    `sys.name`
    `sys.version`
    `sys.description`
    `sys.label`
    `sys.softwareVendor`
    `sys.author`
    `sys.path`
    `sys.rsrcInstallPath`
    `sys.targetRefName`

dynamic-component-variable-name:
    identifier

host-substitution-reference:
    `:[target:`*host-variable-name*`]`
    `:[target(`*host-reference*`):`*host-variable-name*`]`
    `:[:]`
    `:[/]`

host-reference:
    root-host-selector
    parent-host-selector
    host-selector / *root-host-selector*
    host-selector / *parent-host-selector*

host-selector:
    host-name
    complex-substitution-variable-reference

root-host-selector:
    /

parent-host-selector:
    `..(/..)*`

host-variable-name:
    predefined-host-variable-name
    dynamic-host-variable-name

predefined-host-variable-name:
    `sys.hostName`
    `sys.description`
    `sys.hostType`
    `sys.ipAddress`
    `sys.portNumber`
    `sys.raHomeDir`
    `sys.raDataDir`
    `sys.raConfigDir`
    `sys.raTmpDir`
    `sys.OSName`
    `sys.OSVersion`
    `sys.OSArch`

dynamic-host-variable-name:
    identifier

session-variable-substitution-reference:
    `:[session:`*session-variable-name*`]`

session-variable-name:
    predefined-session-variable-name
    dynamic-session-variable-name

predefined-session-variable-name:
    `sys:sessionID`

external-component-substitution-reference:
    `:[`*component-reference-list*`:`*component-variable-name*`]`

component-reference-list:
    primary-component-reference
    primary-component-reference`:`*secondary-component-reference-list*

primary-component-reference:
    explicit-external-component-reference
    system-service-component-reference
    system-type-component-reference
    secondary-component-reference
    targetable-component-reference

secondary-component-reference:
    nested-component-reference
    toplevel-component-reference
    dependee-component-reference
    container-component-reference

secondary-component-reference-list:
    secondary-component
    secondary-component`:`*secondary-component-reference-list*

explicit-external-component-reference:
    `component:`*explicit-component-reference*
    `component(`*host-reference*`):`*explicit-component-reference*

explicit-component-reference:
    full-component-name
    full-component-name`#`*component-version*
    full-component-name`@{`*install-path*`}`
    full-component-name`#`*component-version*`@{`*install-path*`}`

full-component-name:
    path-reference component-name
    component-name

path-reference:
    `/`

/ *relative-path-reference* /
relative-path-reference /

relative-path-reference:
.
..
path-part
. / *relative-path-reference*
.. / *relative-path-reference*
path-part / *relative-path-reference*

system-service-component-reference:
`systemService:`*system-name*

system-type-component-reference:
`systemType:`*system-name*
`systemType:`*system-name*`@{`*install-path*`}`
`systemType(`*host-reference*`):`*system-name*
`systemType(`*host-reference*`):`*system-name*`@{`*install-path*`}`

targetable-component-reference:
`targetableComponent`
`targetableComponent(`*host-reference*`)`

nested-component-reference:
`nestedRef:`*component-reference-name*

toplevel-component-reference:
`toplevelRef:`*component-reference-name*
`toplevelRef:`*component-reference-name*`@{`*install-path*`}`
`toplevelRef(`*host-reference*`):`*component-reference-name*
`toplevelRef(`*host-reference*`):`*component-reference-name*`@{`*install-path*`}`

dependee-component-reference:
`dependee:`*dependency-name*

container-component-reference:
`container`

dependency-name:
component-reference-name:
identifier

dynamic-session-variable-name:
host-name:
`[\p{L}_][\p{N}\p{L}-_. ]*`

path-part:

component-name:
```
[\p{N}\p{L}-_. ]+    (except "." and "..")
```

system-name:
system-name-identifier
plugin-name#system-name-identifer

plugin-name:
identifier
identifier.*plugin-name*

system-name-identifier:
```
[\p{L}_][\p{N}\p{L}-_. +]*
```

component-version:
```
[1-9][0-9]*.[0-9]+
```

install-path:
complex-substitution-variable-reference
path-literal

path-liternal:
```
(([^}])|(}}))+
```

identifier:
```
[\p{L}_][\p{N}\p{L}_]*
```

Glossary

abstract component A component that serves only as a base component for other components to extend. An abstract component cannot be installed and only an abstract component is permitted to declare abstract child elements.

call compatibility A compatibility type for system service components. This compatibility is also called API compatibility or interface compatibility.

category A general class in which you can group objects that are stored in multiple folders.

child component A component that is referenced by a container component. Also called *contained component*.

See also *container component*.

comparison A feature that searches for and identifies differences between hosts and component models. The provisioning system supports these three types of comparisons:

- **Model to model –** Examines the deployment repository and history that is stored on the master server for two hosts and reports any differences
- **Model to install –** Compares what the master server reports is installed on a host to what is actually on the host and reports any differences
- **Install to install –** Examines the contents of two hosts' file systems and reports any differences

component A logical grouping of source information that defines an application. A component also includes a set of instructions that specifies how to manage the source information.

The XML representation of a component includes the following:

- List of resources used by the application
- Installation steps
- Uninstallation steps
- Dependencies

component compatibility A situation where a component can be safely replaced by another. The provisioning system supports two kinds of component compatibility: call compatibility and install compatibility.

component inheritance The means by which a component obtains attributes and behavior from another component. When you create a component, it inherits any variables, snapshots, and procedures from the associated component type.

component procedure A program in a component that controls deployment of the component, such as installation, uninstallation, management, and capturing snapshots. Management procedures are defined in the control block.

component repository A location on the master server where components and their resources are checked in.

component type A special kind of component that encapsulates behavior that can be reused by other components. A component can inherit the behavior of a component type by extending from it.

component variable A user-definable name-value pair that is used to make parts of a component accessible and configurable by objects that are external to the component.

composite component A component that contains only references to other components, both simple and composite. A composite component cannot contain any resources.

composite plan A plan that is composed solely of subplans, which can be simple or composite subplans. A composite plan is not directly targeted, as each subplan can run on a different set of targets.

configuration generation engine A software engine on the master server that replaces substitution variable references with the appropriate variable setting values. The engine interacts with the host repository and component repository to resolve values any time that you run a plan to deploy a component.

contained component A component that is referenced by other components.

container component A component that contains references to other components.

control A procedure defined by a component that can be used to manage the deployed applications. For example, a control might be used to start or stop an application. Also called *control service*.

deployment Using a plan or component procedure to act on a component. The component's lifecycle includes installation, uninstallation, and application management.

direct-run procedure A component procedure that can be run directly from the component by using the browser interface.

downstream In the Sun N1 Service Provisioning System software network hierarchy, the server that is further from the master server. For example, the master server connects downstream to a local distributor. Any remote agents connected to the local distributor are downstream from the local distributor.

`execNative` **call** An optional call out to custom scripts from the XML of a plan or component.

execution plan See *plan*.

extend To base a component on a component type so that the component inherits variables and procedures that are defined by the component type. The component can override variable values and procedure definitions defined by its associated component type.

final component A component that cannot be extended by another component.

folder Directory-like containers that enable you to apply permissions to and organize components, plans, and subfolders.

gold server A reference server that contains files, directories, and other resources that make up an application and that checks in these resources to the master server.

host A server that is managed by the provisioning system.

host search A query run on the host repository that yields a list of hosts whose attributes match those specified by the query. For example, you can use host searches to create a list of hosts that have the same host type, that run the same applications, and that are configured with the same subnet masks.

host set A user-defined, logical grouping of hosts that share one or more common attributes, such as physical location or functional group. Use a host set to quickly and easily update applications on all hosts in the set. You can also use a host set to perform model-to-install comparisons between two hosts.

host type A base class of servers that is bound by a set of common attributes, all of which are user-defined. You can use host types to categorize hosts into logical groupings and to facilitate host searches.

install compatibility A compatibility type for component types. This compatibility is also called structural compatibility.

Java Runtime Environment (JRE) A subset of the Java Development Kit (JDK®) for users and developers who want to redistribute the runtime environment. The Java runtime environment consists of the Java virtual machine (JVM), the Java core classes, and supporting files.

Java Virtual Machine (JVM) The part of the Java runtime environment (JRE) responsible for interpreting bytecodes.

Jython An implementation of the high-level, dynamic, object-oriented language, Python, seamlessly integrated with the Java platform. The predecessor to Jython, JPython, is certified as 100% pure Java.

label A means of marking a component version beyond the provisioning system version number. For example, a component version number describes the version of the component. A label can describe the version of the application that the component represents.

local distributor The application that is installed on a server. The Local Distributor application acts as a link between other servers in the provisioning system in the following ways:

- master server to remote agents

- master server to other local distributors
- local distributor to remote agents

Local distributors maximize bandwidth efficiency and speed, and can also provide secure network connections for navigating restricted environments.

master server The application that is installed on a server that manages the provisioning system. The Master Server application can connect to any of the data center environments managed by the provisioning system. The master server provides centralized data storage, data processing, and user interfaces.

modeling To create components and plans that represent an application that you want to deploy with the provisioning system.

nested component A contained component that, when installed, can provide its services only to its container component. A nested contained component defines a finer-grained unit of functionality required by the container component, but is not otherwise useful to other components.

network protocol A way to transmit data between devices on a network. The Sun N1 Service Provisioning System software uses TCP/IP, SSH, and SSL.

notification email An email sent by the provisioning system to advise that a system, administrative, or custom event has occurred. The system administrator specifies the rules used to determine when notification emails are sent and the email addresses to which the email is sent.

notification rule The criteria used by the provisioning system to determine whether an email notification is sent. The system administrator defines the criteria that that is used to determine when an email notification is sent.

parent component A component that contains references to other components. Also called *container component*.

See also *contained component*.

physical host A physical server that is connected to the network. Within the provisioning system, a physical host can act as a remote agent or a local distributor.

plan A sequence of instructions that is used to manipulate one or more components. A plan can also be a sequence of other plans, which enables common instruction sequences to be shared between one or more plans.

plan executor The software engine on the master server that runs preflights and deployments.

preflight The simulated execution of a plan to a simulated UNIX environment that finds and reports any errors or potential errors that might affect the deployment. A preflight always precedes a deployment, but you can run a preflight as a standalone operation.

procedure See *component procedure*.

provisioning system The software applications that, when installed on servers, form the Sun N1 Service Provisioning System software.

remote agent The application that is installed on any server in the provisioning system to which components are deployed. The Remote Agent application manages tasks, such as installing software, controlling services, and collecting information to deliver to the master server.

resource A file that is deployed to a host when a plan is executed. The file might be a directory, a symbolic link, or another kind of file.

resource descriptor file An XML file that specifies the owner, group, and permission settings to use for the files and directories that comprise the resource of a simple component. By using a resource descriptor file, you can override the permissions that are determined at component check-in time.

server A computer that manages resources and supplies services to a client. In the Sun N1 Service Provisioning System software, a server is a computer on which one of the provisioning system applications has been installed.

session A period of time that is initiated when you log in. A session persists until you log out or inactivity causes the session to expire. Logically, a session represents the authenticated credentials of a particular user. A session is used to identify the user throughout a series of related requests without reauthentication.

session variable A variable that is associated with a user session. The user can change session variable values for each login session. Session variable values can also be securely saved for reuse in subsequent sessions.

simple component A component that contains a single resource. A simple component cannot contain references to other components.

simple plan A sequential list of steps that are executed on a particular set of target servers. A simple plan does not contain or call other plans.

snapshot A capture of the resources that are stored on a host during a deployment. The snapshot is used when performing comparisons between a host and its model on the master server (model-to-install).

step An instruction that can be part of a plan or a component.

substitution variable A variable that appears in plans, components, or configuration files that is substituted by the configuration generation engine during deployment.

system service A component that is automatically deployed to all applicable hosts when the hosts are prepared. System services define utility controls and resources that can be used by other components.

targetable component A component that creates a host that serves as a deployment target for other components when it is installed. When a targetable component is uninstalled, the host it created is automatically deleted.

top-level component A contained component that, when installed, can be used by any component just as if it had been directly installed by a plan. A top-level contained component defines services that will be used by the container component as well as by other components.

upstream In the provisioning system network hierarchy, the server that is closer to the master server. For example, the master server is upstream from the local distributor. The docal distributor is upstream from any remote agents that are connected to that local distributor.

variable See *component variable*.

variable settings A collection of variable values that can be used to override the default values of one or more component variables. Based on the variable settings that you use, you can specify different values for component variables. You specify the variable settings to use when you run a plan.

virtual host Services that act as a host for other services. For example, a virtual host can represent an application server that acts as a host for web applications.

XML schema The language used by the provisioning system to create plans and components.

Index

D

R

S

T

U

V

X