Creating a Custom Oracle® Solaris 11.1 Installation Image

Part No: E28982–01
October 2012

ORACLE

121010@25097

Contents

Preface

Creating a Custom Oracle Solaris 11.1 Installation Image provides instructions for using the Oracle Solaris Distribution Constructor (DC) tool to build custom Oracle Solaris installation images.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Description	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 shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P–2 Shell Prompts

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

◆ ◆ ◆ CHAPTER 1

Introduction to Creating a Custom Installation Image

System administrators and application developers can use the distribution constructor tool to build custom Oracle Solaris installation images.

- If you have not created custom installation images before, read "About the Distribution Constructor" on page 7.
- If you are ready to build custom images, go to "System Requirements for Building Images" on page 11.

About the Distribution Constructor

The distribution constructor is a command-line tool for building preconfigured Oracle Solaris images. The tool takes an XML manifest file as input and builds an image that is based on the parameters specified in the manifest file.

The distribution constructor can build an ISO image, which is an archive file, also known as a disc image, of an optical disc in a format defined by the International Organization for Standardization (ISO). You can also create a USB image based on a generated ISO image. Unlike ISO images, however, a USB image can be created and used only on x86 systems.

Note the following:

- Depending on the image configuration, ISO or USB images can be bootable.
- Both ISO images and USB images can be installed on a system or run in a LiveMedia environment.
- An ISO image can be burned to a CD or DVD.
- A USB image can be copied to a flash drive.

The distribution constructor creates a USB image that works in various types of flash memory devices that have driver support provided by the Oracle Solaris operating system. The `usbcopy` utility must be used to copy the USB image into a USB flash drive. This `usbcopy` utility is available in the `distribution-constructor` package.

Oracle Solaris Image Types

You can use the distribution constructor to create the following types of Oracle Solaris images:

- **Oracle Solaris x86 Live Media** – You can create an x86 ISO image that is comparable to the Live Media image that's distributed with each Oracle Solaris release. You can also customize the content of this ISO image. For example, you can add or remove packages. You can revise the default settings for the resulting booted environment to create a custom ISO image or USB image.

  For more information about Live Media installations, see Chapter 3, "Using Live Media," in *Installing Oracle Solaris 11.1 Systems*. For more information about customizing the image content, see "Modifying the Manifest Content" on page 13.

- **Oracle Solaris x86 or SPARC Text Installation Image** – You can create a SPARC or x86 ISO image that can be used to perform a text installation of the Oracle Solaris operating system. The text installer can be used on systems that do not need graphics cards.

  Note – A text installation does *not* install all of the software packages that are included when installing from the Live Media image. For example, the text installer does not install a desktop. After a text installation, you can add additional packages, such as the `solaris-desktop` package.

  For more information about text installations, see Chapter 4, "Using the Text Installer," in *Installing Oracle Solaris 11.1 Systems*.

- **x86 or SPARC ISO Image for Automated Installations** – The Oracle Solaris operating system includes the automated installer tool. The automated installer (AI) is used to automate the installation of the Oracle Solaris OS on one or more SPARC and x86 systems over a network. The installations can differ in architecture, packages installed, disk capacity, and other parameters. You can use the distribution constructor to create a SPARC AI ISO image that can be used to install the Oracle Solaris OS on SPARC clients, or to create an x86 AI ISO image that can be used to install the Oracle Solaris OS on x86 clients.

  For information about using the automated installer, see Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11.1 Systems*.

Image Creation Process

The distribution constructor creates images based on settings specified in XML files, called *manifest files*. The manifest files contain specifications for the contents and parameters of the ISO images that you create using the distribution constructor. The distribution constructor contains sample manifests that can be used to create a custom x86 Live Media ISO, an x86 or SPARC Automated Install ISO image, or an x86 or SPARC text installation ISO image. See "Sample Manifest Files" on page 12.

All the fields in each manifest file provide preset, default values that will create the type of image you need. You can edit fields in the manifest file to further customize the resulting image. For example:

- You can edit the target element in the manifest to specify a different location for the build area where the image can be constructed.
- You can check the publisher that's specified and ensure that the system you are using can contact that publisher to download the packages needed to build the image.
- You can edit the software name element to specify a different publisher and repository location.

For instructions, see "Customizing Images" on page 12.

You can also create *custom scripts* to modify your installation image. Then, you can add checkpoints to the manifest file to run these custom scripts. For further information, see "Creating and Using Custom Scripts" on page 22.

The distribution constructor also includes a command-line utility, the `distro_const` command, that interprets the manifest specifications and builds the image. After you have finished editing the image blueprint in a manifest file, you run the `distro_const` command to build your image. For further information, see Chapter 3, "Building an Image."

You can use the options provided in the `distro_const` command to stop and restart the build process at various stages in the image-generation process in order to check and debug the image that is being built. This process of stopping and restarting during the build process is called *checkpointing*. Checkpointing is optional. Default checkpoints are specified in each manifest file.

After the `distro_const` command is run, you can check the simple log file and or the detailed log file for build information.

For more information, see "How to Build an Image in Stages" on page 26, or see the `distro_const`(1M) man page.

SPARC and x86 Archive Differences

The root archive for x86 images differs from the root archive for SPARC images. The whole root archive, or `boot_archive`, for x86 images is a UFS file system, compressed by using `lzma`. The SPARC platform does not support the compression of the whole root archive in this way. Instead, SPARC root archives use DCFS, which compresses each file individually. These individually compressed files might require specific handling in the manifest. For instructions, see the `<boot_archive_contents>` field in the `dc_manifest`(4) man page.

◆ ◆ ◆ CHAPTER 2

Design a Custom Installation Image

This chapter provides system requirements and describes how to design a custom installation image.

System Requirements for Building Images

In order to use the distribution constructor, you must have system requirements as described in the following table.

TABLE 2–1 System Requirements

Requirement	Description
Disk space	The recommended minimum size for your distribution constructor work space is 8 Gbytes. Confirm that you have sufficient space on your system to use the distribution constructor.
Oracle Solaris operating system	You must have the Oracle Solaris operating system (OS) installed on your system. Note the following considerations: ■ Your installed system must have network access. The distribution constructor accesses Image Packaging System (IPS) repositories that are available on the network to retrieve packages for the ISO image. You must have network access to the repositories that you specify in the manifest file. ■ When using the distribution constructor, you can create only SPARC images on a SPARC system and only x86 images on an x86 system. ■ The Oracle Solaris release version on your system must be the same as the release version of the image that you intend to create with the distribution constructor. **Note –** You must become the root role to run the distribution constructor.
Required packages	The `distribution-constructor` package, which contains the distribution constructor application.

Customizing Images

The distribution constructor creates images based on settings specified in XML files, called *manifest files*. The manifest files contain specifications for the contents and parameters for the ISO images that you create using the distribution constructor. The `distribution-constructor` package provides sample manifests that can be used to create a custom x86 Live Media ISO, an x86 or SPARC Automated Install ISO image, or an x86 or SPARC text installation ISO image.

The elements in each manifest file provide preset, default values that will create the type of ISO image you need. You can manually edit these preset elements in a manifest file to customize the resulting image. In addition, you can create custom scripts to further modify your image. Then, reference the new scripts in the manifest file.

Sample Manifest Files

The `distribution-constructor` package provides the sample manifest files described in the following table.

TABLE 2–2 Sample Manifests

Manifest Type	Manifest Location	Description
x86 Live Media ISO image	`/usr/share/distro_const/dc_livecd.xml`	Used to create an x86 ISO image comparable to the Oracle Solaris Live Media image
x86 text installation image	`/usr/share/distro_const/dc_text_x86.xml`	Used to create an x86 ISO image that can be used to perform a text installation of the x86 Oracle Solaris operating system
SPARC text installation image	`/usr/share/distro_const/dc_text_sparc.xml`	Used to create a SPARC ISO image that can be used to perform a text installation of the SPARC Oracle Solaris operating system
x86 AI ISO image	`/usr/share/distro_const/dc_ai_x86.xml`	Used to create an x86 Automated Install ISO image for automated installations of the Oracle Solaris OS on x86 clients
SPARC AI ISO image	`/usr/share/distro_const/dc_ai_sparc.xml`	Used to create a SPARC Automated Install ISO image for automated installations of the Oracle Solaris OS on SPARC clients

▼ How to Create and Build a Custom Image

This procedure describes the general steps to create and build a custom image.

1. **Install the `distribution-constructor` package, which contains the distribution constructor application and the sample manifests.**

   You can use the Package Manager tool to install the required package. The Package Manager is available on the menu bar on the desktop of the Oracle Solaris operating system. On the menu bar, go to System>Administration>Package Manager.

   Alternately, use IPS commands such as the following to install this package:

   ```
   # pkg install distribution-constructor
   ```

2. **Copy one of the sample manifests and create a custom manifest file with a new file name.**

   You will reference the manifest file by name when you use the `distro_const` command to create an image.

   Note – Always back up the original manifest file and the default scripts before copying them.

3. **Edit the manifest elements.**

   For example, you can edit the target element in the manifest to specify a different location of the build area where the image can be constructed. You can, also, check the publisher to ensure that your system can contact that publisher to download the packages needed to build the image. If necessary, you can edit the software name element to specify a different publisher and repository location.

   For information, see "Modifying the Manifest Content" on page 13 and the `dc_manifest`(4) man page.

4. **(Optional) Create custom scripts to further modify the image.**

   If you create new scripts, update the script references in the execution section of the manifest file.

   For instructions, see "Creating and Using Custom Scripts" on page 22.

5. **Run the `distro_const` utility to create an image.**

   For instructions, see Chapter 3, "Building an Image."

Modifying the Manifest Content

All the fields in each manifest file provide preset, default values that will create the type of ISO image you need. You can manually edit these preset fields in a manifest file to further customize the resulting image.

The following table describes the primary elements in the sample manifest files.

TABLE 2–3 Manifest Elements

Element	Description
`<distro name="Oracle_Solaris_Text_X86" add_timestamp="false">`	Specifies the image name with optional timestamp
`<boot_mods>`	Specifies GRUB menu modifications for the image
`<target>`	Defines the ZFS build dataset where the image is built
`<software name="transfer-ips-install" type="IPS">`	Specifies the source for the software packages to be installed
`<software_data action="install">`	Lists the packages to be installed
`<software_data action="uninstall">`	Lists the packages to be uninstalled
`<software name="set-ips-attributes">`	Sets different attributes for the IPS after the installation has finished
`<software name="ba-init">`	Specifies the boot archive contents **Caution –** Modify with care. If the boot archive is incorrect, the installed system will fail to boot.
`<execution stop_on_error="true">` `<checkpoint name="transfer-ips-install"/>`	Lists build checkpoints
`<configuration name="pre-pkg-img-mod" type="sysconf" source="/etc/svc/profile/generic_limited_net.xml">`	Specifies SMF services to be applied to the media during the build **Caution –** Rarely modify.

Provide the Image Title

Use the following element to provide a custom or default name for the image you are going to build:

```
<distro name="Oracle_Solaris_Text_X86" add_timestamp="false">
```

If you intend to perform a series of builds of an image and retain the incremental images, you can change the timestamp variable to “true” and a timestamp will be automatically appended to the name for each image.

If you need to specify an HTTP proxy, uncomment the `distro name` element that includes the proxy variable, and provide the proxy location.

Modify the Boot Menu

This boot menu element specifies boot menu modifications to be applied to the image.

In the following example, a specialized boot menu with the title “boot1” will be applied to the image. The timeout attribute specifics the time before the default boot entry is automatically activated.

```
<boot_mods title="boot1" timeout="5">
```

Within the boot menu element, you can add individual boot menu entries by adding a new `boot_entry` element for each new entry. Entries are added sequentially to the boot menu in the order based on the `insert_at` attribute value of “start” or “end” for each boot entry.

Note – Add new entries before the existing “with magnifier” entry.

See the following example of an individual `boot_entry` element.

```
<boot_entry>
   <title_suffix>with screen reader</title_suffix>
   <kernel_args>-B assistive_tech=reader</kernel_args>
</boot_entry>
```

For detailed information, see the `dc_manifest`(4) man page.

Specify the Build Area

You can customize the `target` element. This element defines the ZFS build dataset to be used for the build. This dataset is the area where the image will be created. You must provide a valid dataset location. You should check the default build area to ensure that the build will not destroy content you need to keep on your system. Modify the build area if necessary.

Note – The file system name should not include the name of the zpool.

The following example shows a sample target element.

```
 <target>
   <logical>
     <zpool action="use_existing" name="rpool">
       <dataset>
         <filesystem name="dc/sample-dataset-location"
         action="preserve"/>
       </dataset>
     </zpool>
   </logical>
 </target>
```

Specify the Publisher

The following element specifies a publisher where the distribution constructor can get packages to download and use to build the image.

```
<software name="transfer-ips-install">
```

In the source element nested within this software name section, edit the publisher name and origin name elements to specify which publisher to use and where the package repository is located. The repository location could be an NFS path or a local directory. Multiple publishers can be listed. When the distribution constructor attempts to locate packages to install, publishers are searched in the order they are listed here.

If mirrors for a publisher need to be specified, uncomment and edit the mirror name element.

The following example shows a sample source element found within the software name element.

```
<source>
   <publisher name="publisher1">
      <origin name="http://example.oracle.com/primary-pub"/>
      <mirror name="mirror.example.com"/>
   </publisher>
   <publisher name="publisher2">
       <origin name="http://example2.com/dev/solaris"></origin>
   </publisher>
   <publisher name="publisher3.org">
       <origin name="http://example3.com/dev"></origin>
    </publisher>
</source>
```

For further information about using publishers, see *Adding and Updating Oracle Solaris 11.1 Software Packages*.

List the Packages to Install

The `software_data` element with the `install` attribute lists the set of packages to be installed in order to build a particular type of image, depending on which manifest you are using. For example, the `dc_livecd.xml` manifest lists the packages needed to build a Live Media image. Each name tag lists one package name or the name of a group package that contains many packages.

```
<software_data action="install">
   <name>pkg:/group/system/solaris-desktop</name>
   <name>pkg:/system/install/gui-install</name>
   <name>pkg:/system/install/media/internal</name>
</software_data>
```

If you have packages that you want to add to the image, append the package names by adding a name tag for each package.

By default, the most current package version available in the specified repository is installed. If another version is required, append the version number to the package reference using the following format:

```
<name>pkg:/group/system/solaris-desktop@0.5.11-0.build#</name>
```

Note – The Oracle Solaris release version on your system must be the same as the release version of the image that you intend to create with the distribution constructor.

In addition, packages with a particular version specified might not be installed if other packages with a conflicting version are being installed as specified by an automated install service's manifest file. See Chapter 9, "Customizing Installations," in *Installing Oracle Solaris 11.1 Systems*.

EXAMPLE 2–1 Adding Packages and Additional Publishers

In this example, a second publisher, `mypublisher`, is specified. Additional packages, `mypackage1` and `mypackage2`, are specified.

During the build process, the publishers are checked in the order they are listed. If packages are not found at the first publisher, the next publisher is searched for the specified packages.

```
<software name="transfer-ips-install" type="IPS">
   <destination>
      <xi:include xmlns:xi="http://www.w3.org/2003/XInclude"
            href="/usr/share/distro_const/lang_facets.xml"/>
   </destination>
   <source>
      <publisher name="solaris">
          <origin name="http://pkg.oracle.com/solaris/release"/>
      </publisher>
      <publisher name="mypublisher">
         <origin name="http://mypublisher.company.com"/>
      </publisher>
   </source>
   <software_data action="install">
        <name>pkg:/group/system/solaris-large-server</name>
        <name>pkg:/system/install/text-install</name>
        <name>pkg:/system/install/media/internal</name>
        <name>pkg:/mypackage1</name>
        <name>pkg:/mypackage2</name>
   </software_data>
</software>
```

List the Packages to Uninstall

The `software_data` element with the uninstall attribute can be used to uninstall an individual package or to uninstall a group package definition.

Note – A **group package definition** binds all the individual packages within that group together into one unit that can only be acted upon as a group.

The uninstall attribute is particularly useful if you want to install a full group package, but you want to omit one or more individual packages from that group. You can use the uninstall

attribute to, first, remove the group package definition. Then, you can uninstall individual packages that were installed as part of a group package

For example, you might have chosen to build a Live Media installation image. The default Live Media installation image includes a Firefox brower in the desktop group package.

If you want to omit the Firefox browser from the image you want to build, you would do the following:

1. Install the `solaris-desktop` group package that includes all the software for the usual Live Media desktop. See "List the Packages to Install" on page 16.
2. Uninstall the `solaris-desktop` group package definition by using the uninstall attribute as follows:

```
<software_data action="uninstall">
   <name>pkg:/group/system/solaris-desktop</name>
</software_data>
```

Note – The uninstall action on the group package uninstalls only the group package definition. All the individual packages within that group are still installed as per the first step.

---

3. Now that the individual packages are not bound into a group definition, you can use the uninstall attribute again to uninstall the Firefox package.

```
<software_data action="uninstall">
   <name>pkg:/web/browser/firefox</name>
</software_data>
```

Alternately, you can combine steps 2 and 3 in one entry as follows:

```
<software_data action="uninstall">
        <name>pkg:/group/system/solaris-desktop</name>
        <name>pkg:/web/browser/firefox</name>
</software_data>
```

Append additional packages to be uninstalled at the end of the uninstall section.

Specify the Publisher for an Installed System

The `software name` element affects a system after that system has been installed with the image created using the distribution constructor.

```
<software name="set-ips-attributes">
```

Provide the publisher name and optional mirror name tags to specify where the installed system can access additional packages to download and install.

You can also set IPS attributes in this element. See the `pkg`(1) man page IPS property information.

Set Up Build Checkpoints

The `execution` element in the manifest lists a series of checkpoints that are executed during the image construction process. Checkpoints are executed in the order they are listed in this section. The default checkpoints needed to build the default installation image are included in each manifest.

During the image construction process, the checkpoints modify the contents of the build area that is specified in the manifest.

The build area contains the following directories:

- *ZFS dataset*`/build_data/pkg_image`
- *ZFS dataset*`/build_data/boot_archive`

where the *ZFS dataset* variable is specified by the target element in the manifest.

During the build process, everything that will be included in the final image is added to the `pkg_image` directory. The files in the separate `boot_archive` directory are used during the build process to create a boot archive file which is also added to the `pkg_image` directory.

The following list provides a brief description of each default checkpoint in the order the checkpoints are executed in most manifests.

- `transfer-ips-install` – At this checkpoint, the distribution constructor contacts the IPS publishers and adds to the image the packages that are listed in the `software_data` element of the manifest.
- `set-ips-attributes` – At this checkpoint, the constructor sets the publisher to be used by the installed system. The values set by this checkpoint are not relevant if you are building an automated installation image.
- `pre-pkg-img-mod` – At this checkpoint, the constructor imports into the image the SMF service files that were specified in the `configuration` element of the manifest. Also, the constructor modifies some files to optimize the image.

  All changes up through this checkpoint are included in both the image being built and the root archive. You should add new checkpoints for custom scripts before or immediately after this `pre-pkg-img-mod` checkpoint if you want to ensure that changes from the custom scripts are incorporated in both the root archive and in the image.
- `ba-init` – At this checkpoint, the constructor populates the root archive with the files listed in the `ba-init` section of the manifest. These files are copied from the `pkg_image` area into the `root_archive` area.
- `ba-config` – At this checkpoint, the constructor performs further modifications to the files that were copied into the root archive. The constructor creates symbolic links to other files that are not needed until later in the boot process in order to minimize the size of the root archive.

- `ba-arch` – At this checkpoint, the constructor packs the root archive and creates the root archive as a file within the `pkg_image` directory. The constructor also applies any optimizations to the root archive specific to the type of system being built. After this checkpoint, changes to the boot archive specifications by custom scripts would not be integrated into the root archive because the root archive has already been packed.
- `grub-setup` – At this checkpoint, the constructor sets up the GRUB2 menu based on the entries specified in the `boot_entry` section of the manifest. This checkpoint applies only to images for x86 systems.
- `pkg-img-mod` – At this checkpoint, the constructor creates the main archives for the image being built and optimizes the `pkg_image` area. The constructor moves files in the `pkg_image` directory, creating the archive for the image. Everything included in the `pkg_image` directory is included in the image. Any additions after this checkpoint would not be included in the image.
- `create-iso` – This checkpoint builds the `.iso` files, including everything in the `pkg_image` directory.

Looking at the specific fields included in each checkpoint section, each checkpoint name tag includes the `mod-path` attribute which specifies where the checkpoint script is located.

Some of the default checkpoint tags include arguments with default values provided. The following checkpoint example from the `dc_ai_sparc.xml` sample manifest creates the boot archive for the image build and points to a script that will accomplish that task. The example checkpoint also includes argument fields with specific values provided for each argument.

```
<checkpoint name="ba-arch"
    desc="Boot Archive Archival"
    mod_path="solaris_install/distro_const/checkpoints/
    boot_archive_archive"
    checkpoint_class="BootArchiveArchive">
    <kwargs>
        <arg name="size_pad">0</arg>
        <arg name="bytes_per_inode">0</arg>
        <arglist name="uncompressed_files">
             <argitem>etc/svc/repository.db</argitem>
             <argitem>etc/name_to_major</argitem>
             <argitem>etc/minor_perm</argitem>
             <argitem>etc/driver_aliases</argitem>
             <argitem>etc/driver_classes</argitem>
             <argitem>etc/path_to_inst</argitem>
             <argitem>etc/default/init</argitem>
             <argitem>etc/nsswitch.conf</argitem>
             <argitem>etc/passwd</argitem>
             <argitem>etc/shadow</argitem>
             <argitem>etc/inet/hosts</argitem>
        </arglist>
    </kwargs>
</checkpoint>
```

As shown in this example, the `kwargs` element contains keyword arguments that need to be passed into the checkpoint during the build. Within the `kwargs` element are `arg name` elements

that can be used to specify individual keywords to be passed into the checkpoint. And, the `arglist` element contains a list of multiple `argitem` values to be passed into the checkpoint. This example includes a list of uncompressed files in the `arglist` element.

Each `kargs` list item is enclosed in double quotes. When no double quotes are used, or if one set of double quotes encloses the entire string, the entire string including spaces and new lines is interpreted as one argument. Do not use commas between arguments.

If you create a custom script to be used during the building of an image, you must add a checkpoint element pointing to the script location. The checkpoint for a custom script needs only an `args` element that points to the custom script location. For further information and examples, see "Creating and Using Custom Scripts" on page 22.

Use the `distro_const` command options to control pausing and restarting the build process at particular checkpoints. See "How to Build an Image in Stages" on page 26.

EXAMPLE 2–2 Adding SVR4 Packages

In this example, a new checkpoint is added to the manifest. This new checkpoint lists SVR4 packages to be added to the image and their location. Then, this new checkpoint is referenced in the execution section.

First, the new checkpoint is created by adding a new `software` element. This checkpoint specifies SVR4 as the software type, where to find the packages, and where to install the packages.

In addition, the specific SVR4 packages to be installed are listed in the `software_data` element.

```
<software name=transfer-svr4-install type="SVR4">
   <destination>
       <dir path={PKG_IMAGE_PATH}/>
   </destination>
   <source>
    <dir path="/path/to/packages"/>
   </source>
   <software_data action="install">
      <name>SUNWpackage1</name>
      <name>SUNWpackage2</name>
   </software_data>
</software>
```

If included in the checkpoint, the values of `{PKG_IMAGE_PATH}` and `{BOOT_ARCHIVE}` are replaced by the `distro_const` utility with *ZFS dataset*`/build_data/pkg_image` and *ZFS dataset*`/build_data/boot_archive`, respectively. In this example, the SVR4 packages will be installed into *ZFS dataset*`/build_data/pkg_image`.

Finally, the new checkpoint is referenced in the execution section.

```
<execution stop_on_error="true">
   <checkpoint name="transfer-ips-install"
       desc="Transfer pkg contents from IPS"
```

EXAMPLE 2–2 Adding SVR4 Packages *(Continued)*

```
      mod_path="solaris_install/transfer/ips"
      checkpoint_class="TransferIPS"/>
  <checkpoint name="set-ips-attributes"
      desc="Set post-install IPS attributes"
      mod_path="solaris_install/transfer/ips"
      checkpoint_class="TransferIPS"/>
  <checkpoint name="transfer-svr4-install"
     desc="Transfer pkg contents from SVR4 packages"
     mod_path="solaris_install/transfer/svr4"
     checkpoint_class="TransferSVR4"/>
```

Note that the software name must match the checkpoint name. In this example, both are "transfer-svr4–install."

Creating and Using Custom Scripts

The distribution constructor enables you to specify additional scripts that can be used to make customizations during the image creation process based on the type of image you are building. The manifest files point to the scripts, and the scripts transform the generic image into a media-specific distribution. These scripts are referenced in the execution section of the manifest files. Any number of custom-script checkpoints may be specified.

Note – Support for scripts is limited to any unmodified default scripts that are supplied with the application packages. If you choose to customize these scripts, back up the original scripts first.

In addition, note that scripts specified in the execution section of the manifest file are run during the image creation process. The execution section does not reference pre-install or post-install scripts.

When you create your own custom scripts, note the following:

- Scripts can be Python programs, shell scripts, or binaries.
- Scripts are executed in the order that they are listed in the execution section of the manifest file.
- Standard output (`stdout`) and error output (`stderr`) of commands executed within the scripts (both shell and python modules) are captured in log files that report on the completed or attempted build.

How to Create and Use a Custom Script

1. **Create your new script.**

2. **Add your new scripts to your home directory or elsewhere on the system or network.**

   Make sure that a user assuming the root role can execute these scripts.

3. **Reference the new script by adding a checkpoint in the execution section of the appropriate manifest file.**

   To decide where to a the new checkpoint, review the descriptions of the default checkpoints as described in "Set Up Build Checkpoints" on page 19.

   Be sure to specify the full path to your scripts. Checkpoints are executed in the order they are listed in the execution section of the manifest.

   When you add a reference for a new script in the execution section of a manifest file, you must specify a checkpoint name that can be used to pause the image build before or after this script performs its task. Optionally, you can include a custom message associated with the checkpoint name. If this message is omitted, the path of the script is used as the default checkpoint message. The checkpoint message displays when the checkpoint is run during the build process.

   Note – Use meaningful names for checkpoint names instead of using numbers. If new scripts are added, the new checkpoints for those new scripts will disrupt a numbered checkpoint order.

   The following example checkpoint references a custom script named "my-script."

```
<checkpoint name="my-script"
        desc="my new script"
        mod_path="solaris_install/distro_const/checkpoints/custom_script"
        checkpoint_class="CustomScript">
        <args>/tmp/myscript.sh</args>
</checkpoint>
```

4. **(Optional) Specify a build parameter as part of the checkpoint as follows.**

   Here `{PKG_IMAGE_PATH}` is specified as the build parameter in the arguments section.

```
<checkpoint name="my-script"
        desc="my new script"
        mod_path="solaris_install/distro_const/checkpoints/my_script"
        checkpoint_class="CustomScript">
        <args>/tmp/myscript.sh {PKG_IMAGE_PATH}</args>
</checkpoint>
```

   If included in the checkpoint, the values of `{PKG_IMAGE_PATH}` and `{BOOT_ARCHIVE}` are replaced by the `distro_const` utility with *ZFS dataset*`/build_data/pkg_image` and *ZFS dataset*`/build_data/boot_archive`, respectively.

5 **Build the image.**

You can build the image in one step. Or, to check the status of the build, you can stop and restart the build at various checkpoints.

For instructions, see Chapter 3, "Building an Image."

6 **(Optional) After the build is complete, you can view a log file reporting on the build process.**

The build output displays the location of log files.

◆ ◆ ◆ CHAPTER 3

Building an Image

After you have set up the manifest file that you plan to use and, if desired, customized the finalizer scripts, you are ready to build an image by running the `distro_const` command.

You can use the `distro_const` command to build an image in one step. Or, you can use and restart the build as needed to examine the content of the image and debug the scripts during the build process.

`distro_const` Command

The full syntax for the `distro_const` command is as follows:

`distro_const build [-v] [-r` *checkpoint*`] [-p` *checkpoint*`] [-l]` *manifest*

The `distro_const` command options are described in the following table.

TABLE 3–1 `distro_const` Command Options

Command Options	Description
`distro_const build` *manifest*	Builds an image in one step using specified manifest file
`distro_const build -v` *manifest*	Verbose mode
`distro_const build -l` *manifest*	Lists all valid checkpoints at which you can pause and resume building an image
`distro_const build -p` *checkpoint manifest*	Pauses building an image at a specified checkpoint
`distro_const build -r` *checkpoint manifest*	Resumes building an image from a specified checkpoint
`distro_const build -h`	Displays help for the command

Note – You must assume the root role to use the `distro_const` command.

How to Build an Image in One Step

1. **Download the `distribution-constructor` package.**
2. **Select a manifest for your image.**
3. **(Optional) If needed, customize the manifest, adding references to your custom scripts.**
4. **Become the root role.**
5. **Issue the basic `distro_const` command without options.**

   `# distro_const build` *manifest*`.xml`

   Replace *manifest* with the name of the manifest file to be used as the blueprint for your image.

   For example:

   ```
   # distro_const build /usr/share/distro_const/dc_livecd.xml
   ```

   The distribution constructor pulls the needed packages for the image, and builds the image to the specifications that you set up in the manifest file.
6. **(Optional) After the build is complete, you can view a log file reporting on the build process.**

   The build output displays the location of log files.

How to Build an Image in Stages

You can use the options provided in the `distro_const` command to stop and restart the build process at various checkpoints in the image-generation process, in order to check and debug your selection of files, packages, and scripts for the image that is being built.

1. **Download the `distribution-constructor` package.**
2. **Select manifest for your image.**
3. **(Optional) If needed, customize the manifest, adding references to your custom scripts.**
4. **Become the root role.**

5 Review the valid checkpoints at which you can choose to pause or resume the build.

```
# distro_const build -l manifest.xml
```

This command displays the valid checkpoints at which you can pause or resume building an image. Use the checkpoint names provided by this command as valid values for the other checkpointing command options.

For example, the following command confirms which checkpoints are available for a manifest file named `dc_livecd.xml`.

```
# distro_const build -l /usr/share/distro_const/dc_livecd.xml

Checkpoint        Resumable Description
--------------    --------- -------------
transfer-ips-install X Transfer package contents from IPS
set-ips-attributes   X Set post-installation IPS attributes
pre-pkg-img-mod      X Pre-package image modification
ba-init              X Boot archive initialization
ba-config            X Boot archive configuration
ba-arch              X Boot archive archiving
grub-setup             Set up the GRUB menu
pkg-img-mod            Package image area modifications
create-iso             ISO image creation
```

Note – In this sample command output, an "X" in the resumable field indicates that you can restart the build from this checkpoint.

6 Build the image and pause building the image at the specified checkpoint.

```
# distro_const build -p checkpoint manifest
```

For example, the following command starts building an image and pauses the build before `ba-arch` modifies the image area:

```
# distro_const build -p ba-arch /usr/share/distro_const/dc_livecd.xml
```

7 Resume building the image from a specified checkpoint.

```
# distro_const build -r checkpoint manifest
```

Note – The specified checkpoint must be either the checkpoint at which the previous build stopped executing, or an earlier checkpoint. A later checkpoint is not valid.

For example, the following command resumes building the image at the `ba-arch` stage.

```
# distro_const build -r ba-arch /usr/share/distro_const/dc_livecd.xml
```

Note – You can combine the pause and resume options in a `build` command.

8 **(Optional) After the build is complete, you can view a log file reporting on the build process.**
The build output displays the location of log files.

Index

U