Sun Professional Services

JumpStart(tm) Enterprise Toolkit User Guide

Authors: ***Marty Lee***

Mike Ramchand

Revision: ***3.3.0***

Date: ***27 February 2004***

Sun Professional Services
Sun Microsystems Ltd
Java House
Guillemont Park
Minley Road
Blackwater
Camberley
Surrey
GU17 9QG

Revision History

Version	Author	Reason for Issue	Date
3.0	MJL	First release of JumpStart Enterprise Toolkit version 3	4. Sep. 2002
3.1	MJL	Updates for revision 3.1 of the toolkit	15. July. 2003
3.3	MJL	Updates for revision 3.3 of the toolkit	9. Feb. 2004

Table of Contents

1. Introduction

1.1 Background

The JumpStart^{tm} Enterprise Toolkit is a Sun Professional Services UK installation tool developed around JumpStart^{tm} technology which is available within the Solaris^{tm} Operating Environment. It builds upon the basic operating system functionality of JumpStart to provide a modular approach to post installation configuration. It also implements recommended practice gained from Professional Services extensive experience of installing and configuring Sun servers.

1.2 Document Purpose

The purpose of this document is to:

- Cover how to install and configure the toolkit
- Define how to use the toolkit to build a target server
- Illustrate how to maintain the toolkit configuration

This document does not cover the underlying JumpStart technology in any great depth. Further information on how JumpStart works can be found in the Answerbook documentation, available online at http://docs.sun.com, or in the book 'Automating Solaris Installations. ISBN 0-13-312505-X'

2. JumpStart Enterprise Toolkit

2.1 Introduction

JumpStart was developed by Sun Microsystems to automate the installation of the Solaris Operating Environment over a network. JumpStart makes use of a client- server architecture, where an install server is created that is able to deliver the Solaris Operating Environment packages to a target machine (a client) over a local network

Functionality was also coded so that an administrator could fine tune the configuration and add additional packages through the use of a post-installation script, but this often involved quite complex scripting and in depth product knowledge, to the extent that getting it to work correctly often took far longer than a comparable install by hand.

Sun Professional services in the United Kingdom recognised the benefits of JumpStart but also saw the need to automate the installation and configuration of common applications such as volume manager software and third party or unbundled packages.

The result was the creation of the JumpStart Enterprise Toolkit, which was developed to be;

- Flexible: The toolkit must be flexible enough to support a variety of target server types. The toolkit should allow multiple identical target servers to be configured as well as many different configurations within the same framework.

- Modular: There should be a separation between the toolkit itself, the installable products, and the target server configurations. This provides a variety of benefits such as the ability to update single products or add new ones without impacting the existing configurations.

- Intuitive: The configuration of a target server should require the minimum effort on the part of the installer. Detailed product knowledge of JumpStart or additional products should not be required to automate the build of servers; this does not however preclude the concept of understanding a product in order to configure it properly; only that the installer does not need to know all the installation commands for every product.

The toolkit provides a set of utilities which are used in the following areas:

- Configure the JumpStart server and populate the server with media and patches.

- Set up a target server configuration by specifying the products to be installed and the installation parameters for each of them.

- Provide a library of common functions which are used by both the scripts within the toolkit and any additional product modules, in order to reduce code repetition.

2.2 JumpStart Walkthrough

Although the aim of this document is not to provide an in depth description of the JumpStart technology, this section aims to refresh the reader with the most pertinent points about JumpStart.

When a target server is being jumpstarted, the following steps are performed:

2.2.1 Boot net

When a 'boot net' command is issued at the target servers OBP prompt, the target server starts to broadcast on the network, firstly through the RARP protocol, so it can establish its IP address and then for a boot server from which to boot the Solaris Operating Environment. The Solaris supplied script '*add_install_client*' performs the function of defining the boot server, the location and type of the operating system image, and adds the target servers ethernet address to the local '/etc/ethers' file for the rarpd daemon to use. The main boot information is stored in the /etc/bootparams file on the JumpStart server. In the case of a non-interactive install, the bootparams file also contains information about the location of the JumpStart scripts, and the sysidcfg file for that particular target server.

2.2.2 System Identification

As the kernel is booting on the target server, it tries to obtain its system identification parameters from a variety of sources; NIS, NIS+ and a sysidcfg file. This information is used to plumb up the interfaces, along with setting locales and timezones properly prior to bringing up the actual installation screens. If it is unable to locate this information, the install will go interactive at this point.

2.2.3 Installation Configuration

Once the installation script starts, it searches for a configuration directory and for the rules.ok file within that location. If an entry within the rules.ok file matches the target server, it runs the defined 'begin' script which can be used to preserve any existing configuration information on the target server.

Once the 'begin' script has been executed, the 'profile' defined in the rules.ok files is used to drive the main install. The profile contains the information required to install Solaris on the machine, the boot disk and its partition information as well as what Solaris cluster/packages to install.

The system then proceeds to partition the boot disk/disks, install Solaris, and apply any bundled Solaris patches. Depending on the cluster selected, the network speed and the speeds of the respective servers, the time taken to install could range from minutes to hours.

N.B. *The rules.ok file can indicate that the profile will be generated/provided by the 'begin' script, allowing the automatic generation of profiles as required.*

2.2.4 Post Installation Configuration

After the installation phase, the target server then proceeds to run the *'finish'* script as defined in the profile file. At this point, the target server operating system is running an NFS mounted version of the kernel, with volatile files in /tmp, which is held in memory. The freshly installed real operating system is mounted on "$ROOTDIR" (typically /a). The configuration directory (where the rules.ok file is found, and where the begin, profile and finish files are placed relative to) is mounted on "$SI_CONFIG_DIR".

Typically, a finish script will install the recommended patch cluster and any other hardware specific patch clusters defined, along with whatever else the author has decided to put into the script. It is however, up to the system administrator to write this script and test it thoroughly.

Once the finish script has completed, the system will reboot off the freshly installed disks and become a standalone machine, with no further dependency on the installation server.

2.2.5 Post-boot

Some packages do not install cleanly when the new root disks are available through "$ROOTDIR", even though pkgadd and patchadd have a -R flag to cope with this situation. In most cases, the problem lies with the package or patch postinstall scripts, which try to directly change items under /, i.e. /kernel/drv rather than ${PKG_INSTALL_ROOT}/kernel/drv.

To overcome this limitation, the most common workaround, is to drop an additional 'rc' script into the newly installed system, so that it is executed on the first boot, and additional configuration can be performed. This does have the added benefit of not requiring any special options to pkgadd/patchadd, since root really is on / at this point.

2.3 Traditional Weaknesses

The majority of the weaknesses in the traditional method of automating installs using JumpStart are time based. By the time an administrator has worked out how to install the application in an automated fashion and has crafted the scripts to be executed through the finish mechanism, the problem of debugging the final work is hindered by the fact you need to do a full Solaris installation before getting to the actual script under test.

In many cases, the only time JumpStart is beneficial, is if there are a huge number of nearly identical machines to build. Not a common situation to be found in.

This problem is often compounded by the fact that the scripts themselves are often hard-coded with configuration information. Normally this information is constant across the builds at that point in time, but when re-approached in the future, the values often need changing. At this point, the new system administrator makes the decision to change the script, therefore breaking builds of older machines which still reference it, or making a copy of the script and changing the new version. This code forking of the script leads to problems when bugs are located in the original or indeed one of the derivatives: where else has the code been copied ? If we change the other version(s), how do we satisfy ourselves that the change doesn't break their functionality ?

The JumpStart Enterprise Toolkit is the product of many contributors from around the world within Sun, who wanted to develop a common framework that would eliminate these weaknesses.

3. Installing the JumpStart Enterprise Toolkit

The JumpStart Enterprise Toolkit is delivered in standard Solaris 'pkgadd' format. The key package, SUNWjet, contains the basic framework and a module to configure standard Solaris Operating Environment installation tuneables.

The toolkit uses a modular approach for additional functionality, and these modules are delivered as separate packages, all of which are dependant on the main core toolkit package. Every module package can be considered as the glue between the standardised configuration approach found in the toolkit, and the actual product installation scripts. Each module uses tools and utilities provided within the core toolkit to install and configure a particular (un)bundled product, reducing the complexity of the module scripts, and removing most duplication of code.

The toolkit packages do not contain the application binaries for associated products. or the required Solaris Operating Environment images. These must still be obtained as usual and copied to the appropriate locations on the JumpStart server. The toolkit provides utilities to assist with this task.

3.1 Hardware Requirements

3.1.1 Installation Server

Any machine capable of running the Solaris Operating Environment and which is based on either the SPARC or Intel x86 architecture can be made into a JumpStart server. Although the toolkit itself does not consume any significant amount of disk space, the server must have access to enough local storage to accomodate any Solaris images, application packages and patches that will be used for installations.

In a typical install, the following disk requirements were observed; the actual sizes will vary dependant on the modules, operating systems and applications and patches required.

Filesystem	**Observed size**	**Recommended Free Space**
/opt	2Mb	5Mb
/export/install	3Gb	4Gb or more

3.1.2 Network

The JumpStart server needs to be on the same subnet (without using routers) as the target servers to be built. The toolkit does not offer easy configuration of 'boot servers' as found in traditional JumpStart, but such 'boot servers' can be created if required.

The speed of the network does affect the installation times, but in our experience, the main limiting factor is the updating of the package database on the target server, hence local disk speeds on the individual target server. When using technologies such as WebStart Flash, the speed of the network is much more influencial. Anything from 10BaseT to Gigabit has been successfully deployed.

It is recommended that a backup, admin or private JumpStart network be utilised for building the target servers. If this is not available, the user must be aware of the implications on security and network bandwidth of building target servers over a public and/or production network.

3.2 Installing the JumpStart Enterprise Toolkit

The installation of the toolkit can be performed by adding the core toolkit package followed by any additional modules that are required. For this example we will just use the core toolkit.

You will need to be the root user on the JumpStart server to perform this operation.

```
# pkgadd -d . SUNWjet
```

The toolkit will be installed in /opt/jet by default and during the installation of the package, it will automatically share the /opt/jet directory through NFS, as a read-only filesystem.

3.2.1 Configuring Default Media Paths

The JumpStart Enterprise Toolkit is designed to make the management of associated Solaris and application media as simple as possible. The toolkit uses a simple configuration file (/opt/jet/etc/jumpstart.conf) to define where Solaris media, application media and patches should be placed. By default these locations are set to subdirectories of /export/install but can be changed by the installer.

JS_SOLARIS_DIR	Location of Solaris images
JS_PKG_DIR	Location of application media
JS_PATCH_DIR	Location of Solaris & application patches

These locations should be reviewed at this point and changed before proceeding further. It is also the responsibility of the installer to ensure that the selected areas are properly NFS exported so that the target servers can access them during the build.

To export the areas, the file /etc/dfs/dfstab should be edited and once completed, the NFS server restarted.

3.2.2 Other Toolkit Defaults

Within the configuration file (/opt/jet/etc/jumpstart.conf) are two other configuration parameters. The first is *JS_Default_Root_PW*, which should contain the encrypted form of your default root password. Root passwords can be set on an individual target server basis, but when creating a new configuration, this value will be used.

By default the value '*boajrOmU7GFmY*' will be present, which translates to '*newroot*'.

The other configuration option is *JS_BUILD_DIR* which identifies a location on the target server that will be used to store configuration information and scripts to be used through the JumpStart process. Once the build has completed, this directory is no longer required and can be deleted. Since it also holds configuration information from the JumpStart server, this may well be considered a security risk and/or a good backup in the case of disaster. The choice of location and what to do with the resulting directory is left up to the reader. By default, an area under /var/opt/sun/jet is used and is left on the machine after the build.

3.2.3 Removal of the Toolkit

If you should wish to remove the toolkit, simply remove any modules that have been added, followed by the toolkit framework package itself.

```
# pkgrm SUNWjet
```

If you have shared out any other media directories, you will need to edit the / etc/dfs/dfstab file by hand and remove the unwanted entries.

3.3 Toolkit Directory Structure and Contents

The toolkit resides in a single directory, by default /opt/jet, with each of the main components in sub-directories as follows.

3.3.1 Top level files

The top level directory contains several files:

- rules: a generic rules file for all target servers.
- rules.ok: version of the 'rules' file with a checksum.
- check: script to generate the file 'rules.ok' from the file 'rules'.

None of the above files should ever need to be modified by the installer.

- jumpstart.conf: configuration options for the toolkit - see sections 3.2.1 & 3.2.2.

3.3.2 Templates directory

The *Templates* directory contains a template file for each of the target servers to be built by this JumpStart server.

A template file can be created using the *make_template* script found in the '*/opt/jet/*bin' directory, and can be tailored to include the appropriate product configuration options as required. See Section 4.3 for further information.

3.3.3 Clients directories

The *Clients* directory consists of a set of subdirectories, one named after each target server. On a fresh install, this directory will be empty until the first target server is defined.

The individual target server subdirectory will contain at least the following files:

- sysidcfg : file required for the target server to identify itself non-interactively.
- <target server>.conf: copy of the template used to define the target server.
- profile: normal JumpStart profile file describing the disk partitioning and the Solaris cluster to be installed.

This target server specific directory is created and populated through the use of the *make_client* script found under the *'/opt/jet/bin'* directory.

3.3.4 Products directory

The *Products* directory contains a set of sub-directories, one for each product module available for installation. The core toolkit contains two modules called "base_config" and "custom" which contains the Solaris patching, configuration and tuning scripts, along with a mechanism for adding arbtrary packages, patches, files and scripts to a target server.

Each module subdirectory contains at minimum a single file called "install" and this file will perform all the installation tasks required by the module. The toolkit provides a set of functions that can be called from within this script to perform common tasks such as adding packages, patches, copying files, and specifying post-boot scripts to be run. In essence, the scripts within this directory contain the logic required to install a particular product, with some of the common tasks being performed by a set of toolkit supplied library functions.

If the product installation includes packages to be installed, a package.matrix file is included which contains a list of supported operating systems and product version numbers with a list of the packages required to be installed. There is also a patch.matrix file which will define the required patches in a similar fashion.

Any additional non target server specific files needed for the installation of this product may also reside in this directory.

3.3.5 Utils directory

The *Utils* directory should be considered the private area of the toolkit. Within this directory are several files and scripts, the more important are briefly described below.

- server_functions.sh: This file is only used by scripts that run on the JumpStart server itself; it provides some simple functions that are JumpStart server specific.
- lib: This directory contains the set of functions which can be used by the finish script and all the module install scripts. This is where all the common utilities actually live.

- S99jumpstart: This script is copied onto the target server and is used to run any post installation scripts at subsequent reboots.

- begin: a generic begin script which is to extract the profile information for the target server.

- finish: a generic finish script which is used to set up the toolkit environment on the target server and then call each of the requested product install scripts in turn.

- mediamgmt: this directory contains the methods used to obtain media from the JumpStart server. By default, the target server will use NFS.

3.3.6 bin directory

The *bin* directory contains the utility scripts that are intended for the system administrators who are using the toolkit. It can be added to '$PATH' for the root user, but should not be added to the '$PATH' for regular users, since most of the JumpStart scripts require root access to execute.

A brief description of the more important scripts now follows:

- add_solaris_location: This script is used to inform the toolkit of the location of a Solaris image.

- copy_product_patches: This script is provided as an easy way to copy product patches, provided the product complies to the patch.matrix form of installation. Currently all the base products do.

- copy_product_media: This script is provided as an easy way to copy product media, provided the product complies to the product.matrix form of installation. Currently all the base products do.

- copy_solaris_media: This is a wrapper around the 'setup_install_server' script found on the Solaris CD, and combines the copy with the add_solaris_location script automatically.

- copy_solaris_patches: This script copies the appropriate SunSolve patches to the appropriate location on the JumpStart server for subsequent installation on the target server.

- make_template: This script creates a basic template for the target server by combining the appropriate product configuration files that have been installed into the toolkit.

- make_client: This script parses the target server template file, and creates the required files in the Clients directory for that server.

  The usage of all of these commands can be displayed by running the command with no arguments.

4. JumpStart Server Preparation

Before any target servers can be built from the JumpStart server, a number of steps must be taken in order to populate the server with the required media.

4.1 Solaris Media Installation

To provide basic JumpStart functionality, an image of the appropriate Solaris Operating System media must be installed on the JumpStart server.

The version or versions of Solaris to be copied onto the JumpStart server depend on those required by the target server(s), and indeed, multiple versions of the Solaris media can be installed on the same JumpStart server at the same time.

While it is recommended the media is copied to disk on the JumpStart server, it is not essential, and the CD-ROM can be shared appropriately to allow the target servers to boot of it. The configuration and use of a CD-ROM is beyond the scope of this document and will severely restrict the performance of the target server build, along with the ability to support multiple versions of Solaris.

DO NOT SIMPLY COPY THE SOLARIS MEDIA, YOU MUST USE EITHER THE SCRIPTS ON THE SOLARIS CD, OR THOSE PROVIDED BY THE TOOLKIT, OTHERWISE YOUR JUMPSTART MAY FAIL!

4.1.1 Copying the Solaris Image to Disk Using the Toolkit

Mount the Solaris CD-ROM on the JumpStart server, either through a local CD-ROM, or using a shared CD-ROM on the network.

For a oneshot installation, use the copy_solaris_media command in the /opt/jet/bin directory. This utility requires a single argument, the location of the media to copy.

```
# /opt/jet/bin/copy_solaris_media /cdrom/cdrom0
```

This utility will copy the media to an area under $JS_SOLARIS_DIR (defined in /opt/jet/etc/jumpstart.conf), and then add the location of the media into the toolkit. Once it has completed, it will output details of the image just copied.

Please note, the time taken to copy the image depends on the speed of the media it is being copied from. An old single speed CD-ROM will take a considerable quantity of time, and should probably be left overnight!

4.1.2 Copying an Image using 'setup_install_server'

You can also use the '*setup_install_server*' script that is located on the Solaris CD to copy the media to the JumpStart server. If you use this approach, please be aware of the need to also run the '*add_to_install_server*' script that is found on CD2 of Solaris 8 and above, otherwise you are not creating a complete image on the JumpStart server.

Once completed, please register the location using the next procedure.

4.1.3 Registering Existing Solaris Images

When using self copied or existing media locations, you need to take one further step, which is to inform the toolkit of the location of the media by using the '*/opt/jet/bin/add_solaris_location*' command.

```
# /opt/jet/bin/add_solaris_location <tag> <path to image>
```

The tag parameter allows you to refer to this image by a shortened or abreviated name, rather than having to remember the complete path of the image. The tag is an arbtrary single word of text to describe the image; examples: 8_0202, 9_GA etc

4.1.4 Checking for Existing Images

The toolkit provides a utility command '*list_solaris_locations*' which will list each tag and its corresponding physical location on the JumpStart server.

This utility is especially useful when working on a pre-existing JumpStart server that has been set up or modified by another person.

The locations of the Solaris images are maintained in a simple flat file in the /opt/jet/etc directory - solaris_media_locations. If you remove an image, you can just edit this file and remove the entry for the image.

4.2 Solaris Recommended Patches

In order to install the recommended patch cluster you will first need to locate the tarball on either a SunSolve CD, or have previously downloaded the recommended patch cluster from the SunSolve online web site. Obtaining the recommended patch cluster is beyond the scope of this document.

4.2.1 Copying Recommended Patches Using the Toolkit

The toolkit provides a utility function '*/opt/jet/bin/copy_solaris_patches*' which will put the patches in the correct location, as defined by *JS_PATCH_DIR* in the configuration (/opt/jet/etc/jumpstart.conf) file.

```
# /opt/jet/bin/copy_solaris_patches <os version> <path to patch cluster>
```

The utility takes two arguments, the first being the version of the Solaris Operating Environment, examples 2.6, 7, 8 or 9; the second argument is the full path to the patch cluster itself. This could be the path to the file on the SunSolve CD, or the path to a downloaded online version.

If you are unsure about the correct value for 'os version', the argument is the prefix to the file, for example if the cluster is "2.5.1_Recommended.tar.gz", then 'os version' is "2.5.1".

4.2.2 Manually Populating Recommended Patches

The recommended patches are expected to reside in *JS_PATCH_DIR*, as defined in the configuration file (/opt/jet/etc/jumpstart.conf). The patch clusters should live directly under that directory, for example "*/export/install/patches/8_Recommended*" and be fully extracted from their tarball.

If you wish to add additional patches, there is a file called "*patch_order*" which defines which patches are installed and their associated order of installation.

5. Target Server Configuration

There are a number of steps required to prepare the toolkit for installation of a new target server:

- Ensure the JumpStart server has all the required media, packages and patches.
- Collate all information necessary to build the target server.
- If using DNS, populate the JumpStart servers hosts files with target server details.

5.1 Target Server Preparation

5.1.1 Collate all information necessary to build the target server

This includes such things as hostnames, ip address and product licence keys.

Remember to allow time to apply for and receive any required licence keys.

5.1.2 Populate local hosts files with target server details (optional)

When using DNS without a local entry in the JumpStart servers /etc/hosts file, the JumpStart server will use fully qualified names in /etc/bootparams for the target server, but the target server itself will only supply an unqualified name during the bootparams request. The outcome is that the target server will not be able to locate a suitable configuration entry and the installation will go interactive.

To overcome this issue, create an entry for the target server in /etc/hosts on the JumpStart server. Ensure that this ip address is unique and is on the same network as the JumpStart server.

5.2 Create a 'template' for the Target Server

All of the target server configuration is performed through the use of a single flat file template. To create a new template for the target server, use the '*/opt/jet/bin/make_template*' script with a single argument of the target server name.

```
# /opt/jet/bin/make_template <target server name>
```

This will then create a new template file /opt/jet/Templates/<target server name>

If the Toolkit has been supplemented with additional modules, this template will contain all configuration options for all modules. To limit the modules presented in the template, list the individual modules you require after the target server name.

```
# /opt/jet/bin/make_template <target server name> <module1> <module2>....
```

The standard module "base_config" will always be included. To overwrite an existing template, make_template can be called with the "-f" option.

5.3 Customise the New Template

The template should now be edited using your editor of choice; the minimum configuration should include the architecture type, ethernet address and Solaris version to be installed on the target server. The Solaris version is one of the tags defined earlier in section 4.1; these can be listed by using the '*/opt/jet/bin/list_solaris_locations*' command.

The template file itself is sourced by a bourne shell script, so must conform to standard bourne shell construct rules. For example, variables that have multiple values must be enclosed by either single or double quotes. In addition, special characters such as ';' and '*' must be protected by quotes to prevent shell expansion etc.

The template file contains many comments to assist with the definition of the target server. Further information on module specific configuration can be found in a file named 'ReleaseNotes' which should be present in all modules, under the module specific directory within /opt/jet/Products.

5.4 Set up the JumpStart

Once the template has been edited, the actual files required to enable a JumpStart install must be created, and specific scripts run. The toolkit provides a utility command called '*make_client*' to perform this task. This utility will create and populate the / opt/jet/Clients/<target server> directory with all the required information to boot the target server according to the details in the template. It will also ensure that the JumpStart framework is primed to allow install requests from the target server.

```
# /opt/jet/bin/make_client <target server name>
```

The '*make_client*' script configures the JumpStart server with the appropriate files for the target server and then calls the standard '*add_install_client'* script, supplied within the particular operating system image. It also performs a number of checks on the information supplied in the template and may indicate possible problems with the configuration.

It should be noted that the template is purely a mechanism for collating all the configuration into one location. It serves no other purpose. If you subsequently need to change an option in the template, you must remember to re-run the '*make_client*' command to propogate your changes into the '/opt/jet/Clients' area.

When re-running the '*make_client*' command, you will need to supply the '-f' option to force the overwrite of the exisitng configuration information.

The target server specific directory will now contain all the information that will be required during the install, and in particular, the JumpStart specific files 'profile' and 'sysidcfg' will have been automatically created.

Files and scripts that are to be applied to the target server, as defined by the custom product in the template, should also be placed relative to this target specific directory.

5.5 Instigating the Build

To start the build process on the target server, make sure the target server is at the Open Boot Prom 'ok' prompt, and use the command:

```
ok boot net - install - w
```

Please note: there are spaces between every word on this line including before and after the dashes (-).

5.6 Build Sequence

Before covering some extended features of the toolkit, a brief walk through of the build sequence is required, so that behaviour can be fully understood.

The build sequence of the JumpStart Enterprise Toolkit is as follows:

- Standard Solaris installation phase
- Standard JumpStart finish script called
- Individual module 'install' scripts called
- Target server reboots
- <optional> Platform related installation tasks; reboot after each level
- <optional> Application related installation tasks; reboot after each level
- <optional> Final installation tasks (no reboot)
- login prompt appears on the console etc

The optional steps after the initial reboot are dependant on the individual modules configured within the target server template. Modules can be written in such a way that they request that the toolkit perform additional work after the first reboot. In this request they can identify whether the work should be in the 'Platform' related area, the 'Application' related area or whether the work needs to be done at the end, when there are no more planned reboots.

This distinction of Platform and Application is important, as it allows a module developer to disregard any other modules that have been or may be written. For example, the configuration of a volume management product would be done within the 'Platform' scope, while a database would be in the 'Application' scope. Neither need to know about the existence of the other, or need to consider possible interaction.

5.6.1 Reboot numbering

For the Platform and Application optional reboot sequences, the modules can request that a particular task be executed on reboot 'x' where 'x' is a numeric number greater than 0.

The toolkit will group all requests for that particular reboot level and at the appropriate time, execute the scripts and then reboot the machine. The module itself does not do a reboot.

As the machine reboots the toolkit advances an internal counter so the next set of tasks is performed. If there are no more tasks remaining in the Application set, the toolkit will advance on to the Platform set. Once those are completed, the toolkit finally looks for the set of tasks which do not require a reboot.

5.6.2 No more reboots

Certain operations require the machine to be in a stable condition, in the knowledge that it is not just about to reboot. A classic example is when you attach a mirror to a volume manager controlled device; some volume manager software requires the operation to complete the resync without interruption.

Once these set of tasks are reached, the toolkit will move the 'rc' script into the post_install subdirectoryu of the scratch area defined in the toolkit configuration file (/opt/jet/etc/jumpstart.conf), which is by default /var/opt/sun/jet. This prevents the rc script being invoked by accident at a later date.

6. Enhancing the Build

6.1 Module Overview

The basic toolkit provides common configuration options for the Solaris Operating Environment, but very little else. The design intention was to make a modular toolkit, so that changes can be made to small parts of the toolkit, without affecting the remaining code or configurations. The toolkit comes with one default module 'base_config' and an optional 'custom' module.

Modules may be made available by Sun or any third party. Details on how to write modules are covered later in this document.

6.2 Adding custom Packages, Patches, Files etc

The custom module can be used to install arbitrary packages, patches, files and run arbitrary scripts on the target server.

If the custom module options are not already present in the target server template, the following command can be run to add them:

```
# /opt/jet/bin/make_template -f -T <target server> <target server> custom
```

This command is effectively deriving a new template for the target server, based on its original configuration, but with the addition of the custom module.

When being installed on the target server, the order of installation is always:

packages, patches, files, scripts

If this order is not what you require, consider creating your own module, as described in section 8.

6.2.1 Custom Packages and Patches

The module can be used to install packages and patches at different stages of the build - see section 5.6 for information about the build sequence.

When editing the target server template, the names of the additional packages can be listed in the configuration variables '*custom_packages_[1-n]'* depending on when in the boot sequence the packages need to be installed. Likewise, patches can be defined in the variables '*custom_patches_[1-n]'*.

Packages and patches are installed in the order given, so must be arranged appropriately to satisfy any dependencies between them.

If you have many packages or patches, or there are a common set frequently used on a number of different target servers, you should consider creating your own module, as described in section 8. You may also need to consider this approach if you need to intersperse the installation of packages with patches etc.

6.2.2 Populating custom patch and package media

The custom module allows the definition of a custom package and patch set on a per target server basis and also provides two scripts to enable the package and patch media to be placed in the correct place for the toolkit to find.

When copying patch and package media, the following scripts will place the media according to the definitions of 'JS_PKG_MEDIA' and 'JS_PATCH_MEDIA' as found in the toolkit configuration file (/opt/jet/etc/jumpstart.conf). Should you wish to use an alternative location to hold all the media, please remember to modify the configuration file prior to executing the scripts.

Custom patches can be placed using the '*/opt/jet/bin/copy_custom_patches'* script:

```
# copy_custom_patches <src dir> <patch> [patch....]
```

This script takes at least two arguments, the first one is the source directory which contains the patches; any other arguments are then considered to be patch numbers, which are located within the directory and subsequently copied.

Custom packages can be transferred using the '*/opt/jet/bin/copy_custom_packages'* script:

```
# copy_custom_packages <src dir> <arch> <package> [package....]
```

The arguments are very similar to that of those used for the custom scripts command, with the inclusion of the additional 'arch' argument, which is used to define the target architecture for the packages. This will be defined by the output of 'uname -p' on the target server. Currently, the values will be one of 'sparc' or 'i386' for SPARC and IA86 architectures respectively.

Custom patches do not need the definition of an architecture, since distinctly numbered patches are released for each architecture the package is available for.

The custom module does not offer any provision for holding multiple different versions of the same package for the same architecture. Should this feature be required, consider writing a specific module to cover these requirements; the description of how to write a module can be found in section 8.

6.2.3 Custom Files

The module allows files to be created, overwritten or appended to on the target server. Files are referenced by a triple that defines the source file, relative to the target server directory within '*/opt/jet/Clients'* on the JumpStart server, the mode of operation and the destination file on the target server.

For example, for a target server called 'banana' the following line in the custom area of the template would append the contents of the file '*/opt/jet/Clients/banana/hosts'* on the JumpStart server to the file '*/etc/hosts'* on the machine banana while it was being built.

```
custom_files="hosts:a:/etc/hosts"
```

The source file(s) must be placed accordingly prior to the build of the target server, and must be located within the '*/opt/jet'* directory tree - you can not use absolute paths for the source file, or refer to files outside of the tree i.e. /etc/hosts, /etc/passwd etc. The middle field of the triple, specifies whether to 'a' - append or 'o' - overwrite the destination file on the target server.

If a set of files are common to a number of target servers, consider creating a holding area within the '*/opt/jet/Clients'* directory, and placing the common files within that directory. The templates for the target servers can then refer to the files as:

```
custom_files="../common/hosts:a:/etc/hosts ../common/ftpusers:o:/etc/ftpusers"
```

where the files are placed in '*/opt/jet/Clients/common'* rather than multiple copies in each target server specific directory.

Since the template file is just a bourne shell script, you can continue lines using regular techniques, such as the use of the '\' character, or by appending to the variable 'custom_files="${custom_files} ......"'.

6.2.4 Custom Scripts

Custom scripts are defined just by the source location of the script; the toolkit will copy the script to the scratch area on the target server during the build and execute it at the appropriate point.

In the same fashion as the custom files described in the previous section, the source of the script must be within the '*/opt/jet'* directory but can be a relative reference to a common holding area.

The custom module does not offer any provision for executing scripts prior to the first reboot, when the system is running on the NFS image from the JumpStart server and the real target server filesystem is mounted on '$ROOTDIR' (/a).

This decision was taken to try to cut down problems with people forgetting to include '${ROOTDIR}' in all their file accesses and having to endure the debug cycle that includes re-installing Solaris every time.

If you have a requirement to execute a script prior to the first reboot, consider creating a module as described in section 8.

6.3 Replicating an Existing Target Server Configuration

If the toolkit is being used to rollout a number of identically configured servers, it may be beneficial to simply copy and modify a known good template for a target server instead of creating a new one from scratch each time.

The toolkit provides a mechanism for creating a new template based on an existing one, through the use of the '*make_template'* script.

For example, to create a new template for the target server 'apple' based on the existing server 'banana', you can do the following:

```
# make_template -T banana apple
```

This will copy almost all the settings from 'banana' in to the new 'apple' template, with the exception of ethernet (MAC) address, IP address and netmask. These values will be obtained from the name service switch as would have occurred during a standard make_template invocation.

6.4 Creating a Generic Template

If a number of target servers require a very similar base configuration, consider creating a 'generic' template containing all the common configuration information and then derive the individual target server templates from the generic one using the process described in the previous section.

If individual target servers require additional modules over and above the generic template, you can use the '*make_template'* command shown below:

```
# make_template -T generic orange module2
```

This command is deriving a new template for the target server 'orange', based on the existing template 'generic' and adding the module 'module2' to the newly created template.

6.5 Using Alternative Patch Definition files.

In many installations it may be beneficial to standardise on a known fixed set of patches, which may be different from that defined by the individual module patch.matrix files.

6.5.1 Target Server Specific Definitions

These can be achieved by creating a set of subdirectories within the target server specific '*/opt/jet/Clients'* area, with a similar structure to that of the Product directory. These subdirectories can then be populated with patch.matrix files for each module. For example:

```
# mkdir /opt/jet/Clients/orange/sds
# cp /opt/jet/Products/sds/patch.matrix /opt/jet/Clients/orange/sds
# vi /opt/jet/Clients/orange/sds/patch.matrix
```

In the above example, we have taken the patch matrix file for the Solaris Volume Manager module (previously known as Solstice DiskSuite, hence the module name sds) and made a target server specific version. During the installation, the toolkit will now use this patch matrix over the one supplied by the module itself.

To do something similar for the standard Solaris Recommended patches, the process is slightly different. Within the target specific area under '*/opt/jet/*Clients' create a subdirectory 'base_config' and then a further subdirectory of 'base_config' named after the operating system release you wish to define a patch order file for; for example:

```
# mkdir /opt/jet/Clients/orange/base_config
# mkdir /opt/jet/Clients/orange/base_config/5.8
# cp /export/install/patches/8_Recommended/patch_order \
  /opt/jet/Clients/orange/base_config/5.8
# vi /opt/jet/Clients/orange/base_config/5.8/patch_order
```

This allows the recommended patches for this target server to differ from the regular recommended set that would normally be applied.

6.5.2 Generic Definitions

If a number of servers need to share the same set of patches at specific levels, the previous method of creating target server specific patch files is rather cumbersome.

The toolkit also provides a mechanism for referencing a similar structure to that described in the previous section, but held outside of the target server specific area under '*/opt/jet/Clients'*.

The area must be created somewhere under '*/opt/jet'* but placement is up to the implementor. In this example, the directory '*/opt/jet/PatchDefs'* has been chosen.

```
# mkdir -p /opt/jet/PatchDefs/1.0
# mkdir -p /opt/jet/PatchDefs/1.0/base_config/5.8
# cp /export/install/patches/8_Recommended/patch_order \
  /opt/jet/PatchDefs/1.0/base_config/5.8
# vi /opt/jet/PatchDefs/1.0/base_config/5.8/patch_order
# mkdir -p /opt/jet/PatchDefs/1.0/sds
# cp /opt/jet/Products/sds/patch.matrix /opt/jet/PatchDefs/1.0/sds
```

In the above example, we have created an area under '*/opt/jet/PatchDefs'* called '1.0' to represent the 1.0 release of our configuration. Within this configuration, a given number of patches at specific levels have been agreed upon. It is the intention that all future builds of target servers at level '1.0' use these patches at the defined levels. Should we need to change the patches, a new release level '1.1' or '2.0' would be created in a similar fashion.

When we wish to build a target server using the configuration '1.0' of the patches, we create the template in the usual manner, and use this modified version of '*make*_client':

```
# make_client -p /opt/jet/PatchDefs/1.0 orange
```

This will populate the target server specific *'/opt/jet/Clients'* directory with the patch_order and patch.matrix files as in the previous section.

7. Maintenance Procedures

7.1 Updating the Toolkit and Associated Modules.

The toolkit framework is supplied in pkgadd format; to update it, pkgrm the old version and then pkgadd the updated version. The dependency checking will inform you that any installed modules are dependant on it, but you should continue.

Any modules should be updated in the same fashion.

Where possible, updates to the toolkit will continue to support templates created with older versions of the toolkit. If enhancements have been made to the toolkit supplied 'base_config' or 'custom' modules, or the individual modules have been improved, it may require the creation of new templates to fully realise the new features.

Since the templates are simply bourne shell scripts, any additional variables available in new templates may simply be manually inserted into existing templates to enable those features.

7.2 Exporting the media, pkgs and patches directory

The utility scripts provided with JumpStart have basic functionality for adding the NFS share of the target servers boot directory at configuration time, however, they do not encompass the additional requirements of the JumpStart Enterprise Toolkit.

It is suggested that your chosen location for the media repository is shared; for example, the following lines could be added to the /etc/dfs/dfstab file:

```
share -F nfs -o ro,anon=0 /export/install
```

ENSURE THAT YOU SHARE THE ABOVE DIRECTORIES AS ro,anon=0 AS CORRUPTION OF THE SOLARIS IMAGE CAUSES VERY STRANGE ERRORS THAT ARE DIFFICULT TO FIND AND RESOLVE!

The above example assumes that you have chosen the default locations for media, packages and patches. If not, you must ensure that your seelcted directories, as found in '*/opt/jet/etc/jumpstart.conf*' have been properly shared.

The '*/opt/jet'* directory share is added automatically during the package installation; if for any reason it is not added, please add it as per the example above.

7.3 Product Media Installation

7.3.1 Product Packages

If you are using additional modules which require media, the packages required for the modules, such as Solaris Volume Manager, can be added to the JumpStart server using the command '*/opt/jet/bin/copy_product_media'*.For example:

```
# copy_product_media <module> <version> <src media>
```

The versions the module supports should be defined in the documentation that accompanied it, or inspect '*/opt/jet/Products/<module>/package.matrix'* if it exists.

7.3.2 Product Patches

The patches required for a particular module can be copied using the '*/opt/jet/bin/copy_product_patches*' script. For example:

```
# copy_product_patches <module> <version> <src media>
```

7.3.3 Package and Patch Definitions

Most of the modules that have already been developed by Sun, use two files within the module directory to define what packages and patches are required for a specific version of the product. By using this mechanism, the logic of how to install the particular product remains unchanged and new versions can be added quickly without risk of corrupting a script.

The two files are called '*package.matrix'* and '*patch.matrix'* and can be found in the '*/opt/jet/Products/<module name>'* directory for each module.

The files themselves are colon delimited lines which define the set of packages or patches to install on a per operating system release, per hardware platform and per product version basis.

As new product patches become available over time, it will be necessary to update the relevant patch.matrix.

New patches should be thoroughly tested before being implemented in a production environment.

The JumpStart Enterprise Toolkit has a pre-defined set of patches for the modules supplied, but given the nature of software patches, these definitions will quickly become out of date.

8. Creating Modules

8.1 Scope

The module is the glue between the JumpStart Enterprise Toolkit framework and the facilities provided to perform the actual installation of the product software. The module presents a set of configuration options in the template to allow per-target server options to be set, which are then used to drive the product installation according to those parameters.

There are no special requirements or restrictions about the scope of what options are presented in the template, or how to drive the product installation. There are some guidelines however, to try to ensure that modules co-exist with each other as much as possible and have no strange dependencies etc.

8.2 Module Design Guidelines

The toolkit itself was designed around some basic principles. The modules that were initially created alongside the toolkit also followed this style. While every module developer has the option of going down their own path, the following should be at least considered and followed if possible!

8.2.1 Module Coverage

Each module should cover a particular scope of an application, preferably without replicating the functionality available in another module. In addition, a module should be appropriately sized; if the module can be sensibly broken into sub components, then smaller modules should be created instead of one large monolithic module.

For example:

> A designer may decide to write a module to provide security services to a target server installation. As part of this module, the designer decides to install a firewall and a set of hardening scripts.

In this case, the designer should investigate whether another module already covers either of these two areas and could be leveraged. In addition, if either the firewall or hardening scripts may be used elsewhere in isolation from each other, it may be more flexible to create them as two modules rather than bundling them together.

8.2.2 Module Dependencies

Each module should be self sufficient and not expect or rely on another module to be present. If such interaction is required (and sometimes it does make sense), then the module should be clearly identified as being dependent on the other module.

If the situation does arise, it should be determined whether the functionality is best represented in distinct modules or whether the two modules that are dependent on each other are better represented in one encompassing module.

The basic premise is to keep items together where it makes sense and not to create lots of smaller modules just for the sake of it.

8.2.3 Module Interactions

Where the case is strong for module seperation and the modules are dependent (in at least one direction), the modules should try to interact with each other so that the correct (desirable) outcome is achieved.

The toolkit has a simplistic mechanism for setting and retrieving hints on a per module basis. Whether a module will pick up the hints, is up to the designer of the module, but with co-operation between module developers, the modules can be written in such a way that they work happily in isolation, but when put together they prodvide a stronger solution.

For example:

> From a real-world example, the Sun Cluster 3 product places dependencies on how Solstice Disksuite is configured, but does not always need Disksuite installed, nor does Disksuite require the cluster software installed – indeed it is very valuable to be installed on non-clustered machines.

In this case, we have two very distinct modules; one that covers Disksuite and one that covers the Sun Cluster product. Each works fine in isolation, but when they are combined within the same target server configuration, the Sun Cluster module influences the Disksuite module so it conforms to the restrictions placed on Disksuite by Sun Cluster 3.0.

This interaction is done through the use of module hints and their behaviour in the standalone scenarios can be summarised as follows:

- When only the Sun Cluster module is active, Sun Cluster sets the hints, but Disksuite is not present to pick them up

- When only Disksuite is active, it looks for the hints, but they have not been set by any other module, so it uses it's regular default behaviour.

What the hints actually represent is totally up to the module developer. Close co-operation between module developers will enable the most efficient use of hints; if possible, the hints should be documented within the module Release Notes, so other module developers may take advantage of the additional interfaces.

8.2.4 Module Coding

The module developer should try to select a standard scripting language that will be available during the JumpStart installation. For example, the use of a 'bash' is not possible, as the NFS boot image the target server uses does not contain that shell.

Where possible, the Bourne Shell should be used as this is know to exist on all versions of Solaris. Only as a last resort should a compiled language be considered.

8.3 Module Directory

Every module has its own directory structure, located in the 'Products' subdirectory from where the main toolkit framework was installed. The name of the directory is the name used by the toolkit to reference the particular module.

For example, the module 'sds' (Solstice Disksuite) resides in ..../Products/sds/, and it alone controls what resides in that directory and how it is used; with some notable exceptions listed below.

No further interaction is required to 'register' a module with the toolkit – the presence of the directory is sufficient. It is not advisable to create symbolic links in the 'Products' directory to other locations outside of the main toolkit installation point; the target servers may not be able to access such directories during the installation, unless other measures have been taken to provide such functionality.

8.4 Module configuration

8.4.1 <module>.conf

Each module is intended to be configurable to some extent by the user; although this is not a mandatory requirement, it is normally expected to be implemented.

The toolkit takes a very simplistic view on providing module configuration to the user. When a template is created through the toolkit '*make_template*' command, a single flat file containing the concatenation of the core '*base_config*' configuration file, followed by the configuration files from each of the modules that have been selected, is produced.

By providing a configuration file named after the module itself and with a '.conf' suffix, the toolkit '*make_template*' command will do the rest of the work.

The configuration file is a simple Bourne Shell script, and should present configurable options in the form of variables to the user; each variable should be prefixed with the module name and an underscore, to preserve the variable namespace of the module and prevent one module corrupting another.

For example, the module 'sds' presents an option to the user to select the version of the software to be installed; the corresponding part of the configuration file is as follows:

```
############
#
# Which version of the product is to be installed
#
sds_product_version="4.2.1"
```

In this example, a default value of 4.2.1 is already populated in the configuration file, as this was the most recent version of the product when the module was written.

8.5 Module Interfaces

The toolkit calls particular interfaces within the module during the lifecycle of the JumpStart process - both on the target server and the JumpStart server. Each interface is expected to be an executable shell script (or worst case, a binary), with the context of the target server supplied through environmental variables.

8.5.1 copy_media

Called on	JumpStart server
Arguments	<patches|packages> <version> <srcdir> <destdir> <arch>
Required/Optional	*Required*

The copy_media script is called when the user calls the copy_product_media or copy_product_patches scripts to manage the media for this module. The script should understand what format the application is delivered in, and perform the copy from the passed source media location, to the appropriate media location on the server.

This function allows the module to handle exotic forms of media (tar.gz, zip, bz2, etc), without the main toolkit requiring to be updated for each media type. It also enables the module developer to place the media onto the server in a known state. For example, a product may require a whole directory tree to be visible, or just a bunch of Solaris packages.

8.5.2 make_template

Called on	JumpStart server
Arguments	-none-
Required/Optional	*Optional*

When an administrator wants to create a new definition for a server build, they will execute the top level command '*make_template*' supplied in /opt/jet/bin. This top level make_template script will set up the basic target server configuration information and then call each module specific make_template script, if it is present. The module specific make_template script can perform additional work on the template; for example, populating per-client defaults for the user to edit.

8.5.3 make_client

Called on	JumpStart server
Arguments	-none-
Required/Optional	*Optional*

When an administrator wants to set up a target server for installation, they will execute the top level command '*make_client*' supplied in the bin directory of the toolkit. This top level make_client script will set up the basic target server configuration information and then call each module specific make_client script if it is present. The module specific make_client script can perform additional work in the target server specific '*/opt/jet/Clients'* directory, it may configure module hints, modify the target server profile, sysidcfg or other files accordingly.

8.5.4 begin

Called on	target server
Arguments	-none-
Required/Optional	*Optional*

During the 'begin' phase of the JumpStart process, the toolkit will check to see if the module has a script called 'begin', and if it is present, the script will be executed. Any variables set in the module configuration section of the template will be present in the environment for the script to access.

8.5.5 install

Called on	target server
Arguments	-none-
Required/Optional	*Required*

The install script is the main workhorse of the module. It is called on the target server after the main Solaris installation has completed, during the 'finish' script phase. Its purpose is to instigate the installation or configuration of the particular application.

The script itself is called prior to the first reboot of the newly installed target server. At this point, the root directory (/) is actually an NFS mounted filesystem from the JumpStart server. The real disk based root directory is located through the use of the environment variable $ROOTDIR, traditionally set to '/a'.

If the application can not be installed when root is located on $ROOTDIR, the install script should use the post-installation functionality provided by the toolkit, to schedule a subsequent installation after the first reboot. After the first reboot, the target server will actually boot of its own disk, and root will really be on '/'.

The install script is responsible for taking the user supplied configuration and driving the actual application installation and configuration appropriately. How this is achieved, is up to the ingenuity of the module developer, although a number of utility functions are available from the main toolkit, to assist with the common tasks; such as package or patch installation, file copies, message reporting etc.

Prior to the module 'install' script being called, the module configuration as defined in the template – and originally populated from the <module>.conf file – is loaded into the shell environment. The 'install' script should not expect to be called with arguments, but should instead take it's configuration from the current environment. This technique avoids the problem of each module installation script requiring different numbers of arguments etc.

8.5.6 check_client

Called on	target server
Arguments	-none-
Required/Optional	*Optional*

The '*check_client'* script can be employed by a module developer to perform basic checks about the configuration options specified in the template. When called, the environment will be configured with the variables set in the template, and the script can perform basic checking to try to reduce any installation errors.

The module may decide to check for valid options or that media exists for the selected version etc. The level of functionality provided is up to the implementor.

8.6 Support Functions in the Toolkit

The main toolkit supplies many common functions that can be utilised by modules; this allows for better code re-use and simpler modules. The best way to find out what functions are available, is to examine the directory '*/opt/jet/Utils/lib*' in which they reside.

9. Support and Updates

9.1 Terms

The JumpStart Enterprise Toolkit has been made available under the following terms:

The user acknowledges that the toolkit software script(s) detailed within this document are not a generally available standard Sun product and that it is a fundamental condition of supply of the software script(s) to the user that the user accepts the same "as is" and without warranty of any kind. No support services of any kind are available for the software script(s) and Sun does not represent to the user that:

iv. operation of any of the software script(s) shall be uninterrupted or error free, or

v. functions contained in the software script(s) shall operate in the combinations which may be selected for use by the user or meet the users requirements, or

vi. that upgraded versions of the software script(s) will be issued.

If the user wishes to have the software script(s) modified, or otherwise requires support, Sun may provide the same by means of a separate consulting agreement priced on a time and materials basis.

9.2 Updates

Over time, new versions of the JumpStart Enterprise Toolkit may be made available by Sun. There is no commitment, implied or otherwise, that updates published will continue to support all existing functionality.

9.3 Contact and Feedback

The development team can be contacted through the email address 'jet@sun.com', however, any responses are made on a best endeavours basis.