Creating a Custom Oracle® Solaris 11.3 Installation Image

Part No: E54748
December 2017

ORACLE

Creating a Custom Oracle Solaris 11.3 Installation Image

Part No: E54748

Référence: E54748

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Using This Documentation

- **Overview** – Describes how to build custom Oracle Solaris installation packages using the distribution constructor tool
- **Audience** – Technicians, system administrators, and authorized service providers
- **Required knowledge** – Experience administering an Oracle Solaris system

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394-01.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ 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 9.
- If you are ready to build custom images, go to "System Requirements for Building Images" on page 13.

About the Distribution Constructor

The distribution constructor (DC) 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, also known as a disc image, which is an archive file 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.

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. You must use the `usbcopy` command to copy the USB image into a USB flash drive. The `usbcopy` command is available in the `distribution-constructor` package.

Note the following:

- Depending on the image configuration, ISO or USB images can be bootable.
- You can install ISO images and USB images on a system or run them 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.

Oracle Solaris Installation 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 distributed with each Oracle Solaris release. You can also customize the content of this ISO image, for example, by adding or removing packages. In addition, 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 11.3 Systems*. For more information about customizing the image content, see "Modifying the Manifest Content" on page 14.
- **Oracle Solaris x86 or SPARC Text Installation Image** – You can create a SPARC or x86 ISO image that you can use to perform a text installation of the Oracle Solaris operating system. You can use the text installer on systems that do not have 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 11.3 Systems*.
- **x86 or SPARC ISO Image for Automated Installations** – The Oracle Solaris operating system includes the automated installer (AI) tools, which automates the installation of the Oracle Solaris OS on one or more 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 *Installing Oracle Solaris 11.3 Systems*.

Image Creation Process

The distribution constructor creates images based on settings specified in XML files, called *manifest files*. The DC manifest files contain specifications for the contents and parameters of the ISO images that you create using the distribution constructor. The distribution constructor

package contains sample manifests that you can use 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 DC Manifest Files” on page 14.

All the fields in each DC 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 DC manifest to specify a different location for the build area where the image can be constructed.
- You can check the specified publisher 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 “Modifying the Manifest Content” on page 14.

You can also create *custom scripts* to modify your installation image. Then, you can add checkpoints to the DC manifest file to run these custom scripts. For further information, see “Creating and Using Custom Scripts When Creating an Installation Image” on page 26.

The distribution constructor package also includes the `distro_const` command, that interprets the DC 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 DC manifest file.

After you run the `distro_const` command, 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 30, or see the `distro_const`(1M) man page.

♦ ♦ ♦ CHAPTER 2

Design a Custom Installation Image

This chapter provides system requirements for building images and describes how to design a custom installation image by creating a DC manifest and scripts. It includes the following topics:

System Requirements for Building Images

In order to use the distribution constructor, your system must meet the system requirements described in the following table.

TABLE 1 System Requirements

Requirement	Description
Distribution constructor workspace disk space	8 Gbytes
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 must be able to access Image Packaging System (IPS) repositories 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. ■ 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.

Requirement	Description
Required packages	The `distribution-constructor` package, which contains the distribution constructor tool.

Sample DC Manifest Files

The `distribution-constructor` package provides the sample manifest files described in the following table. All files are installed in `/usr/share/distro_const`.

TABLE 2 Sample DC Manifests

Manifest Type	Manifest Location	Description
x86 Live Media ISO image	`dc_livecd.xml`	Used to create an x86 ISO image comparable to the Oracle Solaris Live Media image
x86 text installation image	`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	`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	`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	`dc_ai_sparc.xml`	Used to create a SPARC Automated Install ISO image for automated installations of the Oracle Solaris OS on SPARC clients

Environment Variables in DC Manifests

The following environment variables can be used in a DC manifest.

- `PKG_IMAGE_PATH` - during the installation image build process this variable is resolved to the path to the package image.
- `BOOT_ARCHIVE` - during the installation image build process this variable is resolved to the path to the installation image. For instance the path to the `/etc/system` file in the archive would be `{BOOT_ARCHIVE}/etc/system`.

Modifying the Manifest Content

All the fields in each DC 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 section lists the primary elements in the sample manifest files and a description of those elements.

- The `distro` element defines the name of the image you are going to build, as well as the ability to provide an HTTP proxy. See "Provide the Image Title" on page 16.
- The `boot_mods` element defines boot menu options, which boot entry is the default, and how long to wait before the default entry is booted. See "Provide the Image Title" on page 16.
- The `target` element defines the ZFS dataset that will hold the installation image when it is created. See "Specify the Build Area" on page 17.
- The `software` element defines the publisher for both the installation image and the install client, as well as the packages to be installed or uninstalled. See "Specify the Build Area" on page 17.
- The `execution` element defines the checkpoints that are executed when building the installation image. You can add your own checkpoints. See "Set Up Build Checkpoints" on page 21.

▼ Selecting a Manifest to Use when Building an Installation Image

This procedure describes the general steps to install the generic DC manifests, which can then be used to create customized manifests.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

2. **Install the `distribution-constructor` package.**

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

3. **Review the contents of the manifest files.**

   "Sample DC Manifest Files" on page 14 gives a general description of the default manifests included in the package.

4. **Make a copy of one of the sample manifests.**

   ```
   # cp /usr/share/distro_const/dc_livecd.xml /var/tmp/test.xml
   ```

Next Steps After this you can edit the new manifest as needed, including adding any custom scripts as described in the following sections. After that you can create the image as described in Chapter 3, “Building an Image”.

Provide the Image Title

Use the `distro name` element to provide a custom or default name for the image you are going to build. For example:

```
<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

The `boot_mods` 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 specifies the time before the default boot entry is automatically activated.

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

Within the `boot_mods` 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 indicated by the `insert_at` attribute value of `start` or `end` for each boot entry.

Note - Add new entries before the existing `with magnifier` entry.

The following example shows a sample 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, which defines the ZFS build dataset where the image will be created. You must provide a valid dataset location. Check the default build area to make sure that the build will not destroy content that you need to keep on your system. The path to the dataset is relative to the pool name defined. In the example below, the dataset would be installed as `rpool/dc/sample-dataset-location`.

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 Software Sources

The `software` element with the `name` attribute set to `transfer-ips-install` includes sections that define the publishers that are used when the installation image is created, as well as a list of the packages to be installed or uninstalled in the image. The `software` element with `name` element set to `set-ips-attributes` attribute defines which publishers the installed client will be configured to use. For more information see:

- “Specify the Publisher Used when Building the Installation Image” on page 18.
- “List the Packages to Install” on page 18.
- “List the Packages to Uninstall” on page 20.

- "Specify the Publisher for an Installed System" on page 21.

Specify the Publisher Used when Building the Installation Image

In the `source` element nested within the `transfer-ips-install` 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. You can list multiple publishers. When the distribution constructor attempts to locate packages to install, it searches publishers in the order in which they are listed in this element.

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

The following example shows a sample source 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"/>
   </publisher>
   <publisher name="publisher3.org">
       <origin name="http://example3.com/dev"/>
   </publisher>
   <publisher name="publisher4">
       <origin name="file:///net/myserver/publisher4/repo"/>
   </publisher>
</source>
```

For further information about using publishers, see *Adding and Updating Software in Oracle Solaris 11.3*.

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 element 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>
```

To add packages to the image, add a name element for each package.

By default, the most current package version available in the specified repository is installed. If you want to install a different version, 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 - If you specify a specific version of a package but another package with a conflicting version is being installed as specified by an automated install service's manifest file, the package might not be installed. See Chapter 9, “Assigning Customizations to AI Clients” in *Installing Oracle Solaris 11.3 Systems*.

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.

EXAMPLE 1 Adding Packages and Additional Publishers

In this example, a second publisher, `mypublisher`, and 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

You can use the `software_data` element with the `uninstall` attribute to uninstall an individual package or a group package definition.

Note - A *group package definition* binds all the individual packages within that group into one unit that can be acted upon only 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 remove the group package definition and then uninstall individual packages that were installed as part of the group package.

For example, the default Live Media installation image includes a Firefox browser in the desktop group package. To omit the Firefox browser from an 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 18.
2. Uninstall the `solaris-desktop` group package definition.

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

   This action uninstalls only the group package definition, which means that the individual packages are no longer bound into a group definition. All the individual packages within that group are still installed.
3. Use the `uninstall` attribute again to uninstall the Firefox package.

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

Tip - You can combine steps 2 and 3 into one entry as follows:

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

Append any additional packages to be uninstalled at the end of the uninstall section of the `software_data` element.

Specify the Publisher for an Installed System

The `source` element nested within the `transfer-ips-install` attribute affects a system after that system has been installed with the image created by using the distribution constructor. It specifies information such as the publisher names and optional mirror names to specify where the installed system can access additional packages.

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

Set Up Build Checkpoints

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

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

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 30.

The build area contains a `pkg_image` and a `boot_archive` directory. 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 brief descriptions of each default checkpoint are presented in the order in which 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` element with the `transfer-ips-install` attribute.
- `set-ips-attributes` – At this checkpoint, the constructor sets the publisher to be used by the installed system. These values are set in the `software` element with the `set-ips-attributes` attribute. The values set by this checkpoint are not relevant if you are building an automated installation image. An AI client is configured to use the publisher of the AI server.
- `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 package related operations and checkpoints must precede this checkpoint.
- `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.

  All changes made before this checkpoint are included in both the image being built and the root archive. You should add new checkpoints for custom scripts before this checkpoint if you want to ensure that changes from the custom scripts are incorporated in both the root archive and the image.
- `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 are not integrated into the root archive because the root archive has already been packed.
- `boot-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 are not included in the image.

- `create-iso` – This checkpoint builds the `.iso` files, including everything in the `pkg_image` directory.
- `create-usb` – This checkpoint builds the `.usb` file from the generated `.iso` file.

Each checkpoint element includes the `mod-path` attribute that specifies where the checkpoint script is located. Also, some of the default checkpoints 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 build the image. It 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. 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 `kwargs` list item is enclosed in double quotes. If 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 When Creating an Installation Image" on page 26.

EXAMPLE 2 Adding SVR4 Packages to An Installation Image

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. This new checkpoint then 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>
    <publisher/>
      <origin name="path-to-packages"/>
    </publisher>
   </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` command with the path to the build area for the package image and the boot archive, respectively. In this example, the SVR4 packages will be installed into the package image directory.

Finally, the new checkpoint is referenced in the execution section. The checkpoint name can be any string, but for this example, the `checkpoint_class` value must be `TransferSVR4`.

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

EXAMPLE 3 Creating Hashes of the Media in an Installation Image

The `checksums` checkpoint enables users to automatically generate hashes of the media generated by the `distro_const` command.

```
<checkpoint name="checksums"
  desc="Checksum calculation for media"
  mod_path="solaris_install/distro_const/checkpoints/checksums"
  checkpoint_class="Checksums">
  <kwargs>
    <arglist name="algorithms">
      <argitem file_path="/tmp/md5sums.txt">md5</argitem>
      <argitem>sha1</argitem>
      <argitem>sha224</argitem>
      <argitem>sha256</argitem>
      <argitem>sha384</argitem>
      <argitem>sha512</argitem>
    </arglist>
  </kwargs>
</checkpoint>
```

The `arglist` element includes all of the algorithms that are used to generate hashes for the generated media. Each `argitem` specifies an algorithm. The valid algorithms can be determined by running the `/usr/bindigest -l` command. Each `argitem` can have a `path` attribute that specifies the absolute path of an additional file that will be appended with the hashes produced by that algorithm. If no algorithms are specified, the default is `md5`.

While the image is built, files will be generated for each algorithm containing checksums for each media.

Creating and Using Custom Scripts When Creating an Installation Image

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. You can specify any number of `custom-script` checkpoints.

Often custom scripts are used to modify a configuration file or make some other change that can not be done using a manifest.

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.

Note - Do not change scripts that are installed from packages. To prevent problems with future package updates, make any changes in a script you create.

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 in which 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 When Creating an Installation Image

1. **Create a new script.**
2. **Add the new script to your home directory or elsewhere on the system or network.**

   Make sure that a user assuming the `root` role can execute the script.
3. **Modify the manifest.**

Add information referencing the new script in the `execution` section of the manifest. To decide where to add the new checkpoint, review "Set Up Build Checkpoints" on page 21.

Be sure to specify the full path to your scripts. Checkpoints are executed in the order in which 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 the 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 rather than ordinal numbers. If you use numbers, adding new checkpoints for new scripts will disrupt the 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. **Build the image.**

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

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

5. **(Optional) After the build is complete, view the log file for the build process.**

   The build output displays the location of the log files.

Example 4 Using Environment Variables in a Checkpoint

In the following example, the image directory path is used as an argument to `myscript.sh`.

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

Example 5 Including a Short Script in a Custom DC Manifest

The following custom script prevents anyone from logging in as the user called `jack`. You could also change it to add a password, or remove the line. Note that this checkpoint was added at a specific place in the DC manifest. The script must run after the packages have been laid down, but before the boot_archive is created.

```
     <checkpoint name="set-ips-attributes"
         desc="Set post-install IPS attributes"
         mod_path="solaris_install/transfer/ips"
         checkpoint_class="TransferIPS"/>
     </checkpoint>
     <checkpoint name="lock-jack-account"
         desc="Lock the jack account from login"
         mod_path="solaris_install/distro_const/checkpoints/custom_script"
         checkpoint_class="CustomScript">
         <args>sed 's/jack:.[^:]*:/jack:*LK*:/g' {PKG_IMAGE_PATH}/etc/shadow
> {PKG_IMAGE_PATH}/etc/shadow.new; cp {PKG_IMAGE_PATH}/etc/shadow.new
{PKG_IMAGE_PATH}/etc/shadow; rm {PKG_IMAGE_PATH}/etc/shadow.new</args>
     </checkpoint>
     <checkpoint name="pre-pkg-img-mod"
```

The following script will set the password for the `solaris` user which can be used to access an install client during the installation process.

```
     <checkpoint name="set-ips-attributes"
         desc="Set post-install IPS attributes"
         mod_path="solaris_install/transfer/ips"
         checkpoint_class="TransferIPS"/>
     </checkpoint>
     <checkpoint name="solaris-password"
         desc="Set the password for the solaris account used during the installation
 process"
         mod_path="solaris_install/distro_const/checkpoints/custom_script"
         checkpoint_class="CustomScript">
         <args>sed 's/solaris:.[^:]*:/solaris:string:/g' {PKG_IMAGE_PATH}/etc/shadow
> {PKG_IMAGE_PATH}/etc/shadow.new; cp {PKG_IMAGE_PATH}/etc/shadow.new
{PKG_IMAGE_PATH}/etc/shadow; rm {PKG_IMAGE_PATH}/etc/shadow.new</args>
     </checkpoint>
     <checkpoint name="pre-pkg-img-mod"
```

♦ ♦ ♦ 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 stop and restart the build as needed to examine the content of the image and debug the scripts during the build process.

This chapter includes the following procedures:

`distro_const` Command

The following table describes the `distro_const` command options.

TABLE 3 `distro_const` Command Options

Command Options	Description
`distro_const build` *manifest*	Builds an image in one step using the 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.

Building an Installation Image

This section includes the following procedures:

▼ How to Build an Image in One Step

Before You Begin You need to have selected a manifest to use before building an installation image. See "Selecting a Manifest to Use when Building an Installation Image" on page 15.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

2. **Build the installation image.**

   ```
   # distro_const build /var/tmp/test.xml
   ```

   The distribution constructor pulls the needed packages for the image and builds the image to the specifications that you set up in the DC manifest file.

3. **(Optional) View the log file for the build process.**

   The build output displays the location of log files.

▼ How to Build an Image in Stages

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

Before You Begin You need to have selected a manifest to use before building an installation image. See "Selecting a Manifest to Use when Building an Installation Image" on page 15.

1. **Become an administrator.**

   For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

2. **Display the valid checkpoints at which you can choose to pause or resume the build.**

```
# distro_const build -l /var/tmp/test.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
boot-setup                      Setup LiveCD boot menu
pkg-img-mod                     Package image area modifications
create-iso                      ISO image creation
create-usb                      USB image creation
```

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

3. **Build the image and pause building the image at the specified checkpoint.**

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

```
# distro_const build -p ba-arch /var/tmp/test.xml
```

4. **Resume building the image from a specified checkpoint.**

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.

The following command resumes building the image at the `ba-arch` stage.

```
# distro_const build -r ba-arch /var/tmp/test.xml
```

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

5. **(Optional) View the log file for the build process.**

The build output displays the location of log files.

Index

M

N

O

P

R

S

T

U

Z