Oracle® Solaris 11 Installation Man Pages

Part No: E21797
March, 2012, E21797-03

ORACLE

120306@25097

Contents

Preface

This document provides the man pages for the Oracle Solaris 11 system installation tools.

Overview

The following contains a brief description of each man page section and the information it references:

- Section 1 describes commands available with the operating system.
- Section 1M describes commands that are used chiefly for system maintenance and administration purposes.
- Section 5 contains miscellaneous documentation such as character-set tables.

Below is a generic format for man pages. The man pages of each manual section generally follow this order, but include only needed headings. For example, if there are no bugs to report, there is no BUGS section. See the `man` command for more information about man pages in general.

NAME — This section gives the names of the commands or functions documented, followed by a brief description of what they do.

SYNOPSIS — This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.

. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename...`".
|	Separator. Only one of the arguments separated by this character can be specified at a time.
{ }	Braces. The options and/or arguments enclosed within braces are interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL	This section occurs only in subsection 3R to indicate the protocol description file.
DESCRIPTION	This section defines the functionality and behavior of the service. Thus it describes concisely what the command does. It does not discuss OPTIONS or cite EXAMPLES. Interactive commands, subcommands, requests, macros, and functions are described under USAGE.
OPTIONS	This section lists the command options with a concise summary of what each option does. The options are listed literally and in the order they appear in the SYNOPSIS section. Possible arguments to options are discussed under the option, and where appropriate, default values are supplied.
OPERANDS	This section lists the command operands and describes how they affect the actions of the command.
OUTPUT	This section describes the output – standard output, standard error, or output files – generated by the command.
RETURN VALUES	If the man page documents functions that return values, this section lists these values and describes the conditions under which they are returned. If a function can return only constant values, such as 0 or –1, these values are listed in tagged paragraphs. Otherwise, a single paragraph describes the return values of each function. Functions declared void do not return values, so they are not discussed in RETURN VALUES.
ERRORS	On failure, most functions place an error code in the global variable `errno` indicating why they failed. This section lists

	alphabetically all error codes a function can generate and describes the conditions that cause each error. When more than one condition can cause the same error, each condition is described in a separate paragraph under the error code.
USAGE	This section lists special rules, features, and commands that require in-depth explanations. The subsections listed here are used to explain built-in functionality: Commands Modifiers Variables Expressions Input Grammar
EXAMPLES	This section provides examples of usage or of how to use a command or function. Wherever possible a complete example including command-line entry and machine response is shown. Whenever an example is given, the prompt is shown as `example%`, or if the user must be superuser, `example#`. Examples are followed by explanations, variable substitution rules, or returned values. Most examples illustrate concepts from the SYNOPSIS, DESCRIPTION, OPTIONS, and USAGE sections.
ENVIRONMENT VARIABLES	This section lists any environment variables that the command or function affects, followed by a brief description of the effect.
EXIT STATUS	This section lists the values the command returns to the calling program or shell and the conditions that cause these values to be returned. Usually, zero is returned for successful completion, and values other than zero for various error conditions.
FILES	This section lists all file names referred to by the man page, files of interest, and files created or required by commands. Each is followed by a descriptive summary or explanation.
ATTRIBUTES	This section lists characteristics of commands, utilities, and device drivers by defining the attribute type and its corresponding value. See the `attributes`(5) man page for more information.

SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

REFERENCE

System Administration Commands

Name aimanifest – Modify an XML file used by Automated Installer (AI)

Synopsis

```
/usr/bin/aimanifest [-h]

aimanifest add [-r] path value

aimanifest get [-r] path

aimanifest set [-r] path value

aimanifest load [-i] filename

aimanifest validate
```

Description The `aimanifest` command creates a new XML manifest or modifies an existing one. While `aimanifest` can be used with any XML file that contains a valid `!DOCTYPE` reference to a DTD definition, it is intended for creating derived manifests used by the Automated Installer (AI). See *Installing Oracle Solaris 11 Systems* for information about AI derived manifests.

The `aimanifest` command can be invoked multiple times to develop a manifest. The `AIM_MANIFEST` environment variable specifies the location of the manifest for `aimanifest` to modify. `AIM_MANIFEST` must be set. Each invocation of the `aimanifest` command with the `load`, `add`, or `set` subcommand opens, modifies, and saves the `AIM_MANIFEST` file.

The minimum `AIM_MANIFEST` file that the `aimanifest` command can modify must contain both of the following pieces:

- A `!DOCTYPE` reference to a DTD that is valid for the XML manifest being developed.
- The root element for this manifest.

If you start with an empty `AIM_MANIFEST` file, as when AI is executing a derived manifests script, then the first `aimanifest` command must specify the `load` subcommand to load at least the minimum required `AIM_MANIFEST` file. Subsequent `aimanifest` commands that modify the manifest use the DTD to determine where to add elements in the developing manifest.

To save error and informational messages to a file in addition to displaying messages to `stdout` and `stderr`, set the `AIM_LOGFILE` environment variable to a log file location. Information is appended to the log file. The log file is not cleared.

Options The `aimanifest` command has the following option:

`-h`, `--help` Show the usage help message.

The `add`, `get`, and `set` subcommands of the `aimanifest` command have the following option:

`-r`, `--return-path` Return the path of the XML element that this `aimanifest` command creates or operates on. This returned path is a chain of node IDs. You can save this returned path value to use in subsequent calls to `aimanifest`. Using the path returned by the `-r` option is more reliable than specifying the path using XML element and attribute values, since the values can change as the AI manifest is being built. See the

"Return Paths" section for more information about the path returned by the `-r` option.

The `load` subcommand of the `aimanifest` command has the following option:

`-i`, `--incremental` Do not clear the `AIM_MANIFEST` data before adding new data.

Sub-commands The following subcommands are supported:

`add [-r | --return-path]` *path value*

Add a new element to an XML manifest. Add the new element at *path* and with value *value*. See the "Operands" section for more information about *path*. If *path* ends in an attribute (@*attr*), then the new element has the *attr* attribute, and *value* is the value of the attribute.

No validation is performed except to examine parent/child relationships in *path*.

The `-r` option returns a path to the newly-added node. See the "Return Paths" section for more information.

If the parent path matches an element in the `AIM_MANIFEST` file, it must match only one element. The new element is created as a child of the matching parent element. The path can specify element and attribute values to match a unique parent element, as shown in "Example 2: Path With a Value" in this section.

If the parent path does not match an element in the `AIM_MANIFEST` file, new elements are created as necessary, and the new child element is added to the new parent. The path to an added element is split off from the preexisting elements according to the following rules:

- The split occurs after all parts of the path that specify a value.
- The split occurs at the first place where multiple relevant same-tagged elements are allowed by the DTD, after all parts of the path that specify a value.

Use this XML manifest schema to analyze the following examples:

- The manifest begins with a single A node.
- The A node can have only one B node child.
- The B node can have multiple C node children.
- A C node can have multiple D node children.

Example 1: Simple Path. The AI manifest has one A node, one B node, and one C node: `/A/B/C`. An `add` subcommand is issued with a *path* of `/A/B/C/D`. In this case, a new C node is created because C nodes are the first nodes along the path that can have same-tagged siblings. A new D node is added as a child to the new C node. The resulting manifest has the structure `/A/B/{C,C/D}`. Issuing the same command for a different value of D results in three C nodes: `/A/B/{C,C/D,C/D}`.

Example 2: Path With a Value. The AI manifest has one A node, one B node, and two C nodes. Only one of the C nodes has a value of 1 so that the manifest has the structure `/A/B/{C,C=1}`. An `add` subcommand is issued with a *path* of `/A/B/C=1/D` and a *value* of 10.

In this case, no new C node is added because specifying the value of 1 for C identifies a unique node, and the path cannot be split at or before a branch where a value is specified. The first place where this path can be split is at D. A new D node with a value of 10 is added as a child of the C node that has a value of 1. The resulting manifest has the structure `/A/B/{C,C=1/D=10}`. Issuing the same command with a value of 20 for D results in `/A/B/{C,C=1/{D=10,D=20}}`.

`get [-r | --return-path]` *path*
Retrieve an element or attribute value. An empty string ("") is displayed for empty element or attribute values. The *path* must match a unique existing element or attribute. See the "Operands" section for more information about *path*.

The `-r` option returns a path to the accessed node as a second returned string. See the "Return Paths" section for more information.

`set [-r | --return-path]` *path value*
Change the value of an existing element or attribute, or create a new attribute of an existing element. No validation is performed.

When changing the value of an existing element, *path* must match a unique existing element. If the element has same-tagged siblings, use an element value or attribute, or a child element of the target element to make the path unique. See "The Path Operand" section.

When setting the value of an attribute, the attribute does not need to exist, but the element to which the attribute belongs must exist.

The `-r` option returns a path to the changed element. See the "Return Paths" section for more information.

`load [-i | --incremental]` *filename*
Load an XML manifest or partial XML manifest from the file *filename*. No validation is performed except to examine parent/child relationships of elements.

When the `-i` option is not specified, overwrite any existing XML data. All data in the `AIM_MANIFEST` file is replaced with the contents of the *filename* file. The *filename* file must include a `!DOCTYPE` reference to a DTD so that subsequent `aimanifest` commands can modify the file.

When the `-i` option is specified, do not clear the `AIM_MANIFEST` data before adding new data. Instead, incrementally insert or merge the new data with the existing XML data. The DTD given by the `!DOCTYPE` reference in `AIM_MANIFEST` is used to determine how and where to merge the *filename* data. If the `!DOCTYPE` reference is missing, the AI manifest DTD at `/usr/share/install/ai.dtd` is used. If the data in *filename* cannot be reconciled with the DTD, a non-zero error status is returned.

The following considerations affect where new data is inserted into the AIM_MANIFEST manifest:

- To what extent the tags of elements near the beginning of the AIM_MANIFEST data paths and *filename* data paths match
- What child elements are allowed under those AIM_MANIFEST data elements
- Where same-tagged sibling elements are allowed
- Where childless AIM_MANIFEST data nodes are located

As each element of *filename* data is processed, if all of the following conditions are true, then in general a new node is not created for this element in the AIM_MANIFEST data. Instead, an existing node is replaced with the new data.

- Both sets of data contain a node with the same tag and same location.
- The DTD given by the !DOCTYPE reference in AIM_MANIFEST does not allow both of these nodes to exist together as same-tagged sibling elements.
- The *filename* data element has children.

When an element from *filename* is inserted, the split where new nodes start to be created is done as close as possible to the AIM_MANIFEST data root. The first new node of the split is created at the earliest point where same-tagged sibling elements are allowed, or at the earliest appropriate point when no same-tagged element exists in AIM_MANIFEST.

Use this XML manifest schema to analyze the following examples:

- The manifest begins with a single A node.
- The A node can have only one B node child.
- The B node can have multiple C node children.
- The B node can have only one E node child.

Example 1: Inserting Same-Tagged Elements. If the content of AIM_MANIFEST is /A/B/C1/D1 and the content of *filename* is /A/B/C2/D2, then after the load -i command, the content of the AIM_MANIFEST file is /A/B/{C1/D1,C2/D2}. The C node is the first place where new nodes can be added. The C node from the *filename* data is added after the existing C node in the AIM_MANIFEST data. If the two A elements have different values or if the two B elements have different values, the value of the *filename* element replaces the value of the AIM_MANIFEST element. If the two A elements have different attributes, or if the two B elements have different attributes, the attribute values are merged.

- Attributes of A and B that exist in both the AIM_MANIFEST file and the *filename* file have the values from the *filename* file in the merged file.
- Attributes of A and B that exist in either the AIM_MANIFEST file or the *filename* file but not in both files are all retained in the merged file.

Example 2: Inserting Differently Tagged Elements. If the content of AIM_MANIFEST is /A/B/C/D and the content of *filename* is /A/B/E/F, then after the load -i command, the

content of the AIM_MANIFEST file is /A/B/{E/F,C/D}. The E node is added at the first location where it is allowed by the DTD. The values of elements A and B are the values from *filename*, and the attributes of A and B are merged from *filename* to AIM_MANIFEST as described in Example 1 above.

Sometimes the correct merge location cannot be determined. This can happen if a sibling that is required to follow a node to be merged has not yet been added. To avoid this issue, add multiple nodes or subtrees to a common parent node in the order mandated by the DTD. A node is placed at the end of its list of new siblings if its proper place among them cannot be determined.

validate
Validates the AIM_MANIFEST manifest against the DTD referenced in the !DOCTYPE statement. Errors are printed to stderr. A non-zero status is returned if validation fails.

Operands The following operands are required.

The Filename Operand The load subcommand requires the *filename* operand, which is the name of a full or partial manifest to load to the AIM_MANIFEST manifest.

The Value Operand The add and set subcommands require the *value* operand. The *value* operand is a valid value of the element or attribute specified by the *path* operand.

The Path Operand The add, get, and set subcommands of the aimanifest command require the *path* operand. The path defines a node in an XML hierarchy of elements and attributes.

The XML element hierarchy structure is also called an XML tree. In the following partial AI manifest, the auto_install element is the root of the tree, and the ai_instance and software elements are branches or the roots of subtrees.

```
<auto_install>
  <ai_instance>
    <software type="IPS"/>
  </ai_instance>
</auto_install>
```

In aimanifest path syntax, use forward slash characters (/) to indicate branches in the tree structure. In the current example, the path to the software element is /auto_install/ai_instance/software.

Attributes are bound to an element. In aimanifest path syntax, use an at symbol (@) to identify an attribute name. The path to the type attribute of the software element is /auto_install/ai_instance/software@type.

An aimanifest *path* operand must correspond to a single element. Include element and attribute values as necessary to make the path unique. For example, to specify a size for the second slice defined in the following partial AI manifest, you could use the path /auto_install/ai_instance/target/disk/slice[@name="4"]/size@val to identify which slice you are specifying the size for.

```
<auto_install>
  <ai_instance>
    <target>
        <disk>
          <slice name="0"/>
          <slice name="4"/>
        </disk>
    </target>
  </ai_instance>
</auto_install>
```

Relative paths are permitted. The `slice` path shown in the previous paragraph could be specified starting at `ai_instance`, `target`, `disk`, or `slice` , since there is only one `slice` with a `name` attribute value of 4. For example, you could use the path `slice[@name="4"]/size@val`.

If a *value* within a *path* contains forward slash characters, then that value must be enclosed in single or double quotation marks, as in `/name="pkg:/entire"`.

When the `aimanifest` call is in a shell script, values that contain quotation marks might require additional special treatment. Within a shell script, quotation marks in `aimanifest` path values might need to be escaped with a preceding backslash character (\) so that the shell does not remove or interpret the quotation marks. Check the rules of the shell you are using. The following example shows a value with a forward slash character in a `ksh93` script:

```
/usr/bin/aimanifest get software_data[name=\"pkg:/entire\"]@action
```

Most examples in this man page omit backslash escape characters because this man page does not assume that `aimanifest` is being called in a script or in a particular shell. See *Installing Oracle Solaris 11 Systems* for information about AI derived manifests scripts.

The following forms of branches show how to construct a path to an element or element attribute.

`/A`
: `A` is the tag name of an element, as in `/auto_install`. This branch specification is also called a simple branch. Paths with only simple branches are called simple paths.

`/A=`*value*
: `A` is the tag name of an element, and *value* is the value of that element, as in `/name="pkg:/entire"`.

`/A[B/C=`*value*`]`
: `A` is an element, `B` is an element that is a child of `A`, `C` is an element that is a child of `B`, and *value* is the value of the `C` element. This path form specifies the `A` element that has a grandchild element `C` that has value *value*. For example, if your AI manifest has more than one software section, you could use this form to operate on the software section that installs package `pkg:/entire`, as in the following path:

  ```
  software[software_data/name="pkg:/entire"]
  ```

`/A[@Aattr=`*value*`]`
A is an element, `Aattr` is an attribute of `A`, and *value* is the value of the `Aattr` attribute. This path form specifies the `A` element that has attribute `Aattr` with value *value*. For example, if your AI manifest defines more than one slice, you could use this form to operate on the slice that has a `name` value of 4, as in `slice[@name="4"]`

`/A[B/C@Cattr=`*value*`]`
`A` is an element, `B` is a child of `A`, `C` is a child of `B`, `Cattr` is an attribute of `C`, and *value* is the value of the `Cattr` attribute. This path form specifies the `A` element that has a grandchild element `C` that has attribute `Cattr` with value *value*. For example, if your AI manifest has more than one software section, you could use this form to operate on the software section that has a publisher section with a name value of `solaris`, as in the path `software[source/publisher@name="solaris"]`.

`/A[1]`
`/A[1]` specifies the first instance of an `A` element in the manifest. For example, if your AI manifest has more than one software section, you could use this form to operate on the second software section, as in `/auto_install[1]/ai_instance[1]/software[2]`.

This is the form of path that is returned by the `-r` option. See the "Return Paths" section.

`/A@Aattr`
This path specifies the `Aattr` attribute of the `A` element. This path does not specify the `A` element but rather the `Aattr` attribute. Use this form to set or get the `Aattr` attribute.

`/A[B/C=`*value*`]@Aattr`
This path specifies the `Aattr` attribute of the `A` element that has a grandchild element `C` that has value *value*.

`/A[B/C@Cattr=`*value*`]@Aattr`
This path specifies the `Aattr` attribute of the `A` element that has a grandchild element `C` that has attribute `Cattr` with value *value*.

`/A/B=`*value*`@Battr`
This path specifies the `Battr` attribute of the `B` element with value *value*. The `B` element is a child of the `A` element.

Return Paths With the `-r` option, the `add`, `get`, and `set` subcommands return the address of the element that was created or accessed by the subcommand. This returned address is in the form of a chain of node IDs. This returned address can be used to access the same element again, even if values associated with that element have changed.

The following examples show that the address returned by the `-r` option can be much easier to use than a path that specifies element and attribute values. Start with the following node tree:

```
auto_install
     |
ai_instance
     |
```

```
          target
            |
          disk
        attribute: whole_disk=true
            |
        disk_name
        attribute: name=data1
        attribute: name_type=volid
```

Add a new `disk` node with `name` attribute value `data2` and `name_type` attribute value `volid`:

```
       auto_install
            |
       ai_instance
            |
          target
            |
  |----------|----------|
disk                  disk
whole_disk=true       whole_disk=true
    |                     |
disk_name             disk_name
name=data1            name=data2
name_type=volid       name_type=volid
```

A new `disk_name` element with one attribute can be added easily with a single command. To add the second and third attributes, you must specify which `disk_name` element to change. Compare the following two methods for accessing the same node multiple times.

Specifying Paths By Using Values

The commands in this example specify paths using values. Note that you must assign a unique value in the first command so that you can use that value to specify a unique path in the subsequent commands. This method could yield an incorrect result if the values are changed.

```
$ aimanifest add target/disk/disk_name@name data2
$ aimanifest set \
> target/disk/disk_name[@name=data2]@name_type volid
$ aimanifest set \
> target/disk[disk_name@name=data2]@whole_disk true
```

Specifying Paths By Using Returned Paths

The most reliable way to access the same node multiple times is to save the path to the new `disk_name` element, and then use that saved path for subsequent accesses.

```
$ NewDisk=$(aimanifest add -r target/disk@whole_disk true)
$ aimanifest add ${NewDisk}/disk_name@name data2
$ aimanifest add ${NewDisk}/disk_name@name_type volid
```

The path that is returned to `$NewDisk` through the `-r` option expresses the node in terms of IDs and is free of values:

```
$ aimanifest add -r target/disk/@whole_disk true
/auto_install[1]/ai_instance[1]/target[1]/disk[2]
```

Examples To try these examples, you need to set `AIM_MANIFEST`.

```
$ export AIM_MANIFEST=/tmp/aimtest.xml
```

The minimum `AIM_MANIFEST` file that the `aimanifest` command can modify must contain both of the following pieces:

- A `!DOCTYPE` reference to a DTD that is valid for the XML manifest being developed.
- The root element for this manifest.

The following example shows the minimum `AIM_MANIFEST` manifest file for an AI manifest:

```
<!DOCTYPE auto_install SYSTEM "file:///usr/share/install/ai.dtd">
<auto_install/>
```

Usually, you will use the `aimanifest` command in a derived manifests script that operates on an existing valid AI manifest. To try these examples, you can copy `/usr/share/auto_install/manifest/default.xml` and then define `AIM_MANIFEST` to refer to this copy. Make sure the copy is writable.

EXAMPLE 1 Set the auto_reboot Attribute

```
$ aimanifest set /auto_install/ai_instance@auto_reboot false
```

EXAMPLE 2 Get the auto_reboot Value

```
$ aimanifest get /auto_install/ai_instance@auto_reboot
false
```

EXAMPLE 3 Add a Publisher by Using Values Paths

The package repository in this example is a file repository at `file:///net/host2/export/extras_repo`. The publisher is `extras`.

```
$ aimanifest add \
> software/source/publisher@name extras
$ aimanifest add \
> software/source/publisher[@name=extras]/origin@name \
> file:///net/host2/export/extras_repo
$ aimanifest set \
> software[source/publisher@name=extras]@name extras
$ aimanifest set \
> software[source/publisher@name=extras]@type IPS
```

These `aimanifest` commands result in the following AI manifest entries. The `software` element is the first element in the *path* where same-tagged siblings are allowed, so this section of XML code follows the last `software` section that is already in the output file.

```
<software name="extras" type="IPS">
  <source>
    <publisher name="extras">
```

EXAMPLE 3 Add a Publisher by Using Values Paths *(Continued)*

```
      <origin name="file:///net/host2/export/extras_repo"/>
    </publisher>
  </source>
</software>
```

EXAMPLE 4 Add a Publisher by Using Returned Paths

This example is the same as the previous example but uses a different method to achieve the same result.

```
$ SW_PATH=$(aimanifest add -r \
> /auto_install/ai_instance/software@name extras)
$ aimanifest set ${SW_PATH}@type IPS
$ PUB_PATH=$(aimanifest add ${SW_PATH}/source/publisher@name extras)
$ aimanifest add \
${PUB_PATH}/origin@name file:///net/host2/export/extras_repo)
```

EXAMPLE 5 Add a Publisher By Adding a Manifest Fragment

This example is the same as the previous example but uses a third method to achieve the same result.

Create a file named `extras.xml` with the following content:

```
<auto_install>
  <ai_instance>
    <software name="extras" type="IPS">
      <source>
        <publisher name="extras">
          <origin name="file:///net/host2/export/extras_repo"/>
        </publisher>
      </source>
    </software>
  </ai_instance>
</auto_install>
```

Even though you only want the software section, you must include the `auto_install` and `ai_instance` elements as well. You must include everything that would be required in the *path* operand. If the loaded file specifies attributes for the `auto_install` or `ai_instance` elements, then those attribute values replace existing values or are added.

Use the following command to add this `software` section to the `AIM_MANIFEST` manifest:

```
$ aimanifest load -i extras.xml
```

EXAMPLE 6 Add a Package by Using a Values Path

This example adds a package to the `software` element that has a `publisher` element with name `extras` from the previous example by specifying the publisher name as a value in the path. The example also shows using a relative path specification.

```
$ aimanifest add \
> software[source/publisher@name=extras]/software_data/name \
> pkg:/system/utils
```

This `aimanifest` command adds the following `software_data` section.

```
<software name="extras" type="IPS">
  <source>
    <publisher name="extras">
      <origin name="file:///net/host2/export/extras_repo"/>
    </publisher>
  </source>
  <software_data>
    <name>pkg:/system/utils</name>
  </software_data>
</software>
```

EXAMPLE 7 Add a Package by Using a Returned Path

This example is the same as the previous example but uses a different method to achieve the same result. Instead of specifying the name of the publisher as a value in the path, this example uses the path to the `software` element that was saved in `SW_PATH` in the example "Add a Publisher by Using Returned Paths."

```
$ aimanifest add ${SW_PATH}/software_data/name pkg:/system/utils
```

EXAMPLE 8 Validate a Manifest

Validate the `AIM_MANIFEST` manifest.

```
$ aimanifest validate
```

Exit Status

The following exit values are returned:

`0` The command was processed successfully.

`>0` An error occurred.

Files

`AIM_MANIFEST` The value of this environment variable is the location of the AI manifest that is being built.

`AIM_LOGFILE` The value of this environment variable is the location of the log file of `aimanifest` operations.

Attributes See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`system/install/auto-install/auto-install-common`
Interface Stability	Uncommitted

See Also `installadm(1M)`, `pkg(1)`

Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11 Systems*

Name distro_const – Utility for creating Oracle Solaris images and media

Synopsis

```
/usr/bin/distro_const

distro_const --help

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

Description The `distro_const` command enables users to create an image by using a specified manifest file as the blueprint for the image.

You can create a text installer image that can be used to install the Oracle Solaris operating system on either x86 systems or SPARC systems.

Or, you can create an ISO image that is comparable to a LiveCD image containing the Oracle Solaris operating system.

Alternately, you can create a SPARC AI ISO image that can be used for network installations of the Oracle Solaris OS on SPARC clients, or an x86 AI ISO image that can be used for network installations of the Oracle Solaris OS on x86 clients.

Or, you can create a custom ISO image.

The basic `distro_const` command with no options creates a full image in one step.

The command options enable users to pause and resume the image-creation process at various "checkpoints," thus enabling users to review status of the image and to check for bugs at each stage. Checkpointing saves time during builds by allowing you to bypass lengthy steps which have already been done at least once.

Note – You must assume the root role or have root privileges to run the `distro_const` command.

When using the distribution constructor, you can create only SPARC images on a SPARC system. And, you can create only x86 images on an x86 system. And, the operating system release version on your system must be the same release version as the image that you are building.

Sub-commands The `distro_const` command has the subcommand and options listed below. Also see the Examples section.

`--help`
Displays usage.

build
`distro_const build` *manifest*

The subcommand, "build", is required.

Creates a full image, using the specified manifest file as the blueprint for that image. The manifest name is required.

`-v` `distro_const build -v`
Specifies the verbose mode.

`-l` `distro_const build` [`-l`] *manifest*
Lists all valid checkpoints at which you can choose to pause or resume building an image. This command option queries the manifest file for valid checkpoints. Use the names provided by this command as valid values for the other checkpointing command options. The build subcommand is required.

The checkpoint values depend on entries in the manifest file.

`-p` `distro_const build` [`-p` *checkpoint*] *manifest*
Builds an image, but pauses building the image at the specified checkpoint name. Use the -l option to find valid names. You can combine the -r option and the -p option. The checkpoint name and manifest name are required. The build subcommand is required.

`-r` `distro_const build` [`-r` *checkpoint*] *manifest*
Resumes building the image from the specified checkpoint name. The specified name must be either the checkpoint at which the previous build stopped executing, or an earlier checkpoint. A later checkpoint is not valid. Use the -l option to determine which checkpoints are resumable. The -p option can be combined with the -r option. The checkpoint name and manifest name are required. The build subcommand is required.

`-h` `distro_const` [`-h`]
Displays command usage.

Examples

EXAMPLE 1 Create an Image Using Checkpoint Options

1. Check which checkpoints are available. The manifest name in this example is `dc_livecd.xml`.

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

   The valid checkpoint names are displayed, as in this sample output.

   ```
   Checkpoint           Resumable  Description
   ------------------------------------------
   transfer-ips-install X Transfer pkg contents from IPS
   set-ips-attributes   X Set post-install 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 archival
   grub-setup           X Set up GRUB menu
   pkg-img-mod          X Pkg image area modification
   create-iso             ISO media creation
   create-usb             USB media creation
   ```

2. Start building the image and pause at the ba-init checkpoint.

EXAMPLE 1 Create an Image Using Checkpoint Options *(Continued)*

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

3. Restart the build from the ba-init checkpoint. Finish creating the image.

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

EXAMPLE 2 Create an Image in One Step

To run a complete build of an image without pausing, use the basic `distro_const` command without checkpointing options. The manifest file name is dc_livecd.xml.

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

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	install/distribution-constructor
Interface Stability	Under Development

See Also

`dc_manifest(4)`

Creating a Custom Oracle Solaris 11 Installation Image in the OTN documentation library for the current release.

Name installadm – Manages automated installations on a network

Synopsis

```
/usr/bin/installadm [-h|--help]

installadm help [subcommand]

installadm create-service
    [-n|--service svcname]
    [-t|--aliasof existing_service]
    [-p|--publisher prefix=origin]
    [-a|--arch architecture]
    [-s|--source FMRI_or_ISO]
    [-b|--boot-args boot_property=value,...]
    [-i|--ip-start dhcp_ip_start]
    [-c|--ip-count count_of_ipaddr]
    [-B|--bootfile-server server_ipaddr]
    [-d|--imagepath imagepath]
    [-y|--noprompt]

installadm set-service
    -o|--option prop=value svcname

installadm rename-service svcname newsvcname

installadm enable svcname

installadm disable svcname

installadm delete-service
    [-r|--autoremove] [-y|--noprompt] svcname

installadm list
    [-n|--service svcname]
    [-c|--client] [-m|--manifest] [-p|--profile]

installadm create-manifest -n|--service svcname
    -f|--file manifest_or_script_filename
    [-m|--manifest manifest_name]
    [-c|--criteria criteria=value|list|range... |
    -C|--criteria-file criteriafile]
    [-d|--default]

installadm update-manifest -n|--service svcname
    -f|--file manifest_or_script_filename
    [-m|--manifest manifest_name]

installadm delete-manifest
    -m|--manifest manifest_name
    -n|--service svcname

installadm create-profile -n|--service svcname
    -f|--file profile_filename...
    [-p|--profile profile_name]
    [-c|--criteria criteria=value|list|range... |
    -C|--criteria-file criteriafile]
```

```
installadm delete-profile -p|--profile profile_name...
    -n|--service svcname

installadm export -n|--service svcname
    -m|--manifest manifest_name...
    -p|--profile profile_name...
    [-o|--output pathname]

installadm validate -n|--service svcname
    -P|--profile-file profile_filename... |
    -p|--profile profile_name...

installadm set-criteria
    -m|--manifest manifest_name
    -p|--profile profile_name...
    -n|--service svcname
    -c|--criteria criteria=value|list|range... |
    -C|--criteria-file criteriafile |
    -a|--append-criteria criteria=value|list|range...

installadm create-client
    [-b|--boot-args property=value,...]
    -e|--macaddr macaddr -n|--service svcname

installadm delete-client macaddr
```

Description 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 machine topography necessary to employ AI over the network is to have an install server, a DHCP server (this can be the same system as the install server), and the installation clients. On the install server, install services are set up to contain an AI boot image, which is provided to the clients in order for them to boot over the network, input specifications (AI manifests and derived manifests scripts), one of which will be selected for the client, and Service Management Facility (SMF) configuration profiles, zero or more of which will be selected for the client.

The AI boot image content is published as the package `install-image/solaris-auto-install`, and is installed by the `create-service` subcommand. The `create-service` subcommand is also able to accept and unpack an AI ISO file to create the AI boot image.

Install services are created with a default AI manifest, but customized manifests or derived manifests scripts (hereafter called "scripts") can be added to an install service by using the `create-manifest` subcommand. See *Installing Oracle Solaris 11 Systems* for information about how to create manifests and derived manifests scripts. The `create-manifest` subcommand also allows criteria to be specified, which are used to determine which manifest or script should be selected for an installation client. Criteria already associated with a manifest or script can be modified using the `set-criteria` subcommand.

Manifests can include information such as a target device, partition information, a list of packages, and other parameters. Scripts contain commands that query a running AI client system and build a custom manifest based on the information it finds. When AI is invoked with a script, AI runs that script as its first task, to generate a manifest.

When the client boots, a search is initiated for a manifest or script that matches the client's machine criteria. When a matching manifest or script is found, the client is installed with the Oracle Solaris release according to the specifications in the matching manifest file, or to the specifications in the manifest file derived from the matching script. Each client can use only one manifest or script.

Each service has one default manifest or script. The default is used when the criteria of no other manifest or script matches the system being installed. Any manifest or script can be designated as the default. Any criteria associated with a default manifest or script become inactive and are not considered during manifest or script selection. If a different manifest or script is later made the default, the criteria of the former default manifest or script become active again. Manifests or scripts with no criteria associated with them can only be used as default manifests or scripts. Manifests or scripts without criteria become inactive when a different manifest or script is designated the default.

System configuration profiles are complementary to manifests and scripts in that they also contain specifications for an installation. In particular, profiles are used to specify configuration information such as user name, user password, time zone, host name, and IP address. Profiles can contain variables to get configuration parameters from the install server environment or from criteria specified in the `create-profile` subcommand. In this way, a single profile file can set different configuration parameters on different clients. See the "Examples" section.

System configuration profiles are processed by `smf`(5) and conform to document format `service_bundle`(4). See `sysconfig`(1M) and *Installing Oracle Solaris 11 Systems* for more information about system configuration profiles. Each client can use any number of system configuration profiles. A particular SMF property can be specified no more than once for each client system.

If you want a specific client to use a specific install service, you can associate that client with the service by using the `create-client` subcommand. You can also use `create-client` to modify an existing client.

The `installadm` utility can be used to accomplish the following tasks:

- Set up install services and aliases
- Set up installation images
- Set up or delete clients
- Add, update, or delete manifests and scripts
- Specify or modify criteria for a manifest or script
- Export manifests and scripts

- Add or delete system configuration profiles
- Validate profiles
- Specify or modify criteria for profiles
- Export profiles
- Enable or disable install services
- List install services
- List clients for an install service
- List manifests and scripts for an install service
- List profiles for an install service

Install Server Configuration Properties

The following three properties of the `svc:/system/install/server:default` SMF service are used to configure the install server.

`all_services/networks`
A list of networks in CIDR format (for example, 192.168.56.0/24) to allow or disallow, depending on how the `all_services/exclude_networks` property is set.

Use this list of networks to specify which clients this install server serves. By default, the AI install server is configured to serve install clients on all networks that the server is connected to if the server is multihomed.

`all_services/exclude_networks`
A boolean value. If true, exclude networks specified by the `all_services/networks` property from being served by this install server. If false, include networks specified by the `all_services/networks` property.

`all_services/port`
Specifies the port that hosts the AI install services web server. By default, the web server is hosted on port 5555.

If you want to use a different port number from the default, customize the `port` property before you create any install services.

Options

The `installadm` command has the following option:

`-h`, `--help` Show the usage help message.

Sub-commands

The `installadm` command has the subcommands listed below. See also the "Examples" section below.

`help [`*subcommand*`]`
Displays the syntax for the `installadm` utility.

subcommand Displays the syntax for only the specified subcommand.

`create-service [-n|--service` *svcname*`] [-t|--aliasof` *existing_service*`]`
`[-p|--publisher` *prefix=origin*`] [-a|--arch` *architecture*`] [-s|--source` *FMRI_or_ISO*`]`
`[-b|--boot-args` *boot_property=value*`,...] [-i|--ip-start` *dhcp_ip_start*`]`
`[-c|--ip-count` *count_of_ipaddr*`] [-B|--bootfile-server` *server_ipaddr*`]`
`[-d|--imagepath` *imagepath*`] [-y|--noprompt]`

This subcommand sets up a network boot image (net image) in the specified *imagepath* directory, and creates an install service that specifies how a client booted from the net image is installed.

The AI boot image content is published as the package `install-image/solaris-auto-install`. If the `-s` option is not specified, that package is installed from the first publisher in the system's publisher preference list that provides an instance of that package. The `-s` option accepts the `pkg` specification as a full FMRI or location of an image ISO file. The resulting net image is eventually located in *imagepath*. The net image enables client installations.

Note the following specifications:

- When the first install service of a given architecture is created on an install server, an alias of that service, `default-i386` or `default-sparc`, is automatically created. This default service is used for all installations to clients of that architecture that were not added to the install server explicitly with the `create-client` subcommand. To change the service aliased by the `default-`*arch* service, use the `set-service` subcommand.

  If a `default-`*arch* alias is changed to a new install service and a local ISC DHCP configuration is found, this default alias bootfile is set as the default DHCP server-wide bootfile for that architecture.
- If you want a client to use a different install service than the default for that architecture, you must use the `create-client` subcommand to create a client-specific configuration.
- If the `-i` option and the `-c` option are used, and a DHCP server is not yet configured, an ISC DHCP server is configured.

  If an ISC DHCP server is already configured, that DHCP server is updated.

  Even when `-i` and `-c` arguments are provided and DHCP is configured, no binding exists between the install service being created and the IP range. When `-i` and `-c` are passed, the IP range is set up, a new DHCP server is created if needed, and that DHCP server remains up and running for all install services and all clients to use. The network information provided to the DHCP server has no specific bearing on the service being created.
- If the IP range requested is not on a subnet that the install server has direct connectivity to and the install server is multihomed, the `-B` option is used to provide the address of the bootfile server (usually an IP address on this system). This should only be necessary when multiple IP addresses are configured on the install server and DHCP relays are employed. In all other configurations, the software can determine this automatically.

`-n|--service` *svcname*
Optional: Uses this install service name instead of a system-generated service name. The *svcname* can consist of alphanumeric characters, underscores (_), and hyphens (-). The first character of *svcname* cannot be a hyphen.

If the `-n` option is not specified, a service name is generated automatically.

`-t|--aliasof` *existing_service*
Optional: This new service is an alternate name for *existing_service*.

`-a|--arch` *architecture*
Optional: Selects a specific variant architecture. Legal values are `i386` or `sparc`. If not specified, the variant corresponding to the server's architecture is selected.

The `-a` option only applies when the `-s` argument is a `pkg`(5) package.

`-p|--publisher` *prefix=origin*
Optional: A `pkg`(5) publisher, in the form *prefix=origin*, from which to install the client image.

If the `-p` option is not specified, then the first publisher in the system's publisher preference list that provides an instance of the package is used.

`-s|--source` *FMRI_or_ISO*
Optional: Specifies the data source for the net image. This can be either of:

- The full FMRI of a `pkg`(5) package.
- The path to an AI ISO image.

If `-s` is not specified, the `install-image/solaris-auto-install` package used is from either of:

- The publisher specified with the `-p` parameter.
- The first publisher in the system's publisher preference list that provides an instance of that package.

`-b|--boot-args` *boot_property=value*`,...`
Optional: For x86 clients only. Sets a property value in the service-specific `menu.lst` file in the service image. Use this option to set boot properties that are specific to this service. This option can accept multiple comma-separated *boot_property=value* pairs.

`-i|--ip-start` *dhcp_ip_start*
Optional: Specifies the starting IP address in a range to be added to the local DHCP configuration. The number of IP addresses is provided by the `-c` option. If a local ISC DHCP configuration does not exist, an ISC DHCP server is started.

`-c|--ip-count` *count_of_ipaddr*
Optional: Sets up a total number of IP addresses in the DHCP configuration equal to the value of the *count_of_ipaddr*. The first IP address is the value of *dhcp_ip_start* that is provided by the `-i` option.

`-B|--bootfile-server` *server_ipaddr*
Optional: Used to provide the IP address of the boot server from which clients should request bootfiles. Only required if this IP address cannot be determined by other means.

`-d|--imagepath` *imagepath*
Optional: Specifies the path at which to create the net image. If not specified, the default location, `/export/auto_install/`*svcname*, is used. A confirmation prompt is displayed unless `-y` is also specified.

`-y|--noprompt`
Optional: Suppresses any confirmation prompts and proceeds with service creation using the supplied options and any default values (see `-d`).

`set-service -o|--option` *prop=value svcname*

`-o|--option` *prop=value*
Specifies the property and value to set.

prop=value can be:

- `aliasof=`*existing_service*

  Makes *svcname* an alias of *existing_service*.

- `default-manifest=`*manifest_name*

  Designates a particular manifest or script that is already registered with a given service to be the default manifest or script for that service. Use the following command to show a list of manifests and scripts registered with this service.

  ```
  $ installadm list -n svcname -m
  ```

svcname
Required: Specifies the name of the install service whose property is being set.

`rename-service` *svcname newsvcname*
Renames the install service *svcname* to *newsvcname*. The *newsvcname* can consist of alphanumeric characters, underscores (_), and hyphens (-). The first character of *newsvcname* cannot be a hyphen.

`enable` *svcname*
Enables the *svcname* install service.

`disable` *svcname*
Disables the *svcname* install service.

`delete-service [-r|--autoremove] [-y|--noprompt]` *svcname*
Deletes an install service. Accomplishes the following:

- Deletes the manifests, profiles, client configuration files, and web server configuration for this install service.
- Deletes the image used to instantiate the service.

- If the service is a default alias and a local ISC DHCP configuration exists, the bootfile associated with this service is removed from the ISC DHCP configuration.

`-r|--autoremove`
If specified, any clients assigned to this service, and any services aliased to this service, are also removed.

`-y|--noprompt`
Suppresses any confirmation prompts and proceeds with service deletion.

svcname
Required: Specifies the install service name to delete.

`list [-n|--service` *svcname*`] [-c|--client] [-m|--manifest] [-p|--profile]`
Lists all enabled install services on a server.

`-n|--service` *svcname*
Optional: Lists information about the specific install service on a local server.

- If the `-c` option is specified, lists the client information associated with the install service.
- If the `-m` option is specified, lists the manifests and scripts associated with the install service.
- If the `-p` option is specified, lists the profiles associated with the install service.

`-c|--client`
Optional: Lists the clients of the install services on a local server.

`-m|--manifest`
Optional: Lists the manifests and scripts associated with the install services on a local server.

When `-n` is not specified, displays an abbreviated listing per service. This includes the default manifest or script, and all non-default manifests and scripts that have criteria associated with them. Criteria are not listed.

When `-n` is specified, displays all manifests and scripts for the given service, using a more complete listing format that includes criteria for each manifest. Inactive manifests, which have no associated criteria and are not designated as the default, are so marked. Criteria associated with a default manifest are marked as inactive.

`-p|--profile`
Optional: Lists the profiles associated with the install services on a local server.

When `-n` is not specified, displays an abbreviated listing per service that includes the profile names.

When `-n` is specified, displays the profiles for the requested service along with their criteria.

`create-manifest -n|--service` *svcname* `-f|--file` *manifest_or_script_filename* `[-m|--manifest` *manifest_name*`] [-c|--criteria` *criteria=value*|*list*|*range*`... | -C|--criteria-file` *criteriafile*`] [-d|--default]`

Creates a manifest or script for a specific install service, thus making the manifest or script available on the network, independently from creating a service. A non-default manifest or script can be used (can be active) only when criteria are associated with it. Criteria can be entered on the command line (`-c`) or via a criteria XML file (`-C`). Any criteria specified along with the `-d` option are temporarily ignored until the manifest or script is no longer designated as the default.

The name of the manifest is determined in the following order:

1. The *manifest_name* specified by the `-m` option, if present.
2. The value of the `ai_instance` `name` attribute, if present in the manifest.
3. The base name of the manifest or script file name.

`-n|--service` *svcname*
Required: Specifies the name of the install service this manifest or script is to be associated with.

`-f|--file` *manifest_or_script_filename*
Required: Specifies the path name of the manifest or script to add.

`-m|--manifest` *manifest_name*
Optional: Specifies the AI instance name of the manifest or script. Sets the `name` attribute of the `ai_instance` element of the manifest to *manifest_name*. The manifest or script is referred to by *manifest_name* in subsequent `installadm` commands and `installadm list` output.

`-c|--criteria` *criteria=value*|*list*|*range*`...`
Optional: Specifies criteria to be associated with the added manifest or script. See the "Criteria" section below. When publishing a default manifest, criteria are registered but kept inactive until the manifest or script is no longer designated the default. The `-c` option can be specified multiple times.

`-C|--criteria-file` *criteriafile*
Optional: Specifies the path name of a criteria XML file containing criteria to be associated with the added manifest or script. When publishing a default manifest or script, criteria are registered but kept inactive until the manifest or script is no longer designated the default.

`-d|--default`
Optional: Specifies that this manifest or script is the new default manifest or script for the service. Any criteria specified are ignored until the manifest or script is no longer the default.

`update-manifest -n|--service` *svcname* `-f|--file` *manifest_or_script_filename* `[-m|--manifest` *manifest_name*`]`

Updates a specific manifest or script that is associated with a specific install service. Any criteria or default status remain with the manifest or script following the update.

The name of the manifest is determined in the following order:

1. The *manifest_name* specified by the `-m` option, if present.
2. The value of the `ai_instance name` attribute, if present in the changed manifest and if it matches the `ai_instance name` value of an existing manifest.
3. The base name of the manifest or script file name, if it matches the `ai_instance name` attribute value in an existing manifest, or the name given by `installadm list` if it matches the name of an existing script.

The replacement manifest or script is given by the *manifest_or_script_filename*.

`-n|--service` *svcname*
Required: Specifies the name of the install service this manifest or script is to be associated with.

`-f|--file` *manifest_or_script_filename*
Required: Specifies the path name of the replacement manifest or script.

`-m|--manifest` *manifest_name*
Optional: Specifies the AI instance name of the replacement manifest or script.

`delete-manifest -m|--manifest` *manifest_name* `-n|--service` *svcname*

Deletes a manifest or script that was published with a specific install service. A default manifest or script cannot be deleted.

`-m|--manifest` *manifest_name*
Required: Specifies the AI instance name of a manifest or script as output by `installadm list` with the `-n` option.

`-n|--service` *svcname*
Required: Specifies the name of the install service this manifest is associated with.

`create-profile -n|--service` *svcname* `-f|--file` *profile_filename*`... [-p|--profile` *profile_name*`] [-c|--criteria` *criteria=value*|*list*|*range*`... | -C|--criteria-file` *criteriafile*`]`

Creates profiles for a specific install service. Criteria can optionally be associated with a profile by either entering them on the command line (`-c`) or via a criteria XML file (`-C`). Profiles created without criteria are associated with all clients of the service.

The name of the profile is determined in the following order:

1. The *profile_name* specified by the `-p` option, if present.
2. The base name of the profile file name.

Profile names must be unique for an AI service. If multiple `-f` options are used to create more than one profile with the same criteria, then the `-p` option is invalid and the names of the profiles are derived from their file names.

`-n|--service` *svcname*
Required: Specifies the name of the install service of the profile being updated.

`-f|--file` *profile_filename...*
Required: Specifies the path name of the file with which to add the profile. Multiple profiles can be specified.

`-p|--profile` *profile_name*
Optional: Specifies the name of the profile being created. Valid only for single profile creation.

`-c|--criteria` *criteria=value|list|range...*
Optional: Specifies criteria to be associated with the profiles. See the "Criteria" section below. Multiple `-c` options can be specified.

`-C|--criteria-file` *criteriafile*
Optional: Specifies the path name of a criteria XML file containing criteria to be associated with the specified profiles.

`delete-profile -p|--profile` *profile_name...* `-n|--service` *svcname*
Deletes the *profile_name* profile from the *svcname* install service.

`-p|--profile` *profile_name...*
Required: Specifies the name of the profile to delete. Multiple `-p` options can be specified.

`-n|--service` *svcname*
Required: Specifies the name of the install service of the profile being deleted.

`export -n|--service` *svcname* `-m|--manifest` *manifest_name...* `-p|--profile` *profile_name...* `[-o|--output` *pathname*`]`
Displays (exports) the specified manifest/scripts and/or profiles belonging to a service. At least one manifest/script or profile must be specified. Display goes to `stdout` unless the `-o` option redirects to a file or directory.

`-n|--service` *svcname*
Required: Specifies the install service associated with the manifest or profile to export.

`-m|--manifest` *manifest_name...*
Specifies the AI instance name of a manifest or script to export. Multiple `-m` options can be specified.

`-p|--profile` *profile_name...*
Specifies the name of a profile to export. Multiple `-p` options can be specified.

`-o|--output` *pathname*
Optional: Redirect output. The *pathname* must be a directory if multiple manifests, scripts and/or profiles are requested. The *pathname* can be a file if only one manifest, script, or profile is requested.

`validate -n|--service` *svcname* `-P|--profile-file` *profile_filename*`...` |
`-p|--profile` *profile_name*`...`
Validates specified profiles. The `validate` subcommand can be used to either validate profiles in the database (`-p`) or to validate profiles while they are being developed before their entry into the database (`-P`).

`-n|--service` *svcname*
Required: Specifies the service with which the profiles are associated.

`-P|--profile-file` *profile_filename*`...`
Specifies an external profile file to validate.

`-p|--profile` *profile_name*`...`
Specifies the name of the profile to validate.

`set-criteria -m|--manifest` *manifest_name* `-p|--profile` *profile_name*`...`
`-n|--service` *svcname* `-c|--criteria` *criteria*=*value*|*list*|*range*`... |`
`-C|--criteria-file` *criteriafile* `| -a|--append-criteria` *criteria*=*value*|*list*|*range*`...`
Updates criteria of an already published manifest/script, profile, or both. Criteria can be specified via the command line or or via a criteria XML file. Criteria must be specified with one of the mutually exclusive options, `-a`, `-c`, or `-C`.

Valid criteria are described under the `create-manifest` subcommand.

`-m|--manifest` *manifest_name*
Specifies the AI instance name of a manifest or script.

`-p|--profile` *profile_name*`...`
Specifies the name of a profile. Any number of profiles can be specified.

`-n|--service` *svcname*
Required: Specifies the name of the install service associated with this manifest/script or profile.

`-c|--criteria` *criteria*=*value*|*list*|*range*`...`
Specifies criteria to replace all existing criteria for the manifest/script or profile. See the "Criteria" section below.

`-C|--criteria-file` *criteriafile*
Specifies the path name of a criteria XML file containing criteria to replace all existing criteria for the manifest/script or profile.

`-a|--append-criteria` *criteria*=*value*|*list*|*range*`...`
Specifies criteria to be appended to the existing criteria for the manifest/script or profile. See the "Criteria" section below. If the *criteria* specified already exists, the

value|*list*|*range* of that criteria is replaced by the specified *value*|*list*|*range*.

`create-client [-b|--boot-args` *property*=*value*`,...] -e|--macaddr` *macaddr* `-n|--service` *svcname*

Accomplishes optional setup tasks for a specified client, in order to provide custom client settings that vary from the default settings used by the `create-service` subcommand. Enables the user to specify a non-default service name and boot arguments for a client. Can also be used to modify an existing client.

If the client is an x86 system and a local ISC DHCP configuration exists, the client is configured in the ISC DHCP configuration.

`-b|--boot-args` *property*=*value*`,...`

Optional: For x86 clients only. Sets a property value in the client-specific `menu.lst` file in `/etc/netboot`. Use this option to set boot properties that are specific to this client. This option can accept multiple *property*=*value* pairs.

`-e|--macaddr` *macaddr*

Required: Specifies a MAC address for the client.

`-n|--service` *svcname*

Required: Specifies the install service for client installation.

`delete-client` *macaddr*

Deletes an existing client's specific service information that was previously set up using the `create-client` subcommand.

If the client is an x86 system and a local ISC DHCP configuration exists, the client is unconfigured in the ISC DHCP configuration.

macaddr Required: Specifies the MAC address of the client to delete.

Criteria Manifests, scripts, and profiles can be used to configure AI clients differently according to certain characteristics, or criteria. Only one manifest or script can be associated with a particular client. Any number of profiles can be associated with a particular client.

The criteria values are determined by the AI client during startup.

The following AI client system criteria can be specified for both manifests/scripts and profiles unless otherwise noted.

See the "Examples" section to see how to specify criteria on the command line. For information about creating a criteria file, see *Installing Oracle Solaris 11 Systems*.

Criteria	Description
`arch`	Architecture per `uname -m`.
`cpu`	CPU class per `uname -p`.

Criteria	Description
`hostname`	Assigned hostname. Can be used only for profiles, not for manifests.
`ipv4`	IP version 4 network address.
`mac`	Hexadecimal MAC address with colon (:) separators.
`mem`	Memory size in MB per `prtconf`(1M).
`network`	IP version 4 network number.
`platform`	Platform name per `uname -i`.
`zonename`	Name of a zone per `zones`(5).

The `ipv4`, `mac`, `mem`, and `network` specifications can be expressed as ranged values separated by a hyphen (`-`). To specify no limit to one end of a range, use `unbounded`.

Any criteria that are not specifiable as a range can be specified as list of values separated by white space.

Examples

EXAMPLE 1 Set Up a New x86 Install Service From an ISO File

Set up an install server and an x86 install service for the first time. The command includes a starting IP address and total count of IP addresses, in order to configure the DHCP server.

```
# installadm create-service -n sol-11-i386-svc \
-s /export/isos/sol-11-i386.iso \
-i 172.0.0.10 -c 10 -d /export/images/soli386
```

The AI ISO image is at `/export/isos/sol-11-i386.iso`. The command sets up a net image and an install service at `/export/images/soli386` that is based on the AI ISO image. This net image enables client installations.

The starting IP address of 172.0.0.10 and ten IP addresses are added to the local ISC DHCP configuration. If a local ISC DHCP configuration does not exist, an ISC DHCP server is started.

Because this is the first `i386` service created, the `default-i386` service is automatically created and aliased to this service. The `default-i386` alias is operational, and a client booted via PXE will boot and install from the `default-i386` service.

EXAMPLE 2 Set Up a New SPARC Install Service From an ISO File

Set up a SPARC install service for the first time.

```
# installadm create-service -n sol-11-sparc-svc \
-s /export/isos/sol-11-sparc.iso \
-d /export/images/solsparc
```

EXAMPLE 2 Set Up a New SPARC Install Service From an ISO File *(Continued)*

The AI ISO image is at `/export/isos/sol-11-sparc.iso`. The command sets up a net image and an install service at `/export/images/solsparc` that is based on the AI ISO image. This net image enables client installations.

Because this is the first SPARC service created, the `default-sparc` service is automatically created and aliased to this service. The `default-sparc` alias is operational, and a SPARC client will boot and install from the `default-sparc` service.

EXAMPLE 3 Set Up an i386 Install Service From a Package Repository

```
# installadm create-service -y -n mysvc
```

On an i386 install server, this command sets up an i386 net image and install service named `mysvc` at the default image location, `/export/auto_install/mysvc`. The `-y` option provides confirmation that the default location is acceptable. Since the architecture is not specified, the service created is of the same architecture as the install server. This command assumes that a package repository on the `pkg publisher` list for the install server contains the `install-image/solaris-auto-install` package.

To specify the creation of a SPARC service on this server, use the `-a` option.

To specify the source of the `solaris-auto-install` package, use the `-p` option. For example, use the following command to specify the `ai-image` repository located at `http://example.company.com:4281` as the source of the `solaris-auto-install` package:

```
# installadm create-service -y -n mysvc \
-p ai-image=http://example.company.com:4281
```

EXAMPLE 4 Associate a Client With an Install Service

Use the following sample command to associate a client with a specific install service. The install service must already exist.

```
# installadm create-client -b "console=ttya" \
-e 0:e0:81:5d:bf:e0 -n my-i386-service
```

In this example, the command creates a client-specific setup for the system with MAC address `0:e0:81:5d:bf:e0`. This client will use the install service previously set up, named `my-i386-service`, and that service's associated net image. The command sets the boot property `console=ttya` in the client-specific `menu.lst` file in `/etc/netboot`.

EXAMPLE 5 Add a New Install Service Without Modifying the Default Service

Use the following sample command to add a new service named `my-sparc-service`, retaining existing services, and leaving the existing default unchanged.

```
# installadm create-service -n my-sparc-service \
-s /export/isos/mysparc.iso \
-d /export/ai/mysparc-image
```

EXAMPLE 6 Add a New Install Service and Update the Default Service

Use the following two sample commands to add a new service named `my-sparc-service`, retaining existing services, and making the new service the default for SPARC clients.

```
# installadm create-service -n my-sparc-service \
-s /export/isos/mysparc.iso \
-d /export/ai/mysparc-image
# installadm set-service \
-o aliasof=my-sparc-service default-sparc
```

EXAMPLE 7 Add a Custom Default AI Manifest to an Install Service

Use the following sample command to add a new manifest to the `service_092910` install service, and make it the service's default manifest. The manifest data is in `my_manifest.xml`. Future `installadm` commands will refer to this manifest as `my_manifest`.

```
# installadm create-manifest -d -f my_manifest.xml \
-m my_manifest -n service_092910
```

EXAMPLE 8 Add a Derived Manifests Script to an Install Service

Use the following sample command to add a derived manifests script named `my_script` to an existing install service named `service_092910`. Scripts are added the same way as manifests.

```
# installadm create-manifest -f my_script.py \
-m my_script -n service_092910
```

See *Installing Oracle Solaris 11 Systems* for information about how to create derived manifests scripts.

EXAMPLE 9 Replace the Default AI Manifest for an Install Service

Use the following sample command to replace the default manifest for an existing install service, `service_092910`, with a custom manifest that has already been added to the service as `my_manifest`. The manifest was added to the service by specifying `-m my_manifest` to the `create-manifest` subcommand.

```
# installadm set-service -o default-manifest=my_manifest \
service_092910
```

EXAMPLE 10 List Install Services

Use the following sample command to list the install services on a local server.

```
$ installadm list
Service Name     Alias Of         Status  Arch  Image Path
------------     --------         ------  ----  ----------
default-i386     sol-11-i386-svc  on      x86   /export/images/soli386
default-sparc    sol-11-sparc-svc on      Sparc /export/images/solsparc
sol-11-i386-svc  -                on      x86   /export/images/soli386
sol-11-sparc-svc -                on      Sparc /export/images/solsparc
```

EXAMPLE 11 List Clients Associated With an Install Service

Use the following sample command to list the clients of a specific install service on a local server.

```
$ installadm list -c -n my-x86-service
Service Name   Client Address    Arch  Image Path
------------   --------------    ----  ----------
my-x86-service 01:C2:52:E6:4B:E1 i386  /export/images/myimage
```

EXAMPLE 12 List Manifests Associated With an Install Service

Use the following sample command to list the manifests and scripts associated with a specific install service on a local server.

```
$ installadm list -m -n my-x86-service
Manifest      Status    Criteria
--------      ------    --------
manifest2               arch = i86pc
                        mem  = 4096 MB - unbounded

sparc_setup             arch = sun4v

new_default   Default   (Ignored: mem = 2048 MB - 4095 MB)

orig_default  Inactive  None
```

This example shows the following output:

- A non-default manifest with criteria (`manifest2`)
- A non-default script with criteria (`sparc_setup`)
- A default manifest with criteria that are ignored (`new_default`)
- A non-default manifest (`orig_default`) that is marked inactive because it has no criteria

EXAMPLE 13 List Profiles

Use the following sample command to list the profiles on a local server.

```
$ installadm list -p
Service Name Profile
------------ -------
sparc2       myprofile.xml
             myprofile2.xml
svc0817      profile3
svc0819      profile4.xml
             newprofile
             foo.xml
```

EXAMPLE 14 Add a Custom AI Manifest With No Name to an Install Service

Use the following sample command to add the manifest in `/export/my_manifest.xml` to `svc1` with a criterion of MAC address equaling `aa:bb:cc:dd:ee:ff`.

```
# installadm create-manifest -f /export/my_manifest.xml \
-n svc1 -c MAC="aa:bb:cc:dd:ee:ff"
```

In this example, the manifest does not contain a name attribute, so the manifest name is taken from the file name.

```
$ installadm list -m -n svc1
Manifest       Criteria
--------       --------
my_manifest    mac  = AA:BB:CC:DD:EE:FF
```

EXAMPLE 15 Add a Custom AI Manifest With a Custom Name to an Install Service

Use the following sample command to add the manifest in `/export/my_manifest.xml` to `svc1` with the criterion of IPv4 range from 10.0.2.100 and 10.0.2.199.

```
# installadm create-manifest -f /export/my_manifest.xml \
-n svc1 -m chosen_name \
-c IPV4="10.0.2.100-10.0.2.199"
```

In this example, the manifest name is taken from the `-m` option.

```
$ installadm list -m -n svc1
Manifest       Criteria
--------       --------
chosen_name    ipv4 = 10.0.2.100 - 10.0.2.199
```

EXAMPLE 16 Add a Custom AI Manifest With Name Specified In the Manifest

Use the following sample command to add the manifest in `/export/manifest3.xml` to `svc1` with criteria of 2048 MB memory or greater and an architecture of `i86pc`.

```
# installadm create-manifest -f /export/manifest3.xml \
-n svc1 -c MEM="2048-unbounded" -c ARCH=i86pc
```

In this example, the manifest name is taken from the `name` attribute of the `ai_instance` element in the manifest, as shown in the following partial manifest:

```
<auto_install>
    <ai_instance name="my_name" />
</auto_install>
```

```
$ installadm list -m -n svc1
Manifest    Criteria
--------    --------
my_name     arch = i86pc
            mem  = 2048 MB - unbounded
```

EXAMPLE 17 Add a System Configuration Profile To an Install Service

Use the following sample command to add the profile in `/export/profile4.xml` to `svc1` with criteria of any of the host names `myhost1`, `host3`, or `host6`.

```
# installadm create-profile -f /export/profile4.xml \
-n svc1 -p profile4 -c hostname="myhost1 host3 host6"
$ installadm list -p -n svc1
Profile          Criteria
-------          --------
profile4         hostname = myhost1 host3 host6
```

EXAMPLE 18 Add a System Configuration Profile For All Clients

If you do not specify criteria, then the profile is used by all clients that use the specified install service. In the following example, the created profile is used by all clients that use the `svc1` service.

```
# installadm create-profile -f /export/locale.xml -n svc1
$ installadm list -p -n svc1
Profile          Criteria
-------          --------
profile4         hostname = myhost1 host3 host6
locale
```

EXAMPLE 19 Add a System Configuration Profile With Replacement Tags

A profile can use replacement tags, which serve as placeholders for custom client configuration information that comes from either the user's environment (see `environ`(4)) or from the criteria specified in the `-c` option in the `create-profile` subcommand. Using replacement tags, a profile file can be reused for any number of different systems. In the following example, each profile is stored with the `hostname` value taken from the `-c` criteria option:

```
# installadm create-profile -p myhost1_hostname \
-f /export/hostname.xml -n svc1 -c hostname=myhost1
# installadm create-profile -p myhost2_hostname \
-f /export/hostname.xml -n svc1 -c hostname=myhost2
$ installadm list -p -n svc1
Profile              Criteria
-------              --------
myhost1_hostname     hostname = myhost1
myhost2_hostname     hostname = myhost2
```

The `hostname.xml` file contains the following line:

```
<propval name="nodename" value="{{AI_HOSTNAME}}"/>
```

The `create-profile` command makes the substitution so that the `myhost1_hostname` profile contains the following line:

```
<propval name="nodename" value="myhost1"/>
```

EXAMPLE 19 Add a System Configuration Profile With Replacement Tags *(Continued)*

Using the same `hostname.xml` input file, the `myhost2_hostname` profile contains the following line:

```
<propval name="nodename" value="myhost2"/>
```

The replacement tag, `{{AI_HOSTNAME}}`, is replaced with different values for each `create-profile` invocation because the `hostname` criteria was used and substituted into the profile. For more information about using replacement tags with profiles, see *Installing Oracle Solaris 11 Systems*.

EXAMPLE 20 Add Criteria To an Existing Manifest

Use the following sample command to append the criterion of 4096 MB memory or greater to the criteria of `manifest2` of `svc1`.

```
# installadm set-criteria -m manifest2 -n svc1 \
-a MEM="4096-unbounded"
```

EXAMPLE 21 Replace the Criteria for an Existing Manifest

Use the following sample command to replace the criteria of `manifest2` of `svc1` with the criteria specified in the file `/tmp/criteria.xml`.

```
# installadm set-criteria -m manifest2 -n svc1 \
-C /tmp/criteria.xml
```

See *Installing Oracle Solaris 11 Systems* for information about the contents of the criteria XML file.

EXAMPLE 22 Validate Profile Files Under Development

Use the following sample command to validate the profiles stored in the files `myprofdir/myprofile.xml` and `herprofdir/herprofile.xml` during their development.

```
# installadm validate -P myprofdir/myprofile.xml \
-P herprofdir/herprofile.xml -n svc1
```

EXAMPLE 23 Export Profile Contents

Use the following sample command to export the profile `myprofile.xml` in the service `svc1`.

```
$ installadm export -p myprofile -n svc1
```

EXAMPLE 24 Replace the Contents of an Existing AI Manifest

Use the following sample command to update the manifest in service `svc2` that has the manifest name, or AI instance name, `spec` with the contents of the manifest in the file `/home/admin/new_spec.xml`.

```
# installadm update-manifest -n svc2 \
-f /home/admin/new_spec.xml -m spec
```

EXAMPLE 25 Export and Update an Existing AI Manifest

Use the following sample commands to export the data of an existing manifest named `spec` in service `svc2`, and then update the manifest with modified content.

```
$ installadm export -n svc2 -m spec -o /home/admin/spec.xml
```

Make changes to `/home/admin/spec.xml`.

```
# installadm update-manifest -n svc2 \
-f /home/admin/spec.xml -m spec
```

Exit Status

The following exit values are returned:

`0` The command was processed successfully.

`>0` An error occurred.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`install/installadm`
Interface Stability	Uncommitted

See Also

`aimanifest(1M)`, `sysconfig(1M)`, `dhcp(5)`, `dhcpd(8)`, `smf(5)`, `service_bundle(4)`, `ai_manifest(4)`, `environ(5)`

Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11 Systems*

Transitioning From Oracle Solaris 10 JumpStart to Oracle Solaris 11 Automated Installer

Name js2ai – Translate JumpStart rules and profiles for use with the Automated Installer (AI).

Synopsis

```
js2ai [-h | --version]

js2ai -r | -p profile_name [-d jumpstart_dir]
    [-D destination_dir] [-lSv]

js2ai -s [-d jumpstart_dir]
    [-D destination_dir] [-Sv]

js2ai -V manifest
```

Description `js2ai` is a utility for converting Oracle Solaris 10 JumpStart `rules`, profile, and `syscfg` configuration files to a format compatible with Automated Installer (AI). This utility makes a "best effort" to translate those JumpStart keywords that can be translated to the AI context. While this conversion does not create a complete one-to-one equivalence with JumpStart, it does provide AI manifest and system configuration profile entries that can then be used as a template for creating a complete AI configuration setup based on information gathered from JumpStart configuration files.

Using `js2ai`, you can do the following:

- Process the `rules` file and the associated profiles in the current working directory.
- Process the `rules` file and the associated profiles in a specified directory.
- Process a specific profile or `sysidcfg` file.
- Direct the resulting output files to a specific directory. For more information on the `js2ai` output files, see the "Examples" and "Files" sections.

Translating Rule Keywords

TABLE 1 JumpStart Rule Keywords Translation

JumpStart Rule Keyword	AI Criteria Keyword
`arch`	`cpu`
`hostaddress`	`ipv4`
`karch`	`arch`
`memsize`	`mem`
`model`	`platform`
`network`	`ipv4`

JumpStart rule keywords not supported by `js2ai`:

```
any             installed
disksize        osname
domainname      probe
hostname        totaldisk
```

Converting Profile Keywords

TABLE 2 JumpStart Profile Keywords

JumpStart Profile Keyword	Notes
`boot_device`	The `rootdisk` is set to the specified device if not previously set by the `root_device` keyword.
`fdisk`	The value of `disk_name` must be a device. A device of `all` is not supported. The `fdisk` type must be `solaris`. A size of 0 or `delete` is not supported. If `partitioning` is `default` and the `rootdisk` has not been set, the first `fdisk solaris` partition encountered is used as the `rootdisk`.
`filesys`	The local and mirrored file systems are supported when the mount point specified is / or `swap`. No validation of the size is performed. The size specified in the resulting AI manifest might need to be adjusted to achieve a successful installation with this manifest.
`install_type`	Only the value `initial_install` is supported.
`locale`	No translation is performed. Make sure the locale specified is supported in Oracle Solaris 11.
`package`	An attempt to convert the specified package to its Oracle Solaris 11 equivalent is performed. Specifying the location of the package is not supported. Package lookups can take a considerable amount of time. If your profiles contain a long list of packages, you might want to use the `--local` flag during the conversion process.
`partitioning`	Supported types are `default` and `explicit`. Unlike JumpStart, when `partitioning default` is specified, only the disks that `js2ai` knows about are used. If no disks are specified in any keywords, the generated profile tells AI to choose which disk to use.
`pool`	If a pool is specified in a profile, the ZFS root pool is created using the specified devices. The `pool` keyword supersedes all other keywords when determining which devices to use for the ZFS root pool. No validation of the pool size, swap size, or dump size is performed. These sizes might need to be adjusted in the resulting AI manifest to achieve a successful installation with this manifest.
`root_device`	The `rootdisk` is set to the specified device.
`system_type`	Only the value `standalone` is supported.
`usedisk`	The specified device might be used to resolve the `any` or `rootdisk` device during the conversion. Any devices specified that are not used for this purpose are added to the ZFS root pool, when that pool is not mirrored.

JumpStart profile keywords not supported by `js2ai`:

```
archive_location        geo
backup_media            layout_constraint
bootenv                 local_customization
client_arch             metabd
client_root             no_master_check
client_swap             no_content_check
cluster                 num_clients
dontuse                 patch
forced_deployment
```

How the System's Root Disk is Determined During Profile Translation

Since `js2ai` does not have access to the actual system a profile references during the profile translation process, `js2ai` attempts to determine what the root disk is during translation using a process that matches JumpStart as much as possible.

The `js2ai` tool performs the following steps to determine what device to use for the root disk.

Stage	Action
1	If the `root_device` keyword is specified in the profile, `js2ai` sets `rootdisk` to the device on which the slice resides.
2	If `rootdisk` is not set and the `boot_device` keyword is specified in the profile, `js2ai` sets `rootdisk` to the boot device.
3	If `rootdisk` is not set, `partitioning default` is specified, and a `solaris fdisk` entry is encountered, `js2ai` sets `rootdisk` to the specified `disk_name`.
4	If `rootdisk` is not set and a `filesys` *cwtxdysz size* `/` entry is specified in the profile, `js2ai` sets `rootdisk` to the *cwtxdysz* disk specified in the entry.
5	If `rootdisk` is not set and a `usedisk` *disk_name* entry is specified in the profile, `js2ai` sets `rootdisk` to the *disk_name* disk specified in the entry.
6	If `rootdisk` is not set and the following specification is encountered in the profile where *size* is not 0 or `delete` and *disk_name* is not `all`, then `rootdisk` is set to this *disk_name*. `fdisk` *disk_name* `solaris` *size*
7	If `rootdisk` is not set, any occurrence where the device is specified as `rootdisk` generates a conversion error.

How the any Device Is Translated During Profile Translation

The `js2ai` tool performs the following steps to determine what device to use when the `any` keyword is specified.

Stage	Action
1	If the any device is specified and the keyword action specified (non-mirrored pool, or `filesys` with a / mount point), the any device is set to `rootdisk` if `rootdisk` is set.
2	If the any device has not been translated and a `usedisk` statement exists in the profile, the any device is set to the device specified by the `usedisk` statement.
3	If the any device has not been translated and the action where the any device is specified causes the ZFS root pool to be created, AI chooses the device. This is not applicable when a mirrored pool is specified.

How the ZFS Root Pool is Determined During Profile Translation

The `js2ai` tool performs the following steps to determine what device to use for the ZFS root pool. Once the ZFS root pool is determined, subsequent definitions encountered are flagged as errors if they conflict with the ZFS root pool that has already been determined.

Stage	Action
1	If the profile specifies the `pool` keyword, `js2ai` sets the ZFS root pool to the devices specified by the `pool` keyword.
2	If the ZFS root pool has not been determined and the profile specifies a `filesys` with a mount point of /, the ZFS root pool is created using the devices specified.
3	If the ZFS root pool has not been determined and all keywords in the profile have been processed, and if `rootdisk` is set, the ZFS root pool is created using the `rootdisk` device.
4	If the ZFS root pool has not been determined and the partition type is `default`, AI chooses the device to use for the ZFS root pool.
5	If the ZFS root pool has not been determined and no errors have occurred during processing, AI chooses the device to use for the ZFS root pool.
6	If the ZFS root pool is not a mirrored pool and one or more `usedisk` devices that were specified have not been used for a `rootdisk` or any device translation, those disks are added to the ZFS root pool.

Converting sysidcfg Keywords

TABLE 3 JumpStart sysidcfg Keywords

`sysidcfg` Keyword	Notes
`keyboard`	No translation is performed. Make sure the keyboard specified in the `sysidcfg` file is supported in Oracle Solaris 11.
`name_service`	Supports values `None`, `DNS`, `NIS`, and `LDAP`. NIS+ name services are translated as NIS.
`network_interface`	Only a single interface is supported. Limited support for `PRIMARY`. Only the first interface encountered in the `sysidcfg` file is processed.

TABLE 3 JumpStart sysidcfg Keywords *(Continued)*

sysidcfg Keyword	Notes
root_password	No translation is necessary.
security_policy	Supports value: None
service_profile	Supports value: limited_net
system_locale	No translation is performed. Make sure the locale specified in the sysidcfg file is supported in Oracle Solaris 11.
terminal	No translation is performed. Make sure the terminal type specified in the sysidcfg file is supported in Oracle Solaris 11.
timeserver	Supports value: localhost
timezone	No translation is necessary.

JumpStart sysidcfg keywords not supported by js2ai:

nfs4_domain

Options The js2ai command has the following options. The use of these options is illustrated in the "Examples" section.

-h, --help
Show the usage help message.

--version
Show the version number of the js2ai utility.

-d *jumpstart_dir*, --dir *jumpstart_dir*
Specify the location of the rules and profile files or the sysidcfg file.

-D *destination_dir*, --dest *destination_dir*
Specify the location for the output files.

-l, --local
When searching for Image Packaging System (IPS) equivalents for the package keyword value in a JumpStart profile, search the IPS packages installed on the host system rather than the packages in an IPS package repository.

-p *profile_name*, --profile *profile_name*
Convert the specified JumpStart profile and generate a manifest for the profile processed. In this case, no criteria file is needed or generated.

-r, --rule
Convert rules and associated profiles and generate a manifest for each profile processed.

-s, --sysidcfg
Process the sysidcfg file and output the results to sc_profile.xml.

`-S`, `--skip`
Skip validation.

`-v`, `--verbose`
Provide details on the actions that occurred during processing.

`-V` *filename*
Validate the specified AI manifest file or SMF system configuration profile file. AI criteria validation is not supported.

Error Report The `js2ai` tool generates an error report when one or more errors occurs during the conversion.

```
# js2ai -r
                    Process  Unsupported  Conversion  Validation
Name      Warnings  Errors   Items        Errors      Errors
--------  --------  -------  -----------  ----------  ----------
rules            0        0            2           0           -
profile1         0        0            0           2           1

Conversion completed. One or more failures occurred.
For errors see ./js2ai.log
```

The report contains one entry for each file in which `js2ai` encountered an error. To generate an error report even when no errors occur, specify `-v` or `--verbose`.

The report tells you what type of errors occurred in what files. Five error types are defined: Warnings, Process Errors, Unsupported Items, Conversion Errors, and Validation Errors.

Warnings
Items in these messages are not required to be corrected. For example, you might receive a warning message that information such as host name or root password was not provided, and default values will be used.

Process Errors
These errors refer to problems that prevent `js2ai` from processing a file or a line within the file. Process errors typically occur when the file has a syntax error.

Unsupported Items
These items refer to a line that `js2ai` does not support. Changing the value associated with a keyword might eliminate this error.

Conversion Errors
These errors refer to a condition that prevents `js2ai` from processing a line. These errors should be manually corrected, or the offending lines should be removed from the file.

Validation Errors
These errors refer to the errors that occurred when the generated manifest was validated against the schema definition used by AI. These errors must to be corrected before the manifest can be used by AI.

The `js2ai.log` file indicates what error occurred on what line.

```
# cat js2ai.log
rules: line 4: unsupported keyword: disksize
rules: line 4: unsupported keyword: installed
net924_sun4c: line 4: unsupported keyword: cluster
net924_sun4c: line 5: unsupported keyword: num_clients
net924_sun4c: line 6: unsupported keyword: client_swap
net924_sun4c: line 7: unsupported keyword: client_arch
upgrade: line 1: unsupported value for 'install_type' specified: upgrade
```

If a validation error of the manifest occurs, the `js2ai.log` file contains a pointer to the log file that contains the validation errors, as shown in the following example:

```
Validation Errors:
    profile1: manifest validation of
    ./AI_profile1/profile1.xml failed.
    For details see ./AI_profile1/profile_validation.log
```

Conversion Strategy

Recommended Strategy for Rule and Profile Conversion

A one-to-one conversion between JumpStart and AI does not exist. The following steps provide a general procedure for performing the conversion.

1. The `js2ai` utility attempts to flag any errors it encounters, but `js2ai` assumes the rules, profiles, and `sysidcfg` files that are being converted are valid.
2. Copy the JumpStart configuration directory of `rules`, profile, and `syscfg` configuration files to an Oracle Solaris 11 system that has the `install/installadm` package installed.
3. In the JumpStart configuration directory that you copied to the Oracle Solaris 11 system in step 2, run the `js2ai` conversion tool.

   ```
   # js2ai -rS
   ```

   This command performs a conversion operation on the `rules` file and the profiles referenced by the `rules` file. Each profile referenced in the `rules` file is processed against the AI client provisioning manifest, `/usr/share/auto_install/manifest/default.xml`. This step creates a directory named `AI_`*profile_name* for each profile specified in the JumpStart `rules` file. The `AI_`*profile_name* directory contains one or more AI manifests for the translated profile in the form *profile_name*`${arch}.xml`. See the "Files" section for more information.

   The `-S` option skips the validation sequence. Validation is done in step 5.
4. If the message "Successfully completed conversion" is output, skip to step 5. Otherwise, examine the `js2ai.log` file and follow these steps:
   a. Correct any process errors.
   b. Remove any lines from the `rules` and profile files that are listed as Unsupported Items.
   c. Examine the conversion errors and correct the errors if possible. Otherwise, remove the lines that are causing the errors.

d. Examine any warning messages and make sure no corrections are necessary.

e. Repeat step 3 until no processing errors, unsupported items, and conversion errors are reported.

5. Rerun `js2ai` without the `-S` option.

   ```
   # js2ai -r
   ```

   If any validation errors occur for any of the processed profiles, the resulting AI manifest must be manually corrected. Examine the `js2ai.log` file for details of the failure. See the AI documentation for information about AI manifests.

6. Convert any `sysidcfg` files that are associated with this JumpStart configuration.

   For each `sysidcfg` file, execute the following command:

   ```
   # js2ai -sS -d sysidcfg_dir
   ```

   For each `sysidcfg` file processed, this step creates an AI system configuration profile file named `sc_profile.xml` in the directory where the `js2ai` command was invoked. Use the `-D` option to specify a different directory for the `sc_profile.xml` file.

7. If the message "Successfully completed conversion" is output, skip to step 8. Otherwise, examine the `js2ai.log` file and follow these steps:

   a. Correct any process errors.

   b. Remove any lines from the `sysidcfg` file that are listed as unsupported items.

   c. Examine the conversion errors and correct the errors if possible. Otherwise, remove the lines that are causing the errors.

   d. Examine any warning messages and make sure no corrections are necessary.

   e. Repeat step 6 until no processing errors, unsupported items, and conversion errors are reported.

8. Rerun `js2ai` without the `-S` option.

   ```
   # js2ai -s -d sysidcfg_dir
   ```

   If any validation errors occur for any of the processed `sysidcfg` files, the resulting AI system configuration profile must be manually corrected. Examine the `js2ai.log` file for details of the failure. See the AI documentation for information about system configuration profiles.

9. The `js2ai` conversion process is complete. Perform a manual verification of the resulting criteria, AI manifest, and system configuration profile files. The disk space requirements for an Oracle Solaris 11 installation are different from the disk space required for an Oracle Solaris 10 installation. Make sure the disk space allocated in your AI manifests meets the requirements of Oracle Solaris 11.

10. Configure AI to use the newly generated files. Add the newly generated criteria, AI manifest, and system configuration profile files to an existing AI install service.

Use the `installadm` command with the `create-manifest` subcommand to add each AI manifest with criteria for selecting that manifest. Each client can use only one AI manifest.

```
# installadm create-manifest -n ai_service_name \
-f manifest_file -m manifest_name \
-C criteria_file
```

Use the `create-profile` subcommand to add each profile with criteria for selecting that configuration profile. Each client can use one or more system configuration profiles.

```
# installadm create-profile -n ai_service_name \
-f profile_file -p profile_name \
-C criteria_file
```

See the AI documentation and the `installadm`(1M) man page for information about configuring AI install services.

Examples

EXAMPLE 1 Processing a JumpStart Configuration

The following command processes the JumpStart rules and profiles in the current directory. The output is also placed in this directory.

```
# js2ai -r
```

EXAMPLE 2 Processing a Specific JumpStart Directory

The following command processes the JumpStart rules and profiles from the specified directory and places the output files in the same directory.

```
# js2ai -r -d /export/jumpstart
```

For more information about the output files, see Example 4 and the "Files" section.

EXAMPLE 3 Processing a Profile in a Specific JumpStart Directory and Separate Destination Directory

The following command processes the JumpStart `rules` and profile files from the `/export/jumpstart` directory and places the output files in `/export/output`.

```
# js2ai -p profile1 -d /export/jumpstart -D /export/output
```

EXAMPLE 4 Example Input and the Resulting Output for a Specified Rule and Its Profile

Rule:

```
arch sparc && karch sun4u && \
    model 'SUNW,Serverblade1'  -   profile    -
```

Profile:

```
install_type    initial_install
pool mypool auto auto auto c1t0d0s0
```

Conversion command:

EXAMPLE 4 Example Input and the Resulting Output for a Specified Rule and Its Profile *(Continued)*

```
# js2ai -r -d /jumpstart -D /tmp/output
```

Output files:

```
/tmp/output/AI_profile/profile.x86.xml
/tmp/output/AI_profile/profile.sparc.xml
/tmp/output/AI_profile/criteria-1.xml
```

Two manifest files are created, one for SPARC and one for x86, even though the `rules` file specifies the CPU type as SPARC. During the conversion process, rules and profiles are processed independently of one another.

EXAMPLE 5 Adding Generated Files to an AI Install Service

This example adds the manifest and criteria to an existing service, using the files generated in Example 4.

Files:

```
/tmp/output/AI_profile/profile.sparc.xml
/tmp/output/AI_profile/criteria-1.xml
```

`installadm` command:

```
# installadm create-manifest -n svc-name \
-f /tmp/output/AI_profile/profile.sparc.xml \
-m sparc_profile \
-C /tmp/output/AI_profile/criteria-1.xml
```

EXAMPLE 6 Processing a sysidcfg File

The following command processes the `sysidcfg` file in the current directory and outputs the resulting SMF system configuration profile as `sc_profile.xml` in the same directory.

```
# js2ai -s
```

Exit Status

The following exit values are returned:

`0` All the files were processed successfully.

`>0` An error occurred.

Files

output_directory`/AI_${`*profile_name*`}`
Directory that contains all the corresponding files that have been translated to the new AI syntax associated with the profile.

output_directory`/AI_${`*profile_name*`}.${`*arch*`}.xml`
The manifest file created as a result of translating the profile. `${`*arch*`}` can be one of these three values: `sparc`, `x86`, or `generic`. A manifest file that is in the form `${`*profile_name*`}.generic.xml` can be used to install both x86 and SPARC systems.

output_directory/AI_${*profile_name*}/criteria-*rule_number*.xml
The criteria-*rule_number*.xml file produced corresponds to the rule in the rules file, and the *rule_number* is the rule number based on its position in the rules file. This criteria file can then be used with the -C option to the installadm command.

Since more than one rule can specify the same profile, more than one criteria file can exist in each directory, but only one instance of the ${*profile_name*}.${*arch*}.xml file should exist in each output directory.

Note – If the -p option is used, no criteria file is produced for the profile that is processed. Criteria files are only generated when used with the -r option.

output_directory/js2ai.err
This file contains a stack trace of an unexpected condition that occurred during processing. This file is not typically created.

output_directory/js2ai.log
This file contains a log of the files processed and any errors found during processing.

output_directory/sc_profile.xml
This file is the SMF system configuration profile that is generated when the -s option is used to convert a sysidcfg file.

Attributes

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	install/js2ai
Interface Stability	Uncommitted

See Also

installadm(1M), pkg(1)

Transitioning From Oracle Solaris 10 JumpStart to Oracle Solaris 11 Automated Installer

Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11 Systems*

REFERENCE

File Formats

Name ai_manifest – Automated installation manifest file format

Synopsis `/usr/share/install/ai.dtd.1`

Description Automated Installer (AI) provides a customizable, hands-free installation mechanism for Oracle Solaris and uses an XML-based file format as the description of the installation parameters. This installation parameters file is called an AI manifest. The installation can be customized in various ways such as disk layout and the software to be installed on the system.

The AI manifest has the following sections:

- Automated installation settings. Specifies settings used during the installation.
- Disk layout. Specifies the disk layout for the installation.
- Software. Specifies the software packages to be installed.
- Boot configuration (x86 only). Specifies how to configure the GRUB boot menu.
- Other configuration. Specifies other configuration components to be installed onto the system.

These sections are described in more detail below.

To create a new AI manifest, use a copy of the template or default manifest from the relevant install service image. For example, if the install service image is located at *imagepath*, the following files are available:

imagepath`/auto_install/manifest/default.xml`
The original default AI manifest for this install service.

imagepath`/auto_install/manifest/ai_manifest.xml`
An annotated, sample AI manifest with example customizations.

You can use the `installadm export` command to retrieve a copy of any manifest that already exists in an install service.

AI manifests are also used for installing non-global zones using the `zoneadm install` command. An AI manifest file can be passed to this command to customize the zone installation. Only a subset of AI manifest specifications applies to installing non-global zones. These specifications are noted in the sections below.

Complementing the AI manifest are Service Management Facility (SMF) configuration profiles. These profiles specify the system configuration for the installed system such as hostname, networking, and root and initial user account settings.

For more information about install services, AI manifests, and configuration profiles, see the `installadm(1M)` man page and Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11 Systems*. For information about the configuration profile file format, see `smf(5)`.

Automated Installation Settings

The `ai_instance` element has the following attributes:

`name` The name of this manifest instance.

`http_proxy` The HTTP proxy to use to access remote files during the installation. Examples of remote files accessed during installation are software packages in an Image Packaging System (IPS) package repository. The value of `http_proxy` is an HTTP URI such as `http://myproxy.mycompany.com:8080/`.

This attribute is not applicable when installing a non-global zone and is ignored if provided.

`auto_reboot` The flag that specifies whether to automatically reboot after installation. The default value of `auto_reboot` is `false`. When `auto_reboot` is `false`, the installation waits for manual intervention to reboot.

When `auto_reboot` is `true`, on a successful installation, the machine automatically reboots into the newly installed boot environment.

This attribute is not applicable when installing a non-global zone and is ignored if provided.

The following example demonstrates how to use the `ai_instance` element:

```
<auto_install>
    <ai_instance name='default' auto_reboot='true'
      http_proxy='http://myproxy.mycompany.com:8080/'>
    <!-- target and software sections -->
    </ai_instance>
</auto_install>
```

Disk Layout

AI enables a range of disk specification, varying from completely automatic selection of the installation target to fine-grained control of the disk layout.

The `target` element specifies the disk layout. The default disk layout when no `target` element is specified has the following characteristics:

- The whole of one disk is used to install the Oracle Solaris OS. This disk is usually the boot disk or first disk.
- For x86, an `fdisk` partition is allocated that consumes the full disk contents. See the `fdisk(1M)` man page for more information about `fdisk` partitions.
- A single slice 0 is allocated that is the full size of the disk for SPARC or the full size of the `fdisk` partition for x86.
- A single root pool is created that uses the complete slice 0.
- A swap volume and a dump volume are created in the root pool if space is available.

The `target` element has the following structure:

```
<!-- zero or one target element -->
<target>
    <!-- zero or more disk elements -->
    <disk ...>
    </disk>
    <logical ...>
        <!-- zero or more zpool elements -->
        <zpool ...>
        </zpool>
    </logical>
</target>
```

Child elements of the `target` element enable you to specify disks and logical layout.

Disk specifications are not applicable when installing a non-global zone and are ignored if provided.

Some disk layout elements have a size sub-element. The `size` element has the following format:

`<size val="`*size*`" start_sector="`*start_sector*`"/>`

The *start_sector* value is a numeric value that specifies the desired start sector for the new partition or slice. If the *start_sector* attribute is omitted, the installer searches for the first location large enough to contain the specified *size*.

Values for *size* are numeric with one of the following suffixes:

- `s` or `sec`: sectors
- `b`: bytes
- `k` or `kb`: kilobytes (2^{10})
- `m` or `mb`: megabytes (2^{20})
- `g` or `gb`: gigabytes (2^{30})
- `t` or `tb`: terabytes (2^{40})
- `p` or `pb`: petabytes (2^{50})
- `e` or `eb`: exabytes (2^{60})
- `z` or `zb`: zettabytes (2^{70})

The remainder of this section describes the `disk` and `logical` elements in detail.

Installation Location

If you do not specify a location for installing the Oracle Solaris OS on a client, AI selects a default location for that client.

The default location for the installation is the first disk found on each client that meets the size requirement. If the size of a disk is greater than or equal to the required size, the installer selects that disk as the installation location. If the size of the disk is less than the required size,

the installer checks the next disk. If no disk is found that meets the size requirement, the automated installation fails for that client. The install log at `/system/volatile/install_log` shows the details of the disk selection process for that system.

The `disk` section of the `target` section specifies the installation location.

Disk specifications are not applicable when installing a non-global zone and are ignored if provided.

Disks can be selected using one of the following types of selection criteria:

- Group 1: Deterministic criteria such as disk name or IP address. Use the `<disk_name>` sub-element as described in "Target Device Name" below or the `<iscsi>` sub-element as described in "ISCSI Target Device" below.
- Group 2: Nondeterministic criteria such as disk size or vendor. Use the `<disk_prop>` sub-element as described in "Target Device Properties" below.
- Group 3: Keyword criteria such as the `boot_disk` keyword. Use the `<disk_keyword>` sub-element as described in "Target Device Keyword" below.

You can specify criteria from only one of these three groups. If you use Group 2 selection criteria, you can specify multiple criteria. For example, you can specify both size and vendor. If you use Group 1 selection criteria, you can specify only one of those criteria.

Target Device Name

Use the `disk_name` element to specify the target device name for a device that is not an iSCSI device. The `disk_name` element has the following attributes:

`name` The `name` attribute specifies the name of the target device.

`name_type` The `name_type` attribute specifies the type of the target device name. The `name_type` attribute can have one of the following values:

`ctd`: Controller Target Disk Name
This is a CTD name such as `c0t0d0` or an MPXIO name such as `c0t2000002037CD9F72d0`. This type of name is commonly seen when running the `format(1M)` command.

```
<disk_name name="c0t0d0" name_type="ctd"/>
```

This is the default target device name type if the `name_type` attribute is omitted.

`volid`: Volume Identifier
This is the volume identifier as can be set by the `format`(1M) command.

```
<disk_name name="MY_BOOT_DISK" name_type="volid"/>
```

`devpath`: Device Path
This is the device path relative to the `/devices` directory.

```
<disk_name
  name="/devices/pci@0,0/pci10de,375@f/pci108e,286@0/disk@0,0"
  name_type="devpath"/>
```

`devid`: Device Identifier
This is the device identifier as found in the "Device Id" in the output from the `iostat(1M)` command with the `-iEn` options.

```
<disk_name
  name="id1,sd@TSun_____STK_RAID_INT____F0F0F0"
  name_type="devid"/>
```

`receptacle`: Receptacle Identifier
This is the receptacle value from a CRO (Chassis, Receptacle, Occupant) configuration as found in the output from the `croinfo(1M)` command with the `-o cR` option.

```
<disk_name name="SYS/1" name_type="receptacle"/>
```

ISCSI Target Device

Use the `iscsi` element to specify an iSCSI disk as the installation target. The `iscsi` element has the following attributes:

`source` The `source` attribute specifies the source of the iSCSI configuration data. The `source` attribute can have the following values:

`manifest`
This value refers to this AI manifest. This is the default if no value is specified for the `source` attribute.

When the `source` attribute is omitted or the value of the `source` attribute is `manifest`, the `target_lun` and `target_ip` attributes must be specified.

`dhcp`
This value refers to the use of DHCP where the iSCSI information is sourced by specifying the information in the DHCP `rootpath` parameter.

When the value of the `source` attribute is `dhcp`, do not specify any other `iscsi` attributes.

```
<iscsi source="dhcp"/>
```

`target_name` The `target_name` attribute specifies the iSCSI Qualified Name (IQN) or the Extended Unique Identifier (EUI) of the iSCSI target, as shown in the following example:

```
iqn.1986-03.com.sun:02:a4a694bc-6de2-ee50-8979-e25ba29acb86
```

If the `target_name` attribute is not provided, AI uses `iscsiadm(1M)` in `sendtargets` mode.

```
<iscsi target_lun="0" target_ip="192.168.1.34"/>
```

If the `target_name` attribute is provided, AI uses static discovery.

```
<iscsi target_name="iqn.1986-03.com.sun:02:a4a694bc-6de2-ee50-8979-e25ba29acb86"
    target_lun="0" target_ip="192.168.1.34"/>
```

`target_lun` If an iSCSI target provides more than one LUN, specify which LUN to use by specifying an integer value for `target_lun`. LUN numbers are indexed from 0. To specify the first LUN, specify a `target_lun` value of 0.

If only one LUN is provided, this attribute can be omitted.

`target_port` If not specified, the default `target_port` of 3260 (the iSCSI standard port) is used. This attribute enables you to specify an alternative port number.

`target_ip` The value of this attribute is the IP address of the server.

```
<iscsi target_lun="0" target_ip="192.168.1.34"/>
```

Target Device Properties

Use the `disk_prop` element to specify properties of the target device. Multiple properties can be specified. AI attempts to find a best match based on the criteria provided.

Use attributes of the `disk_prop` element to specify the target properties. The `disk_prop` element has the following attributes:

`dev_type`: Device Type
The type of the target disk. Possible values include SCSI, ATA, and USB. This value is not case sensitive.

`dev_vendor`: Device Vendor
The vendor as shown by the `inquiry` menu option of the `format(1M)` command.

```
<disk_prop dev_vendor="Sun"/>
```

`dev_chassis`: Device Chassis
The chassis value from a CRO (Chassis, Receptacle, Occupant) configuration as found in the output from the `croinfo(1M)` command with the `-o cA` option.

```
<disk_prop dev_chassis="SYS"/>
```

`dev_size`: Device Size
The minimum size for the disk. The value is a number with a size unit.

```
<disk_prop dev_size="100gb"/>
```

The `disk_prop` element allows specification of multiple attributes at the same time to further constrain the disk search. The following example limits the selection of a disk to a Hitachi drive with a size of at least 100 GB.

```
<disk_prop dev_vendor="HITACHI" dev_size="100gb"/>
```

Target Device Keyword

The `disk_keyword` element can be used to specify the system's boot disk as the target disk.

```
<disk_keyword key="boot_disk"/>
```

The only value supported for the `key` attribute is `boot_disk`.

Whole Disk, Partitions, and Slices

The simplest way to lay out a disk is to use the entire disk for installation by setting the `whole_disk` attribute to `true`.

For more complex disk layouts, you can specify partitions (for x86 systems only) and slices.

The `disk` element has the following attributes:

`whole_disk` The default value of this attribute is `false`. When `whole_disk` is `false`, partitions or slices must be defined. Any existing partitions or slices are retained unless you remove them by specifying the `delete` value for the `action` attribute of the partition or slice.

When `whole_disk` is `true`, any existing partitions or slices are removed unless you retain them by specifying the `preserve` value for the `action` attribute of the partition or slice.

The following example specifies using the entire disk for installation:

```
<disk whole_disk="true">
  <disk_name name="c0t0d0" name_type="ctd"/>
</disk>
```

`in_zpool` The `in_zpool` attribute links this disk to a ZFS pool defined in the `logical` section of the AI manifest. The value of the `in_zpool` attribute must match the value of the `name` attribute of the corresponding `zpool` element.

If the `in_zpool` attribute is specified here, then do not specify `in_zpool` for any subordinate partitions or slices.

`in_vdev` The `in_vdev` attribute links this disk to a virtual device defined in the `logical` section of the AI manifest. The value of the `in_vdev` attribute must match the value of the `name` attribute of the corresponding `vdev` element.

If the `in_vdev` attribute is specified here, then do not specify `in_vdev` for any subordinate partitions or slices.

Partitions

Partitions can only be specified when installing to an x86 system. If partitions are specified for a SPARC system, the installation fails. The `partition` element has the following attributes:

`name`
: The `name` attribute is the `fdisk` partition number. Values 1, 2, 3, and 4 are primary partitions. If one of the primary partitions is an extended partition, values 5 through 32 can be specified for logical partitions.

  The `name` attribute is required unless the specified `action` is `use_existing_solaris2`.

`action`
: The `action` attribute can have the following values:

  `create`
  : This is the default action for a partition. The `create` action tells the installer to create a partition with the specified name. If a partition with the same name already exists, that existing partition is deleted first.

  `delete`
  : The `delete` action tells the installer to delete the named partition. If the named partition does not exist, the `delete` action is skipped and a warning message is output.

  `preserve`
  : The `preserve` action tells the installer to leave the named partition untouched. This action is commonly used if another operating system is installed at another location on the same disk.

  `use_existing_solaris2`
  : The `use_existing_solaris2` action tells the installer to use an existing Solaris2 partition. The installer searches for the existing Solaris2 partition.

    When `use_existing_solaris2` is specified, the `name` and `part_type` attributes are ignored.

`part_type`
: The `part_type` is the `fdisk` partition type. The default value is 191, which is the partition type for a Solaris2 partition. See the `fdisk`(1M) command for more information about possible partition types.

`in_zpool`
: The `in_zpool` attribute links this partition to a ZFS pool defined in the `logical` section of the AI manifest. The value of the `in_zpool` attribute must match the value of the `name` attribute of the corresponding `zpool` element.

  If the `in_zpool` attribute is specified, then do not specify `in_zpool` for the associated `disk` element or any subordinate `slice` elements.

`in_vdev`
: The `in_vdev` attribute links this partition to a virtual device defined in the `logical` section of the AI manifest. The value of the `in_vdev` attribute must match the value of the `name` attribute of the corresponding `vdev` element.

  If the `in_vdev` attribute is specified, then do not specify `in_vdev` for the associated `disk` element or any subordinate `slice` elements.

Partitions can have a `size` sub-element to specify the size of the partition. See the beginning of the "Disk Layout" section for details about how to use the `size` element.

The following example creates a 10 GB Solaris2 partition using default attribute values:

```
<disk>
    <disk_name name="c0t0d0" name_type="ctd"/>
    <partition name="1">
        <size val="10gb"/>
    </partition>
</disk>
```

If the size is not specified, the size of the parent element is used.

The `preserve`, `delete`, and `use_existing_solaris2` actions do not need a `size` specification.

Slices

For an x86 system, slices must be contained within a partition definition.

The `slice` element has the following attributes:

`name` The `name` attribute is the slice number. The value can be 0 through 7.

`action` The `action` attribute can have the following values:

- `create` This is the default action for a slice. The `create` action tells the installer to create a slice with the specified name. If a slice with the same name already exists, that existing slice is deleted first.
- `delete` The `delete` action tells the installer to delete the named slice. If the named slice does not exist, the `delete` action is skipped and a warning message is output.
- `preserve` The `preserve` action tells the installer to leave the named slice untouched. This action is commonly used when data exists from a previous installation.

`is_swap` The default value of this attribute is `false`. When `is_swap` is `false`, the installer creates a swap volume in the root pool.

When `is_swap` is `true`, the named slice is used as a swap device. When `is_swap` is `true`, do not use the `in_zpool` or `in_vdev` attributes.

`force` The default value of this attribute is `false`.

When `force` is `true`, the installer ignores any existing slice that might already be in use (for example, a slice that is used in an existing ZFS storage pool) and continues to perform the specified action on the named slice.

`in_zpool` The `in_zpool` attribute links this slice to a ZFS pool defined in the `logical` section of the AI manifest. The value of the `in_zpool` attribute must match the value of the `name` attribute of the corresponding `zpool` element.

If the `in_zpool` attribute is specified, then do not specify `in_zpool` for the associated `partition` or `disk` elements.

`in_vdev` The `in_vdev` attribute links this slice to a virtual device defined in the `logical` section of the AI manifest. The value of the `in_vdev` attribute must match the value of the `name` attribute of the corresponding `vdev` element.

If the `in_vdev` attribute is specified, then do not specify `in_vdev` for the associated `partition` or `disk` elements.

Slices can have a `size` sub-element to specify the size of the slice. See the beginning of the "Disk Layout" section for details about how to use the `size` element. If the size is not specified, the size of the parent element is used.

The following example creates a 20 GB slice using default attribute values and a 4 GB swap slice for a SPARC system:

```
<disk>
    <disk_name name="c0t0d0" name_type="ctd"/>
    <slice name="0">
        <size val="20gb"/>
    </slice>
    <slice name="1" is_swap="true">
        <size val="4gb"/>
    </slice>
</disk>
```

The following example is the same example for an x86 system:

```
<disk>
    <disk_name name="c0t0d0" name_type="ctd"/>
    <partition name="1">
        <slice name="0">
            <size val="20gb"/>
        </slice>
        <slice name="1" is_swap="true">
            <size val="4gb"/>
        </slice>
    </partition>
</disk>
```

Swap and Dump A swap slice can be explicitly defined by setting the `is_swap` attribute of the `slice` element to `true`, as shown in "Slices" above.

A volume in a pool can be explicitly defined as a swap volume or a dump volume by setting the `use` attribute of the `zvol` element to `swap` or `dump`, as shown in "ZFS Volumes" below.

By default, a swap volume and a dump volume are automatically created if space is available.

On low memory systems, a swap slice can be preferable to a swap volume since volumes incur a small memory overhead.

If you want to explicitly specify swap or dump and do not want swap or dump volumes automatically created, set the following attributes of the `logical` element to `true`:

`noswap` The default value of this attribute is `false`. When `noswap` is `false`, if space allows, the installer automatically creates a swap volume in the root pool.

When `noswap` is `true`, no swap volume is automatically created.

`nodump` The default value of this attribute is `false`. When `nodump` is `false`, if space allows, the installer automatically creates a dump volume in the root pool.

When `nodump` is `true`, no dump volume is automatically created.

ZFS Storage Pools

Use the `logical` section of the `target` section to specify any number of ZFS storage pools.

Multiple pools can be defined by using the `zpool` sub-element of the `logical` element. Only one of these pools can be the root pool. The installation fails if multiple root pools are defined.

If a `zpool` element defines a root pool, and no target disks, partitions, or slices are specified in the AI manifest, then the installer selects a target as described in "Installation Location" above. This selection is automatically assigned to the root pool.

If target disks, partitions, or slices are specified in the AI manifest, then the `zpool` must be associated with at least one of these disks, partitions, or slices. To make this association, use the `in_zpool` attribute of the `disk` element, the `partition` element, or the `slice` element.

The `zpool` element has the following attributes:

`name` This is the name of the new pool. This value must be a name that can be passed to the `zpool create` command.

This name could be used as the value of an `in_zpool` attribute of a `disk`, `partition`, or `slice` element to define that disk, partition, or slice as a constituent device in the `zpool`.

`action` The `action` attribute can have the following values:

`create`
This is the default action for a `zpool`. The `create` action tells the installer to create a pool with the specified name.

`delete`
The `delete` action tells the installer to delete the named pool.

`preserve`
The `preserve` action tells the installer to leave the named pool unmodified. This action can be specified only for a non-root pool.

The value of the `action` attribute must be `preserve` in the following cases:

- The value of the `action` attribute of any subordinate `filesystem` is `preserve`.
- The value of the `action` attribute of any subordinate `zvol` is `preserve`.
- The value of the `action` attribute of any subordinate `zvol` is `use_existing`.

`use_existing`
The `use_existing` action tells the installer to install to the existing root pool. Any existing volumes or file systems (datasets) are retained.

`is_root` The default value of this attribute is `false`. When `is_root` is `false`, a data pool is defined.

When `is_root` is `true`, the new boot environment is created in the named pool.

`mountpoint` The `mountpoint` attribute specifies the mount point of the top level file system of the pool. The default mount point is */poolname*. The mount point must be an absolute path.

To set ZFS properties on the new pool, use the `pool_options` element. Similarly, to set ZFS properties on the automatically created ZFS dataset, use the `dataset_options` element. Both the `pool_options` and `dataset_options` elements have an `option` sub-element. Each `option` element has a `name` attribute and a `value` attribute. The properties set with these name/value pairs are subject to the same restrictions that the `zpool`(1M) command enforces. The following example shows how to set these properties:

```
<logical>
    <zpool name="rpool" is_root="true">
        <pool_options>
            <option name="listsnaps" value="on"/>
            <option name="delegation" value="off"/>
        </pool_options>
        <dataset_options>
            <option name="atime" value="on"/>
            <option name="compression" value="on"/>
        </dataset_options>
    </zpool>
</logical>
```

Any number of virtual device redundancy groups (vdev element), ZFS datasets (filesystem element), or ZFS volumes (zvol element) can be defined for a pool. Boot environments (be element) can be specified for a pool. The following sections describe the vdev, filesystem, zvol, and be elements.

Virtual Device Redundancy Groups

Use the vdev element to define the size or structure of a zpool. You can specify multiple vdev elements, each with a different redundancy type.

If a zpool contains more than one vdev element, then you must use the in_vdev attribute on any disk, partition, or slice elements that are defined with in_zpool attributes.

You can omit the in_zpool attribute on a disk, partition, or slice if the vdev name is unique throughout the AI manifest.

If a zpool contains only one vdev element, you can omit the in_vdev attribute on a disk, partition, or slice.

The vdev element has the following attributes:

name
: This is the name of the new vdev.

  This name should be used as the value of an in_vdev attribute of a disk, partition, or slice element to define that disk, partition, or slice as a constituent device in the vdev

redundancy
: The redundancy attribute can have the following values:

  mirror
  : This is the default value. When redundancy is mirror or is not specified, all devices contained are considered to be mirrors of each other.

  raidz, raidz1, raidz2, raidz3
  : Devices in a group with one of these values are used to define a RAIDZ grouping.

  spare
  : Devices in this group are seen as hot spares in case of failure.

  cache
  : Devices in this group provide caching for the pool.

  log, logmirror
  : Devices in this group are used for logging. If logmirror is specified, the devices are mirrors.

  none
  : When redundancy is none, no redundancy is defined. If multiple devices are included in this group, these devices are striped.

A root pool can be defined as having only one of the following configurations:

- A `redundancy` type of `none` with one device. Multiple devices are not supported in this configuration.
- A `redundancy` type of `mirror` with multiple devices.

To add a device to a `vdev`, use the `in_zpool` and `in_vdev` attributes of a `disk`, `partition`, or `slice` element. The following example specifies a root pool named `rpool` that is mirrored over two disks:

```
<disk whole_disk="true" in_zpool="rpool" in_vdev="mirrored">
    <disk_name name="c0t0d0" name_type="ctd"/>
</disk>
<disk whole_disk="true" in_zpool="rpool" in_vdev="mirrored">
    <disk_name name="c1t0d0" name_type="ctd"/>
</disk>
<logical>
    <zpool name="rpool" is_root="true">
        <vdev name="mirrored" redundancy="mirror"/>
    </zpool>
</logical>
```

You can omit one of the `in_zpool` or `in_vdev` attributes if the pool or virtual device they refer to is unambiguous.

File Systems (Datasets)

Use the `filesystem` element to define ZFS file systems or datasets within a ZFS pool.

The `filesystem` element has the following attributes:

`name` This is the name of the new `filesystem`, relative to the `zpool`. For example, if the `filesystem` is named `export` within a `zpool` named `rpool`, the ZFS dataset name is `rpool/export`.

If the `in_be` attribute of the `filesystem` is set to `true`, this name is relative to the root dataset of the boot environment.

`action` The `action` attribute can have the following values:

- `create` This is the default action for a `filesystem`. The `create` action tells the installer to create a file system with the specified name.
- `delete` The `delete` action tells the installer to delete the named file system.
- `preserve` The `preserve` action tells the installer to leave the named file system unmodified. If `preserve` is specified for the `filesystem`, then `preserve` should be specified for the associated `zpool`.

`mountpoint` The `mountpoint` attribute specifies the mount point of the new file system. If a mount point is not specified, the file system inherits the mount point from its parent.

`in_be` The default value of this attribute is `false`. When `in_be` is `false`, the new dataset is shared among all boot environments.

When `in_be` is `true`, a separate copy of this new dataset is created within each boot environment. When `in_be` is `true`, the value of the `name` attribute is relative to the root dataset of the boot environment.

Use the `options` sub-element to set the ZFS dataset properties on a `filesystem`. Any editable ZFS file system property can be set. Use of the `options` element for a `filesystem` is similar to the use of the `dataset_options` element for a `zpool`, as shown in the following example:

```
<logical>
    <zpool name="rpool" is_root="true">
        <filesystem name="export">
            <options>
                <option name="compression" value="off"/>
                <option name="dedup" value="on"/>
            <options>
        </filesystem>
    </zpool>
</logical>
```

A child `filesystem` inherits any property set on a parent `filesystem` unless that property is explicitly set differently. This is the default behavior of ZFS file systems.

ZFS Volumes

Use the `zvol` element to define ZFS volumes within a ZFS pool. A `zvol` is typically used for swap or dump devices, but it can have other uses.

The `zvol` element has the following attributes:

`name` This is the name of the new ZFS volume.

`action` The `action` attribute can have the following values:

`create` This is the default action for a `zvol`. The `create` action tells the installer to create a ZFS volume with the specified name.

`delete` The `delete` action tells the installer to delete the named volume.

`preserve` The `preserve` action tells the installer to leave the named `zvol` unmodified. If `preserve` is specified for the `zvol`, then `preserve` should be specified for the associated `zpool`.

`use_existing` If this value is specified for a swap or dump device, the existing volume is re-used. If `use_existing` is specified for the `zvol`, then `preserve` should be specified for the associated `zpool`.

`use` The `use` attribute can have the following values:

`none` This is the default value. When `use` is `none`, the `zvol` is created but not used during the installation.

`swap` When `use` is `swap`, the `zvol` is created and used as a swap device. The `zvol` is also used as a swap device during the installation.

`dump` When `use` is `dump`, the `zvol` is created and used as a dump device. The `zvol` is also used as a dump device during the installation.

Use the `size` sub-element to specify the size of the `zvol` See the beginning of the "Disk Layout" section for details about how to use the `size` element.

Use the `options` sub-element to set ZFS volume options on a `zvol`. Use of the `options` element for a `zvol` is similar to the use of the `dataset_options` element for a `zpool`, as shown in the following example:

```
<logical>
    <zpool name="rpool" is_root="true">
        <zvol name="swap">
            <options>
                <option name="compression" value="off"/>
            <options>
        </zvol>
    </zpool>
</logical>
```

Boot Environments

Use the `be` element to specify how the boot environment is created during the installation.

The `be` element has one attribute:

`name` This is the name of the new boot environment that is created by the installer. If the `be` element is not specified, the default name for this boot environment is `solaris`.

The installer makes use of the auto-naming feature provided by the boot environment subsystem. When installing into an existing target area (for example, when installing a zone), a boot environment with the name specified by the `be` element `name` attribute might already exist. If the specified boot environment name already exists, this name is used as a base to generate a new name. For example, if `be` is not specified, and a boot environment named `solaris` already exists, the new boot environment is named `solaris-`*n*, where *n* is the first integer in counting order that forms a boot environment name that does not already exist.

A boot environment is created as a ZFS dataset and can have ZFS properties set on it. Use the `options` sub-element to set ZFS properties on a boot environment, as shown in the following example:

```
<logical>
    <zpool name="rpool" is_root="true">
        <be name="installed_be">
            <options>
                <option name="compression" value="on"/>
                <option name="dedup" value="on"/>
            <options>
        </be>
    </zpool>
</logical>
```

Software The `software` element specifies software to install. The `software` section specifies the following information:

- The type of the software source
- The location of the source
- The names of software packages to install or uninstall
- Optional components of software to install
- Image properties
- SSL keys and certificates required to access the IPS repository

The `software` element has the following attributes:

`name` This is the name of the `software` instance. This name must be unique among all `software` instances in this AI manifest.

`type` This is the type of the software source.

The `type` attribute can have one of the following values. The default value if `type` is not specified is `IPS`.

- `IPS`: IPS package repository
- `P5I`: IPS package file
- `SVR4`: SVR4 packages
- `CPIO`: `cpio` archive

The `software` element has the following structure:

```
<!-- one or more software elements -->
<software>
    <!-- zero or one destination element
         This element is only used when type is IPS or P5I.
    -->
    <destination>
        <!-- image properties and
```

```
                optional software components
            -->
        </destination>
        <!-- one or more source elements
             IPS type: only one source element
        -->
        <source>
            <!-- one or more publisher or dir elements
                 IPS, P5I, and SVR4 types:
                     one or more publisher/origin elements
                 CPIO types: one or more dir elements
            -->
        </source>
        <!-- zero or more software_data elements
             At least one software_data element must have an
                 action of install.
             P5I type: zero software_data elements
        -->
        <software_data>
            <!-- one or more name elements -->
        </software_data>
    </software>
```

IPS Installations

The default installation type if the `type` attribute is not specified is `IPS`.

For installations of type IPS, only a single `source` element can be specified.

Use the `source` element to specify which publishers to use for installing the packages. Multiple publishers can be specified. Each publisher must have at least one origin. Each publisher can have multiple origins and mirrors.

The order in which publishers are defined in the AI manifest is the order in which the publishers are searched for IPS packages to install and the order in which the publishers are set in the installed system.

When installing a non-global zone, the system repository is used by the zone. Any publishers specified in the AI manifest are added in the order in which they appear in the AI manifest, after the publishers provided by the system repository. See `pkg(1)` and `pkg.sysrepo(1m)` for more information about the system repository.

The following example specifies multiple publishers, one of which has a mirror as well as an origin:

```
<software type="IPS">
  <source>
    <publisher name="solaris">
      <origin name="http://pkg.oracle.com/solaris/release"/>
      <mirror name="http://localpkg.mycompany.com/solaris"/>
```

```
    </publisher>
    <publisher name="internal-software">
      <origin name="http://internalsoft.mycompany.com/"/>
    </publisher>
  </source>
</software>
```

Use the `software_data` element to specify packages to install or uninstall. The `action` attribute can have one of the following two values:

`install` Installs the IPS packages specified in the `name` sub-elements. This is the default if the `action` attribute is not specified. At least one `software_data` element must have an action of install.

`uninstall` Removes the IPS packages specified in the `name` sub-elements.

Other values of the `action` attribute are not supported for IPS installations.

For each of these actions, one or more packages can be specified in the `name` element, as shown in the following example:

```
<software_data> <!-- defaults to install action -->
    <name>pkg:/entire</name>
    <name>pkg:/group/system/solaris-large-server</name>
</software_data>
<software_data action="uninstall">
    <name>pkg:/unwanted/pkg</name>
</software_data>
```

P5I Installations A `.p5i` file is a file that describes IPS publishers, packages, and possibly mirrors.

To specify one or more `.p5i` files to be processed, provide the files as origins in the `publisher` element, as shown in the following example:

```
<software type="P5I">
    <source>
        <publisher>
            <origin name="/somewhere/image1.p5i"/>
            <origin name="/somewhere/image2.p5i"/>
        </publisher>
    </source>
</software>
```

If this AI manifest does not also have an IPS type software section, make sure your `.p5i` files specify origins.

Specification of packages to install is not supported for P5I installations. Therefore, `software_data` elements are not supported in a `software` element of type `P5I`.

SVR4 Installations

For a SVR4 transfer, a directory containing SVR4 package subdirectories or a SVR4 package datastream file must be specified using a file directory path or a FILE URI. The SVR4 package datastream file can also be specified using an HTTP URI.

```
<software type="SVR4">
    <source>
        <publisher>
            <origin name="/somedir"/>
        </publisher>
    </source>
</software>
```

The `software_data` element is used to specify the action to be performed. The `action` attribute can have one of the following two values:

`install` Copies the files from the source to the new boot environment. This is the default if the `action` attribute is not specified. At least one `software_data` element must have an action of install.

`uninstall` Removes the files from the new boot environment.

Other values of the `action` attribute are not supported for SVR4 installations.

For each of these actions, one or more packages can be specified in the `name` element, as shown in the following example:

```
<software type="SVR4">
    <source>
        <publisher>
            <origin name="/somedir"/>
        </publisher>
    </source>
    <software_data> <!-- defaults to install action -->
        <name>ORGpackage1</name>
        <name>ORGpackage2</name>
    </software_data>
    <software_data action="uninstall">
        <name>ORGpackage8</name>
    </software_data>
</software>
```

CPIO Installations

For a CPIO transfer, a source directory must be specified. The destination directory is set to the mount point for the new boot environment during the installation.

```
<software type="CPIO">
    <source>
        <dir path="/somedir"/>
    </source>
</software>
```

The `software_data` element is used to specify the action to be performed. The `action` attribute can have one of the following values:

`install` Copies the files from the source to the new boot environment. This is the default if the `action` attribute is not specified. At least one `software_data` element must have an action of install.

Use the `name` element to specify the files or directories to be copied. Paths specified in the `name` element are relative to the source.

```
<software_data>
  <!-- defaults to install action -->
  <name>path/relative/to/source</name>
  <name>another/path/relative/to/source</name>
</software_data>
```

`uninstall` Removes files from the new boot environment.

Use the `name` element to specify the files or directories to be removed. Paths specified in the `name` element are relative to the destination.

```
<software_data action="uninstall">
    <name>path/relative/to/destination</name>
</software_data>
```

Optional Software Components and Image Properties

Use the `destination` element and the `image` sub-element to specify the following information:

- Optional components of software to install
- Image properties
- SSL keys and certificates

The `destination` section only applies to IPS and P5I installation types. A `destination` element can have only one `image` sub-element.

SSL Keys and Certificates

Use attributes of the `image` element to specify SSL keys and certificates that are required for publishers using client SSL authentication. The key and certificate specified here apply to the first publisher defined in this AI manifest.

`ssl_key` This attribute maps to the following `pkg` command:

```
pkg set-publisher -k ssl_key
```

The value of the `ssl_key` attribute is the *ssl_key*. See the pkg(1) man page for more information about the `pkg set-publisher` command.

`ssl_cert` This attribute maps to the following `pkg` command:

```
pkg set-publisher -c ssl_cert
```

The value of the `ssl_cert` attribute is the *ssl_cert*.

Optional Software Components

Use the `facet` sub-element of the `image` element to specify optional software components to install. Facets are not separate software packages but are optional components of any given software package such as locales, documentation, and development files such as files with debug information. You can save space by specifying that you only want to install one or two languages, for example. See the pkg(1) man page for more information about IPS facets.

The `facet` element has a boolean `set` attribute and a value that is the name of an IPS facet.

```
<facet set="true|false">facet_name</facet>
```

The following example specifies that only German and English facets of packages should be installed. The example first specifies that no locales should be installed and then specifies that German and English locales should be installed:

```
<destination>
    <image>
        <!-- de-select all locales -->
        <facet set="false">facet.locale.*</facet>
        <!-- specify specific locales to install -->
        <!-- install German and English only -->
        <facet set="true">facet.locale.de</facet>
        <facet set="true">facet.locale.de_DE</facet>
        <facet set="true">facet.locale.en</facet>
        <facet set="true">facet.locale.en_US</facet>
    </image>
</destination>
```

Image Properties

Use the `property` sub-element of the `image` element to specify IPS image properties for the new image this installation creates.

The `property` element has a boolean `val` attribute and a value that is the name of a property.

```
<property val="true|false">property_name</property>
```

See the "Image Properties" section of the pkg(1) man page for information about what properties can be set.

Boot Configuration (x86 Only)

The AI manifest can be used to modify how the GRUB boot menu is configured on the installed system.

This section is not applicable to zone installations and is ignored when installing a non-global zone.

Use the boot_mods element and the boot_entry sub-element to modify the GRUB boot menu.

The boot_mods element has the following attributes:

title The value of the title attribute is the base title of boot entries specified by boot_entry sub-elements of this boot_mods element. This attribute value overrides the name automatically generated from the first line of /etc/release or from the install media.

timeout The value of the timeout attribute is the number of seconds to wait before the default boot_entry of this boot_mods element is selected.

Only the title attribute can be set on SPARC systems. All other settings in this section are ignored for SPARC systems.

Use the boot_entry sub-element to add one or more menu items to the boot menu. These menu items are in addition to any menu items that are automatically generated by the installer.

The boot_entry element has the following attributes:

default_entry If this boolean value is set to true, then this menu item is the default option selected on boot. The default value of this attribute is false.

If multiple boot_entry elements have default_entry set to true, then the last such entry is the default option selected on boot.

insert_at This attribute can be set to one of the following two values:

- end Place the entry at the end of the generated boot menu. This is the default placement.
- start Place the entry at the beginning of the generated boot menu.

The boot_entry menu item is then defined by the following sub-elements:

title_suffix This element is required. This element defines text to be added to the end of the title specified in the boot_mods element.

kernel_args This element is optional. This element is a string of values passed to the kernel by the boot loader.

The following example specifies a boot menu entry named “Boot Testing Default Boot Entry” that is the last entry on the menu and is automatically selected after 20 seconds:

```
<boot_mods title="Boot Testing" timeout="20">
    <boot_entry default_entry="true">
        <title_suffix>Default Boot Entry</title_suffix>
```

```
    </boot_entry>
</boot_mods>
```

Other Configuration

The `configuration` element supports non-global zone configurations. When installing a global zone system, the zone configurations specified in the AI manifest are used to install non-global zones onto the system after the global zone has been installed.

The `configuration` element has the following attributes:

`type` The type of configuration to install. The only type supported by AI is `zone`.

`name` A name given to the configuration. This name must be unique across all configuration elements in an AI manifest. For configurations of type `zone`, this name is also used as the `zonename` for the zone.

`source` The location from which AI downloads the configuration file for this configuration element. The value can be an HTTP or FILE URI specification. For configurations of type `zone`, this value should point to a zone configuration file as produced from the `zonecfg export` command.

The following specification installs `zone1` on the installation clients:

```
<configuration type="zone" name="zone1"
    source="http://myserver.com/configs/zone1/config"/>
```

For more information about configuring and installing zones, see Chapter 12, "Installing and Configuring Zones," in *Installing Oracle Solaris 11 Systems*.

Files

`/usr/share/auto_install/manifest/default.xml`
A default system installation specification with no customizations. This AI manifest is provided on the system for reference only. To create a new AI manifest, use the copy of this file from the relevant install service image. See the "Description" section for information about copying this file from an install service.

`/usr/share/auto_install/manifest/zone_default.xml`
A default zone installation with no customization. This file is used as the default manifest by the `zoneadm install` command to install non-global zones.

`/usr/share/auto_install/manifest/ai_manifest.xml`
A template AI manifest with details commented out. This file provides examples of some customizations that can be performed. This file is provided on the system for reference only. To create a new AI manifest, use the copy of this file from the relevant install service image. See the "Description" section for information about copying this file from an install service.

Attributes

See `attributes(5)` for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	`system/install/auto-install/auto-install-common`
Interface Stability	Uncommitted

See Also `installadm`(1M), `beadm`(1M), `pkg`(1), `grub`(5), `prtconf`(1M), `format`(1M), `zfs`(1M), `zpool`(1M), `pkg.sysrepo`(1m), `smf`(5), `zoneadm`(1M), `zonecfg`(1M)

Part III, "Installing Using an Install Server," in *Installing Oracle Solaris 11 Systems*

Name dc_manifest – Customizing the manifest files for the distribution constructor

Synopsis

```
The following manifest files can be used to build various Oracle Solaris images.
These manifests are included in the distribution-constructor package.

To build x86 Oracle Solaris LiveCD images:
/usr/share/distro_const/dc_livecd.xml

To build x86 automated installation images:
/usr/share/distro_const/dc_ai_x86.xml

To build SPARC automated installation images:
/usr/share/distro_const/dc_ai_sparc.xml

To build x86 text installation images:
/usr/share/distro_const/dc_text_x86.xml

To build SPARC text installation images:
/usr/share/distro_const/dc_text_sparc.xml
```

Description The distribution constructor (DC) can be used to build Oracle Solaris installation images.

DC XML manifest files are used as input to the distribution constructor. These manifests define the image that the distribution constructor builds. Per the list above, you can use different manifests to build different kinds of images.

Use the `distro_const` command to build images, referencing a manifest file in the command.

If you want to customize the image specifications, copy a manifest file, customize the copy, and use the copy as input for the `distro_const` command when you build the image.

At a minimum, you need to edit the target element in the manifest to specify the location of the build area where the image can be constructed. And, you need to edit the software name element to specify the publisher and repository location that contain the packages needed to build the image.

Manifest Sections The manifests include the following primary elements.

Note – The default elements and attributes provided below vary depending on which manifest is used.

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

This element provides the default name, Oracle_Solaris_Text_X86, for the image that you plan to build. You can use this name, or provide a unique name for your image.

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 enter the proxy location. For example,

```
<distro name="Oracle_Solaris_Text_SPARC" add_timestamp="false"
http_proxy="http://example.com">
```

<boot_mods>

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

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

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

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>
```

Since a title sub-element is not included in this example, the default is used. The default title is the first line of `/etc/release`.

The title_suffix is a required sub-element, a text string to be appended to the entry title. An optional kernel_args sub-element passes kernel arguments to the boot loader.

Optional attributes for the boot_entry element include:

default_entry	Set this attribute to “true” to make this boot entry the default. If more than one entry is set to “true”, the last entry defined as such will override preceeding entries.
insert_at	Set value to “start” or “end” to indicate insertion point relative to other boot entries.

<target>

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 enter a valid dataset location.

See the following example.

```
<target>
      <logical>
        <zpool action="use_existing" name="rpool">
```

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

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

This section specifies where the distribution constructor can get packages to download and use to build the image.

Image Packaging System (IPS) publishers provide packages at one or more package repositories.

In the source element in this section, edit the publisher name and origin name elements to specify which publisher to use and where the package repository is located. 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.

See the following example.

```
<source>
  <publisher name="publisher1">
    <origin name="http://example.oracle.com/primary-pub"/>
    <mirror name="mirror.example.com"></mirror>
  </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>
```

Note – This element also includes a destination tag which specifies the data mountpoint to be used during the build of the image. Changing the destination attribute is not recommended.

<software_data action="install">

This 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 LiveCD image.

Each name tag lists one package name or the name of a group package that contains many packages.

```
<software_data action="install" type="IPS">
  <name>pkg:/entire</name>
  <name>pkg:/server_install</name>
  <name>pkg:/system/install/text-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 'entire' reference using the following format:

```
<name>pkg:/entire@0.5.11-0.build#</name>
```

Use the following command to check which versions are available.

```
# pkg list -af entire
```

Note – Do not remove the 'entire' entry. 'entire' is an incorporation used to manage multiple packages.

<software_data action="uninstall" type="IPS">

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

In the following example, 'server_install' is the name of a group package that contains numerous individual packages.

```
<software_data action="uninstall" type="IPS">
  <name>pkg:/server_install</name>
</software_data>
```

You could uninstall a group package. Uninstalling a group package means that only the group definition is actually uninstalled. The individual packages that were previously installed as part of that group are not uninstalled. However, you can uninstall those individual packages without uninstalling the group package. Retaining the group package can be useful for ongoing reference.

You can also use the name tag to uninstall an individual package. Append additional packages to be uninstalled at the end of the uninstall section.

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

This element affects a system after that system has been installed with the image created using the distribution constructor.

In the source element, use the publisher name and optional mirror name tags to specify where the installed system can access additional packages to download and install. See the following example.

```
<source>
  <publisher name="solaris">
    <origin name="http://pkg.oracle.com/solaris/release/"/>
  </publisher>
</source>
```

<software name="ba-init">

This element lists the files and directories to be installed or uninstalled in the boot archive for the image that is built. See the comments in the manifest file for information.

Caution – Modifying the boot archive contents could render the system unbootable.

<execution stop_on_error="true">

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.

Each checkpoint name tag includes the mod-path attribute which specifies where the checkpoint script is located.

Use the `distro_const`(1M) command options to control pausing and restarting the build process at particular checkpoints.

Some of the checkpoint tags include arguments with default values provided. See the manifest comments for details.

If you create a custom script to be used during the building of an image, you must add a checkpoint name tag pointing to the script location.

See the following example about how to add a new checkpoint name tag to point to a custom script.

A user creates a custom script, `/tmp/myscript.sh`, to run in the build process after the default transfer-ips-checkpoint.

In order to point to the new script, add the following tag to the manifest after the transfer-ips-checkpoint name, in order to point to the new script.

```
<checkpoint name="custom-script"
 desc="my new script"
 mod_path="solaris_install/distro_const/checkpoints/custom_script"
 checkpoint_class="CustomScript">
   <args>/tmp/myscript.sh arg1 arg2/{PKG_IMAGE_PATH}</args>
</checkpoint>
```

Where 'arg1' and 'arg2' are optional arguments the script takes.

The values of '{PKG_IMAGE_PATH}' or '{BOOT_ARCHIVE}' are replaced by the `distro_const` utility during execution with `<ZFS Dataset>/build_data/pkg_image` and `<ZFS Dataset>/build_data/boot_archive`, respectively.

Note – Multiple custom-script checkpoints may be specified in a DC manifest. Each checkpoint must have a unique name.

<configuration name="pre-pkg-img-mod" type="sysconf" source="/etc/svc/profile/generic_limited_net.xml">

The configuration name element in the manifest lists SMF service profiles that are applied to the media during the image construction process. These SMF services specify which services will be running, or not running, on the booted media. The profiles are applied in the order specified in this element.

This element would rarely be modified.

Attributes

See attributes(5) for descriptions of the following attributes:

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	install/distribution-constructor package
Interface Stability	Under Development

See Also

`distro_const`(1M), `pkg`(1)

Creating a Custom Oracle Solaris 11 Installation Image in the OTN documentation library for the current release.